1 CONCEPTUL DE ALGEBRĂ MULTISORTATĂ

Conceptul de algebră multisortată apare în jurul anului 1970 prin generalizarea algebrelor universale. Noul concept mai este cunoscut şi sub numele algebre universale eterogene.

Deoarece **datele cu care lucrăm nu sunt toate la fel**, ele sunt clasificate în mai multe tipuri sau sorturi. Acesta este principalul fapt care a dus la apariţia algebrelor multisortate şi în particular a mulţimilor multisortate.

Algebrele multisortate au fost generalizate conducând la algebrele ordonat sortate.

1.1 Mulţimi şi funcţii multisortate

Fixăm mulţimea S a sorturilor.

Definiţia 1.1 O familie de mulţimi $M = \{M_s\}_{s\in S}$ indexată de S se numeşte **mulţime S-sortată.**

Observăm că aceeaşi literă este folosită atât pentru întreaga mulţime M cât şi pentru toate componentele acesteia, M_s unde $s \in S$.

Fie $M = \{M_s\}_{s\in S}$ o mulţime S-sortată. Dacă $s \in S$ şi $m \in M_s$ spunem că elementul m are sortul s sau că s este sortul elementului m.

Conceptele uzuale pentru mulţimi se extind pe componente la mulţimile S-sortate aşa cum se vede din exemplele de mai jos

$$\{M_s\}_{s\in S} \subseteq \{N_s\}_{s\in S} \text{ dacă şi numai dacă } (\forall s \in S) M_s \subseteq N_s,$$
$$\{M_s\}_{s\in S} \cup \{N_s\}_{s\in S} = \{M_s \cup N_s\}_{s\in S},$$
$$\{M_s\}_{s\in S} \cap \{N_s\}_{s\in S} = \{M_s \cap N_s\}_{s\in S},$$
$$\{M_s\}_{s\in S} \times \{N_s\}_{s\in S} = \{M_s \times N_s\}_{s\in S}.$$

O funcţie între două mulţimi S-sortate duce un element din prima mulţime într-un element de acelaşi sort din a două mulţime.

Definiţia 1.2 O **funcţie S-sortată**

$$f : M \longrightarrow N$$

este o familie de funcţii $f = \{f_s\}_{s\in S}$ unde $f_s : M_s \longrightarrow N_s$, componenta de sort s, este pentru orice $s \in S$ o funcţie uzuală.$\odot$

Ca şi în cazul mulţimilor S-sortate, operaţiile cu funcţiile S-sortate se fac pe componente. Dacă $f : M \longrightarrow N$ şi $g = \{g_s\}_{s\in S} : N \longrightarrow P$ sunt funcţii S-sortate atunci compunerea lor

$$f; g = \{f_s; g_s\}_{s\in S} : M \longrightarrow P$$

este definită pentru orice $s \in S$ prin

$$(f; g)_s = f_s; g_s.$$

Mai detaliat $(f; g)_s(x) = g_s(f_s(x))$ pentru orice $s \in S$ şi $x \in M_s$. Semnul ; folosit pentru compunere este inspirat din limbajele de programare. Mai observăm scrierea diagramatică $f; g$ utilizată pentru compunere în opoziţie cu scrierea clasică $g \circ f$

$$\mathrm{M} \xrightarrow{\mathrm{f}} \mathrm{N} \xrightarrow{\mathrm{g}} \mathrm{P}, \quad \mathrm{M} \xrightarrow[\mathrm{f;g}]{} \mathrm{P}$$

Compunerea funcţiilor S-sortate este asociativă. Dacă şi $h = \{h_s\} : P \longrightarrow R$ este funcţie S–sortată, atunci folosind asocitivitatea compunerii funcţiilor uzuale pentru orice $s \in S$

$$((f; g); h)_s = (f; g)_s; h_s = (f_s; g_s); h_s = f_s; (g_s; h_s) = f_s; (g; h)_s = (f; (g; h))_s.$$

Prin urmare $(f; g); h = f; (g; h)$.

Pentru orice mulţime S-sortată M funcţia ei identitate $1_M : M \longrightarrow M$ este definită prin $(1_M)_s = 1_{M_s}$ pentru orice $s \in S$, unde 1_{M_s} este funcţia identitate a mulţimii M_s. Funcţia identitate are efect neutru la compunere. Pentru orice funcţie S–sortată $f : M \longrightarrow N$ au loc egalităţile $1_M; f = f = f; 1_N$.

Cele două proprietăţi de mai sus ne permit să vorbim de categoria mulţimilor S–sortate.

1.2 Signaturi multisortate

În programare, mai mult decât în orice altă activitate, datele utilizate sunt de mai multe feluri, sau **sorturi** aşa cum vom spune în continuare. Mai mult, de cele mai multe ori, în diferitele construcţii sintactice, într-un anumit loc al acestora nu poate fi plasată decât o dată de un anumit sort. Aceasta ar fi explicaţia faptului că algebrele multisortate constituie una dintre cele mai utile unelte pentru informatica teoretică.

Algebrele la rândul lor nu sunt toate la fel. Felul algebrelor este dat de signatura lor. O signatură are două componenete: una pentru date şi una pentru operaţii.

Componenta pentru date este pur şi simplu o mulţime S ale cărei elemente $s \in S$ se numesc sorturi.

Fiecare operaţie este caracterizată de modul acesteia de acţiune. Operaţia acţionează pe un anumit număr fix de date de sorturi precizate şi are rezultatul de un sort dat. Ca exemplu pentru operaţia cu **numele** o notăm cu

$$o : s_1 s_2 \dots s_n \longrightarrow s$$

faptul ca ea are n argumente de sorturi s_1, s_2, ..., s_n şi rezultatul este de sort s. Numele unei operaţii mai este numit **simbol de operaţie** sau operator.

Terminologia folosită este următoarea:

$s_1 s_2 \dots s_n$ se numeşte **aritate**,
s este **sortul rezultat** sau sortul rezultatului,
perechea $(s_1 s_2 \dots s_n, s)$ se numeşte **rang**.

Reamintim că substantivul aritate provine din sufixul "ară" folosit în expresii ca zeroară, unară, binară, ternară, etc.

Toate aceste informaţii privind felul algebrei sunt adunate în conceptul de signatură. Cu S^* notăm mulţimea şirurilor finite formate cu elemente din S.

Definiţia 1.3 O signatură algebrică

$$(S, \{\Sigma_{s_1 s_2 \dots s_n, s}\}_{s_1 s_2 \dots s_n \in S^*, s \in S})$$

este formată dintr-o mulţime S ale cărei elemente se numesc sorturi şi o familie de mulţimi

$$\{\Sigma_{s_1 s_2 \dots s_n, s}\}_{s_1 s_2 \dots s_n \in S^*, s \in S}.$$

Pentru fiecare $s_1 s_2 \dots s_n \in S^*$ şi $s \in S$ mulţimea $\Sigma_{s_1 s_2 \dots s_n, s}$ conţine numele operaţiilor cu n argumente de sorturi $s_1, s_2, \dots, s_n$ şi rezultat de sort s.

Menţionăm că mulţimile $\Sigma_{s_1 s_2 \dots s_n, s}$ pot avea elemente comune, ceea ce permite modelarea supraîncărcării operaţiilor, adică permisiunea ca mai multe operaţii să aibă acelaş nume, sau altfel spus să fie denumite prin acelaşi simbol.

Când nu există pericol de confuzie vom scrie signatură în loc de signatură algebrică şi vom scrie (S, Σ) sau Σ în loc de

$$(S, \{\Sigma_{s_1 s_2 \dots s_n, s}\}_{s_1 s_2 \dots s_n \in S^*, s \in S}).$$

Cea mai cunoscută signatură multisortată provenită din algebra clasică este cea corespunzătoare conceptelor de spaţiu vectorial sau modul. Datele sunt de două sorturi: scalari şi vectori. Simbolurile de operaţii sunt de trei feluri

1. simboluri de operaţii pentru scalari corespunzătoare structurii de corp sau inel
2. simboluri de operaţii pentru vectori corespunzătoare structurii de grup abelian
3. produsul cu scalari : scalar vector $\longrightarrow$ vector.

O consecinţă deosebită a stilului multisortat este faptul că relaţiile pot fi definite ca operaţii cu sortul rezultat boolean. De exemplu

$$\leq \ : \mathit{natural}\ \mathit{natural} \longrightarrow \mathit{boolean}$$

Observăm că $3 \leq 5 =$ adevăr şi $5 \leq 3 =$ fals.

În general prin **relaţie** se înţelege o operaţie al cărei sort rezultat este boolean. Practic, pentru a transforma o relaţie în operaţie se înlocuieşte relaţia cu aşa numita funcţia ei caracteristică. Reamintim că funcţia caracteristică

$$\chi_A : M \longrightarrow \{\text{adevăr, fals}\}$$

a submulţimii A a mulţimii M este definită prin

$$\chi_A(m) = \begin{cases} \text{adevăr} & \text{dacă } m \in A \\ \text{fals} & \text{dacă } m \notin A. \end{cases}$$

Remarcabil este că unele aspecte privind studiul clasic al modelelor unde apar atât operaţii cât şi relaţii poate fi redus la studiul algebrelor multisortate unde apar numai operaţii.

1.3 Algebre multisortate

Algebrele sunt formate în mare din date şi operaţii. Datele sunt de mai multe sorturi, adică pentru fiecare sort s algebra conţine o mulţime a datelor de sort s. Familia acestor mulţimi, numită şi **suportul algebrei**, constitue o mulţime sortată.

Definiţia 1.4 O Σ-algebră $\mathcal{A} = (\{A_s\}_{s\in S}, \{A_\sigma\}_{\sigma\in\Sigma})$ este formată dintr-o mulţime S-sortată, numită suportul algebrei, $A = \{A_s\}_{s\in S}$ şi o familie de operaţii $\{A_\sigma\}_{\sigma\in\Sigma}$. Pentru claritate, dacă $\sigma \in \Sigma_{s_1 s_2 \ldots s_n, s}$, adică $\sigma : s_1 s_2 \ldots s_n \longrightarrow s$, atunci A_σ este o funcţie

$$A_\sigma : A_{s_1} \times A_{s_2} \times \cdots \times A_{s_n} \longrightarrow A_s. \ \odot$$

Dacă nu există pericol de confuzie în loc de $(\{A_s\}_{s\in S}, \{A_\sigma\}_{\sigma\in\Sigma})$ vom scrie mai simplu (A_s, A_σ). Mai menţionăm că pentru o algebră $\mathcal{A}$ şi suportul acesteia A folosim aceeaşi literă cu grafii diferite.

Dacă nu este pericol de confuzie în loc de Σ-algebră vom scrie mai scurt algebră.

În continuare pentru $s_1, s_2, \ldots, s_n \in S$ vom mai folosi şi notaţia

$$A_{s_1 s_2 \ldots s_n} = A_{s_1} \times A_{s_2} \times \cdots \times A_{s_n}.$$

Din definiţia de mai sus rezultă că dacă $\sigma \in \Sigma_{\lambda,s}$ unde λ este şirul vid din S^*, atunci A_σ este o funcţie definită pe o mulţime cu un element şi cu valori în A_s. Pentru a simplifica scrierea această funcţie este înlocuită cu unica ei valoare, element din A_s adică $A_\sigma \in A_s$. Deci operaţiile fără argumente, numite şi **constante**, sunt elemente ale algebrei de sort corespunzător sortului rezultat al numelui operaţiei.

Vom continua prin a defini pentru algebrele multisortate cele mai uzuale concepte specifice algebrei: morfisme, algebre libere, congruenţe, etc.

Asemanător algebrei care abordează pe rând diferite structuri algebrce, trebuie să facem acelaş lucru. Adică trebuie să studiem, dar în acelaşi timp, structuri algebrice de naturi diferite. Prin urmare, în continuare, fixăm signatura (S, Σ) a algebrelor de care ne ocupăm. De altfel unele concepte, ca de exemplu cel de morfism, nu pot fi definite decât pentru algebre având aceeaşi signatură.

Fixarea signaturii arată că ne ocupăm de o anumită structură algebrică. Faptul că signatura este arbitrară arată ca studiul diferitelor structuri algebrice se face simultan.

1.4 Morfisme de algebre multisortate

Un morfism între două algebre multisortate, asemănător oricărui morfism de structuri algebrice, este o funcţie multisortată între suporturile celor două algebre care verifică o condiţie suplimentară. Pentru a scrie această condiţie pentru cazul algebrelor multisortate să plecăm de la conceptul uzual de morfism pentru o structură algebrică bazată pe o operaţie binară. Funcţia $h : A \longrightarrow B$ este morfism $h : (A, *) \longrightarrow (B, \&)$ dacă

$$(\forall a \in A)(\forall b \in A) h(a * b) = h(a) \& h(b).$$

Să analizăm egalitatea de mai sus. Se evaluează cei doi membri pentru un număr de elemente arbitrare din prima algebră egal cu numărul de argumente al operaţiei şi apoi se egalează rezultatele

- membrul stâng:

1) se aplică operaţia $*$ din prima algebră elementelor a şi b din prima algebra
2) se aplica morfismul h rezultatului obţinut $a * b$

- membrul drept:

1) se aplică morfismul h elementelor a şi b din prima algebra obţinându-se nişte elemente $h(a)$ şi $h(b)$ din a doua algebră
2) se aplică operaţia & din a doua algebră acestor elemente

- se cere ca rezultatul evaluării celor doi membri să fie egali.

Să facem acelaşi lucru pentru două algebre multisortate $\mathcal{A} = (A_s, A_\sigma)$, $\mathcal{B} = (B_s, B_\sigma)$ şi o funcţie S-sortată $h : A \longrightarrow B$. Condiţia de mai sus trebuie pusă pentru fiecare operaţie cu numele

$$\sigma : s_1 s_2 \ldots s_n \longrightarrow s$$

şi oricare ar fi elementele $a_1 \in A_{s_1}$, $a_2 \in A_{s_2} \ldots a_n \in A_{s_n}$ din prima algrbră

- membrul stâng:

1) se aplică operaţia din prima algebră elementelor din prima algebra: $A_\sigma(a_1, a_2, \ldots, a_n)$
2) se aplica morfismul h rezultatului obţinut $h_s(A_\sigma(a_1, a_2, \ldots, a_n))$

- membrul drept:

1) se aplică morfismul h elementelor din prima algebra obţinându-se nişte elemente din a doua algebră:

$$h_{s_1}(a_1), h_{s_2}(a_2), \ldots, h_{s_n}(a_n)$$

2) se aplică operaţia din a doua algebră acestor elemente: $B_\sigma(h_{s_1}(a_1), h_{s_2}(a_2), \ldots, h_{s_n}(a_n))$

- se cere ca rezultatul evaluării celor doi membri să fie egali.

$$h_s(A_\sigma(a_1, a_2, \ldots, a_n)) = B_\sigma(h_{s_1}(a_1), h_{s_2}(a_2), \ldots, h_{s_n}(a_n)).$$

Definiţia 1.5 Funcţia S-sortată $h : A \longrightarrow B$ este un morfism de Σ-algebre multisortate

$$h : \mathcal{A} = (A_s, A_\sigma) \longrightarrow \mathcal{B} = (B_s, B_\sigma)$$

dacă pentru orice $s_1 s_2 \ldots s_n \in S^*$, pentru orice $s \in S$, pentru orice $\sigma \in \Sigma_{s_1 s_2 \ldots s_n, s}$, pentru orice $a_1 \in A_{s_1}$, $a_2 \in A_{s_2}$, $\ldots$, $a_n \in A_{s_n}$

$$h_s(A_\sigma(a_1, a_2, \ldots, a_n)) = B_\sigma(h_{s_1}(a_1), h_{s_2}(a_2), \ldots, h_{s_n}(a_n)). \ \odot$$

Dacă nu este pericol de confuzie în loc de morfism de Σ-algebre vom scrie morfism de algebre sau chiar morfism.

Este util să remarcăm că există câte o condiţie pentru fiecare nume de operaţie. În cazul operaţiilor fără argumente, aşa zisele constante, condiţia de morfism este pentru orice $\sigma \in \Sigma_{\lambda,s}$ egalitatea $h_s(A_\sigma) = B_\sigma$. Cu alte cuvinte morfismele trebuie să păstreze constantele. Pe cazuri particulare observăm că orice morfism de monoizi duce elementul neutru în elementul neutru şi că orice morfism de semiinele duce elementul neutru la adunare, respectiv la înmulţire, tot în elementul neutru la adunare respectiv la înmulţire.

În continuare pentru $s_1, s_2, \ldots, s_n \in S$ vom mai folosi şi notaţia

$$h_{s_1 s_2 \ldots s_n} = h_{s_1} \times h_{s_2} \times \cdots \times h_{s_n} : A_{s_1 s_2 \ldots s_n} \longrightarrow B_{s_1 s_2 \ldots s_n}.$$

Cu aceleaşi notaţii mai precizăm că dacă $a_i \in A_{s_i}$ pentru orice $1 \leq i \leq n$

$$h_{s_1 s_2 \ldots s_n}(a_1, a_2, \ldots, a_n) = (h_{s_1}(a_1), h_{s_2}(a_2), \ldots, h_{s_n}(a_n)).$$

Cu această notaţie, condiţia de morfism pentru operaţia $\sigma : s_1 s_2 \ldots s_n \longrightarrow s$ este echivalentă cu

$$A_\sigma ; h_s = h_{s_1 s_2 \ldots s_n} ; B_\sigma.$$

Este deasemenea util să menţionăm diferenţa de notaţie dintre o funcţie S-sortată

$$f : A \longrightarrow B$$

între suporturile a două Σ-algebre $\mathcal{A}$ şi $\mathcal{B}$ şi un morfism între cele două algebre

$$h : \mathcal{A} \longrightarrow \mathcal{B}.$$

Observăm că funcţia identitate 1_A este morfism de Σ-algebre de la $\mathcal{A}$ la $\mathcal{A}$ fapt notat prin $1_\mathcal{A} : \mathcal{A} \longrightarrow \mathcal{A}$.

Propoziţie 1.6 *Compunerea ca funcţii S-sortate a două morfisme de Σ-algebre este un morfism de Σ-algebre.*

Demonstraţie: Fie $h : \mathcal{A} \longrightarrow \mathcal{B}$ şi $g : \mathcal{B} \longrightarrow \mathcal{C}$ două morfisme de Σ-algebre. Probăm că $h; g : \mathcal{A} \longrightarrow \mathcal{C}$ este morfism de Σ-algebre.

Fie $s_1 s_2 \ldots s_n \in S^*$, $s \in S$, $\sigma \in \Sigma_{s_1 s_2 \ldots s_n, s}$ şi $a_1 \in A_{s_1}$, $a_2 \in A_{s_2}$, $\ldots$, $a_n \in A_{s_n}$. Observăm că

$$(h;g)_s(A_\sigma(a_1, a_2, \ldots, a_n)) = g_s(h_s(A_\sigma(a_1, a_2, \ldots, a_n))) = g_s(B_\sigma(h_{s_1}(a_1), h_{s_2}(a_2), \ldots, h_{s_n}(a_n))) =$$

$$= C_\sigma(g_{s_1}(h_{s_1}(a_1)), g_{s_2}(h_{s_2}(a_2)), \ldots, g_{s_n}(h_{s_n}(a_n))) = C_\sigma((h;g)_{s_1}(a_1), (h;g)_{s_2}(a_2), \ldots, (h;g)_{s_n}(a_n)). \ \odot$$

Compunerea morfismelor de Σ-algebre este asociativă.

Morfismul identitate are efect neutru la compunere.

Datorită celor două proprietăţi de mai sus putem vorbi de categoria Σ–algebrelor.

2 ALGEBRE LIBERE

După această mică introducere privind algebrele multisortate, trecem la conceptul de algebră liberă datorită importantelor aplicaţii ale acestuia în informatică. Cu el se modelează noţiunile de expresie şi de evaluarea a unei expresii.

Menţionăm că demonstraţia privind existenţa algebrelor libere este prezentată mai târziu după o motivaţie intuitivă şi un exemplu de utilizare a conceptului în informatică.

2.1 Expresii

2.1.1 Ce este o expresie?

Conceptul de *expresie* aşa cum este el folosit în învăţământul preuniversitar nu are o definiţie şi un înţeles precis. Vom da un exemplu care să ilustreze acest fapt. La întrebarea "este $x * y * z$ o expresie?" răspunsul depinde de contextul în care a fost pusă întrebarea. Dacă operaţia $*$ a fost declarată asociativă, atunci $x * y * z$ este o expresie. În caz contrar ea nu este o expresie deoarece include o ambiguitate putând fi interpretată ca $x * (y * z)$ sau $(x * y) * z$ ambele fiind expresii. Pentru început noţiunea de expresie va fi definită în ipoteza că *operaţiile cu care lucrăm nu au nici o proprietate suplimentară.*

Mai menţionăm că cele două expresii de mai sus mai pot fi scrise în scrierea poloneză $*x * yz$ şi $* * xyz$ sau în scrierea polonezăă inversă $xyz**$ şi $xy*z*$. Ne interesează *o definiţie a conceptului de expresie care să fie independentă de forma de scriere a acesteia.*

Definiţia 2.1 Σ-algebra $\mathcal{A} = (A_s, A_\sigma)$ se numeşte **liber generată** de $V \subseteq A$ dacă pentru orice Σ-algebră $\mathcal{D}$ şi pentru orice funcţie sortată $f : V \to D$, există un unic morfism de Σ-algebre $f^\# : \mathcal{A} \to \mathcal{D}$ care extinde f.

Σ-algebra $\mathcal{A}$ se numeşte liberă dacă există $V \subseteq A$ astfel încât $\mathcal{A}$ este liber generată de V. $\odot$

Definiţia 2.2 Se numeşte **expresie** un element dintr-o algebră liberă.$\odot$

Când privim algebra liberă ca o algebră de expresii, mulţimea V de mai sus reprezintă mulţimea variabilelor. În unele cazuri, elementele lui V sunt numite generatori ai algebrei.

Bineînţeles că acest concept este încă dependent de signatura cu care lucrăm, fapt pentru care atunci când dorim să precizăm signatura vorbim de Σ–expresii în loc de expresii.

În plus noţiunea naivă de expresie ne dă intuiţia necesară pentru înţelegerea conceptului de algebră liberă: algebrele libere nu sunt altceva decât algebre de expresii.

Independenţa de modul de scriere al expresiilor corespunde unicităţii abstracţie de un izomorfism al algebrei libere pentru care este fixată mulţimea V a generatorilor.

2.1.2 Evaluarea expresiilor

Un alt concept deosebit de util atât în matematică cât şi în informatică este cel de *evaluare a unei expresii.*

Începem cu un exemplu. Să presupunem că se doreşte evaluare unei expresii $a + b$ care este suma a două expresii a şu b. Se procedează astfel:

- se evaluează una dintre expresii, de exemplu a, obţinându-se rezultatul evaluării notat $ev(a)$,
- se evaluează cealaltă expresie b, obţinându-se rezultatul evaluării notat $ev(b)$,
- se adună cele două rezultate $ev(a) + ev(b)$, obţinându-se rezultatul evaluării expresiei $a + b$ notat $ev(a + b)$.

In concluzie

$$ev(a + b) = ev(a) + ev(b)$$

ceea ce arată că modelarea matematică adecvată pentru evaluare este conceptul de morfism.

Deşi este clar că pentru a evalua o expresie este necesar să dăm valori variabilelor care apar în ea, mai puţin evident este faptul că trebuie precizat şi unde dăm valori acestor variabile. Pentru a ilustra acest fapt menţionăm că expresia $x?(y \top z)$ nu poate fi evaluată numai dând valori variabilelor x, y şi z într-o mulţime dacă mulţimea nu este înzestrată cu două operaţii binare corespunzătoare simbolurilor de operaţii binare ? şi $\top$. În concluzie pentru a evalua o expresie este necesar să dăm

1. o algebră în care se fac calculele şi care are aceeaşi signatură cu cea a expresiei
2. valori variabilelor din expresie în algebra în care se fac calculele.

Menţionăm că **a da valori variabilelor din mulţimea X în algebra $\mathcal{D}$** este echivalent cu a da o funcţie

$$v : X \longrightarrow D.$$

Pentru orice variabilă x din X valoarea dată lui x este $v(x)$.

Vom nota cu $T_\Sigma(X)$ algebra liber generată de mulţimea X de variabile. Incluziunea $X \subseteq T_\Sigma(X)$ este echivalentă cu faptul intuitiv că orice variabilă este o expresie. Pentru orice algebră $\mathcal{D}$ şi pentru orice funcţie $v : X \longrightarrow D$ există, conform definiţiei algebrelor libere, un unic morfism $v^\# : T_\Sigma(X) \longrightarrow \mathcal{D}$ a cărui restricţie la X coincide cu v.

Fixând algebra $\mathcal{D}$ vom constata că există o bijecţie naturală între $Alg_\Sigma(T_\Sigma(X), \mathcal{D})$ mulţimea morfismelor de Σ-algebre de la $T_\Sigma(X)$ la $\mathcal{D}$ şi $Set_S(X, D)$ mulţimea funcţiilor S-sortate de la X la D. Fie

$$r : Alg_\Sigma(T_\Sigma(X), \mathcal{D}) \longrightarrow Set_S(X, D)$$

funcţia restricţie, adică $r(h) : X \longrightarrow D$ este restricţia $h/_X$ a morfismului $h : T_\Sigma(X) \longrightarrow \mathcal{D}$ la X. Proprietatea de mai sus a algebrei libere spune că

$$(\forall v \in Set_S(X, \mathcal{D}))(\exists! v^\# \in Alg_\Sigma(T_\Sigma(X), \mathcal{D}))r(v^\#) = v$$

adică r este bijecţie. Existenţa acestei bijecţii ne permite să identificăm elementele celor două mulţimi fără a mai face distincţie între un morfism $v^\#$ de la $T_\Sigma(X)$ la $\mathcal{D}$ şi v, restricţia lui la X.

Dacă mai sus scriam că a da valori variabilelor din mulţimea X în algebra $\mathcal{D}$ este echivalent cu a da o funcţie $v : X \longrightarrow \mathcal{D}$ acum putem spune că:

A da valori variabilelor din mulţimea X în algebra $\mathcal{D}$ este echivalent cu a da un morfism $v : T_\Sigma(X) \longrightarrow \mathcal{D}$.

O altă consecinţa a celor de mai sus este:
Pentru a defini un morfism de la algebra liber generată de X la algebra $\mathcal{D}$ este suficient să dăm o funcţie de la X la $\mathcal{D}$.

Definiţia 2.3 Dacă $e \in T_\Sigma(X)$ este o expresie cu variabile din X şi $h : T_\Sigma(X) \longrightarrow \mathcal{D}$ morfismul prin care se dau valori în $\mathcal{D}$ variabilelor, atunci $h(e)$ este rezultatul evaluării expresiei e pentru valorile variabilelor date de funcţia $h/_X : X \longrightarrow \mathcal{D}$. $\odot$

Pentru a ne convinge că această definiţie modelează corect realitatea vom relua exemplul de mai sus privind expresia $x?(y \top z)$. Să evaluăm această expresie în mulţimea numerelor naturale unde ? este înmulţirea şi $\top$ este adunarea.

Pentru valorile $x = 2$, $y = 3$ şi $z = 1$ intuitiv obţinem $2 * (3 + 1) = 8$ iar cu definiţia de mai sus unde

$$h : (T_\Sigma(\{x, y, z\}), ?, \top) \longrightarrow (N, *, +)$$

este morfismul definit prin $h(x) = 2$, $h(y) = 3$ şi $h(z) = 1$ obţinem

$$h(x?(y \top z)) = h(x) * h(y \top z) = h(x) * (h(y) + h(z)) = 2 * (3 + 1) = 8.$$

2.1.3 Semantica instrucţiunii de atribuire

Fie X mulţimea variabilelor utilizate întrun program. O instrucţiune de atribuire este de forma $x := e$ unde x este o variabilă şi e este o expresie, adică $e \in T_\Sigma(X)$.

Fie $\mathcal{D}$ Σ-algebra datelor cu care se fac calculele. Ne interesează partiţia memoriei în care sunt memorate datele utilizate în timpul execuţiei programului, date depozitate în celule ale memoriei care corespund variabilelor din X. Prin urmare starea memoriei este caracterizată în fiecare moment de o funcţie $s : X \longrightarrow D$. Dacă x este o variabilă s(x) este valoarea din celula de memorie corespunzătoare lui x. Fie S mulţimea stărilor memoriei, adică mulţimea funcţiilor de la mulţimea variabilelor X la mulţimea datelor D.

O funcţie parţială de la A la B este o funcţie definită numai pe o parte a lui A cu valori în B.

Semantica unei instrucţiuni, sau mai general a unui program, este o funcţie parţială F de la mulţimea S a stărilor la ea insăşi. Funcţia F este definită pentru starea s a memoriei dacă şi numai dacă execuţia instrucţiunii începută în starea s a memoriei se termină. Mai mult $F(s)$ este starea memoriei în momentul terminării execuţiei.

Vom defini $Sem(x := e) : S \longrightarrow S$, semantica atribuirii $x := e$. Fie $s : X \longrightarrow D$ starea memoriei la începutul execuţiei atribuirii şi $s^\# : T_\Sigma(X) \longrightarrow \mathcal{D}$ unica extindere la un morfism a lui s. Observăm că $s^\#(e)$ este rezultatul evaluării expresiei e în starea s a memorie. Prin urmare, prin definiţie

$$Sem(x := e)(s)(y) = \begin{cases} s^\#(e) & \text{dacă } y = x \\ s(y) & \text{dacă } y \neq x. \end{cases}$$

Pentru o mai bună înţelegere menţionăm că $s(y)$ este valoarea variabilei y în momentul începerii execuţiei instrucţiunii de atribuire şi că $Sem(x := e)(s)(y)$ este valoarea variabilei y în momentul terminării execuţiei instrucţiunii de atribuire $x := e$.

2.1.4 Algebre iniţiale

Definiţia 2.4 O Σ-**algebră** $\mathcal{I}$ se numeşte **iniţială** dacă pentru orice Σ-algebră $\mathcal{A}$ există un unic morfism

$$\alpha_{\mathcal{A}} : \mathcal{I} \longrightarrow \mathcal{A}.$$

Observăm că o algebră este iniţială dacă şi numai dacă este liber generată de mulţimea vidă.

2.2 Algebra arborilor de derivare

O gramatică independentă de context posedă două mulţimi disjuncte, una a neterminalelor N şi una a terminalelor T precum şi multimea $P \subseteq N \times (N \cup T)^*$ a producţiilor. Chiar dacă conceptul de gramatică independentă de context verifică şi alte condiţii, noi ne restrângem doar la cele de mai sus deoarece acestea sunt utile în cele ce urmează.

Fiecărei gramaticii independente de context G i se poate ataşa o signatură:

- neterminalele devin sorturi,
- producţiile devin nume de operaţii,
- dacă $(n, t_0 n_1 t_1 \dots n_k t_k) \in P$ unde literele n sunt neterminale şi literele t sunt cuvinte cu litere terminale, atunci

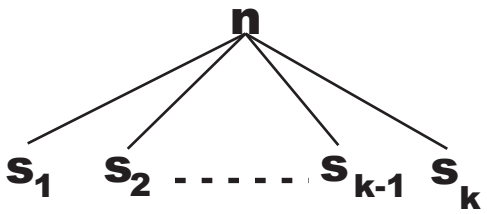
$$(n, t_0 n_1 t_1 \dots n_k t_k) : n_1 n_2 \dots n_k \longrightarrow n$$

adică operaţia corespunzătoare lui $(n, t_0 n_1 t_1 \dots n_k t_k)$ are ca rezultat un element de sortul indicat de neterminalul din membrul stâng al producţiei, un număr de argumente egal cu numărul de neterminale din membrul drept al producţiei şi sorturile argumentelor sunt chiar neterminalele din membrul drept al producţiei.

O algebră a cărei signatură este cea ataşată gramaticii independente de context G se numeşte G-algebră.

Un arbore de derivare într-o gramatică independentă de context are următoarele proprietăţi:

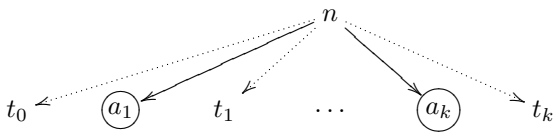
1. este un arbore finit şi local ordonat, adică succesorii fiecărui nod sunt într-o ordine totală;
2. are noduri etichetate cu terminale sau neterminale, rădăcina fiind etichetată cu un neterminal;
3. orice nod etichetat cu un terminal nu are nici un succesor;
4. pentru orice nod, dacă este etichetat cu un neterminal n, atunci perechea formată din n şi cuvântul format de etichetele succesorilor formează o producţie $(n, s_1 s_2 \dots s_{k-1} s_k) \in P$.



Arborii de derivare ai unei gramatici independente de context G pot fi organizaţi ca o G-algebră $\mathcal{A}$ după cum urmează:

- pentru orice sort n, adică pentru orice neterminal n, mulţimea A_n este formată din totalitatea arborilor de derivare care au rădăcina etichetată cu n.
- pentru producţia $p = (n, t_0 n_1 t_1 \dots n_k t_k)$ şi arborii $a_i \in A_{n_i}$ arborele $A_p(a_1, a_2, \dots, a_k)$ este format astfel: rădăcina este etichetată cu n, etichetele succesorilor rădăcinii în număr egal cu numărul literelor din $t_0 n_1 t_1 \dots n_k t_k$ sunt chiar literele acestui cuvânt şi subarborele fiecărui nod etichetat cu n_i este chiar a_i.

$A_p(a_1, a_2, \dots, a_k) =$



2.3 Iniţialitatea algebrei arborilor de derivare

Lemă 2.5 *Pentru orice arbore de derivare $a \in A_n$, unde n este un neterminal există şi sunt unice o producţie $p = (n, t_0 n_1 t_1 \dots n_k t_k)$ şi arborii de derivare $a_i \in A_{n_i}$ cu proprietatea $a = A_p(a_1, a_2, \dots, a_k)$.*

Demonstraţie: Fie n un neterminal şi $a \in A_n$. Analizând rădăcina şi primul nivel al arborelui a deducem existenţa unei producţii $p = (n, t_0 n_1 t_1 \dots n_k t_k)$ cu proprietatea

$$a = A_p(a_1, a_2, \dots, a_k)$$

unde a_i sunt subarborii lui a care au rădăcinile în succesorii rădăcinii lui a care sunt etichetaţi cu neterminale.

Pentru a proba unicitatea presupunem că

$$a = A_q(b_1, b_2, \dots, b_{k'})$$

unde $q = (n', t'_0 s_1 t'_1 \dots s_{k'} t'_{k'})$ este o producţie în care literele n şi s sunt neterminale şi literele t sunt cuvinte formate din terminale. In plus $b_i \in A_{s_i}$ pentru orice $1 \leq i \leq k'$.

Toate concluziile vor rezulta din egalitatea

$$A_p(a_1, a_2, \dots, a_k) = A_q(b_1, b_2, \dots, b_{k'}).$$

Deoarece eticheta rădăcinii este unică deducem $n = n'$.

Egalând cuvintele formate cu etichetele succesorilor rădăcinilor din cei doi arbori egali deducem

$$t_0 n_1 t_1 \dots n_k t_k = t'_0 s_1 t'_1 \dots s_{k'} t'_{k'}.$$

Reamintim că prin definiţia gramaticilor un element nu poate fi în acelaşi timp şi terminal şi neterminal. Deoarece numărul neterminalelor din cele două cuvintre trebuie să fie egal deducem egalitatea $k = k'$. Deoarece primele neterminale din cele două cuvinte trebuie să fie pe aceeaşi poziţie deducem că $t_0 = t'_0$ şi $n_1 = s_1$. Rezultă că

$$t_1 n_2 t_2 \dots n_k t_k = t'_1 s_2 t'_2 \dots s_k t'_k.$$

Continuăm raţionamentul ca mai sus şi deducem $t_i = t'_i$ pentru $0 \leq i \leq k$ şi $n_i = s_i$ pentru $1 \leq i \leq k$. De aici deducem că $p = q$.

În final egalând subarborii cu rădăcinile aflate pe aceeaşi poziţie a primului nivel din arborii egali $A_p(a_1, a_2, \dots, a_k)$ şi $A_q(b_1, b_2, \dots, b_{k'})$ rezultă că $a_i = b_i$ pentru orice $1 \leq i \leq k$.

Teorema 2.6 *Algebra arborilor de derivare ai unei gramatici independente de context este algebră iniţială.*

Demonstraţie: Fie $\mathcal{B} = (\{B_n\}_{n \in N}, \{B_p\}_{p \in P})$ o G–algebră. Vom arăta că există şi este unic un morfism h de la algebra $\mathcal{A}$ a arborilor de derivare la $\mathcal{B}$.

Definim funcţia N–sortată $h : A \longrightarrow B$. Prin inducţie după adâncimea arborelui $a \in A_n$ definim $h_n(a)$ în B_n.

Fie m un număr natural. Pentru orice neterminal $n \in N$ şi orice arbore de derivare $a \in A_n$ de adâncime strict mai mică decât m presupunem definit $h_n(a) \in B_n$.

Fie arborele de derivare $a \in A_n$ de adâncime m. Conform lemei există şi sunt unice o producţie $p = (n, t_0 n_1 t_1 \dots n_k t_k)$ şi arborii de derivare $a_i \in A_{n_i}$ cu proprietatea $a = A_p(a_1, a_2, \dots, a_k)$. Observăm că pentru orice $1 \leq i \leq k$ arborele de derivare $a_i \in A_{n_i}$ are adâncimea strict mai mică decât m, prin urmare $h_{n_i}(a_i) \in B_{n_i}$ este definit. Definim

$$h_n(a) = B_p(h_{n_1}(a_1), h_{n_2}(a_2), \dots, h_{n_k}(a_k)).$$

Se observă că prin această definiţie

$$h_n(A_p(a_1, a_2, \dots, a_k)) = B_p(h_{n_1}(a_1), h_{n_2}(a_2), \dots, h_{n_k}(a_k))$$

adică $h : \mathcal{A} \longrightarrow \mathcal{B}$ este morfism de G–algebre.

Probăm unicitatea. Fie $f : \mathcal{A} \longrightarrow \mathcal{B}$ un morfism de G–algebre. Demonstrăm egalitatea $f = g$ printr-o inducţie la fel ca mai sus.

Fie m un număr natural. Pentru orice neterminal $n \in N$ şi orice arbore de derivare $a \in A_n$ de adâncime strict mai mică decât m presupunem că $f_n(a) = h_n(a)$.

Fie arborele de derivare $a \in A_n$ de adâncime m. Conform lemei există şi sunt unice o producţie $p = (n, t_0 n_1 t_1 \dots n_k t_k)$ şi arborii de derivare $a_i \in A_{n_i}$ cu proprietatea $a = A_p(a_1, a_2, \dots, a_k)$. Deoarece arborii de derivare $a_i \in A_{n_i}$ au adâncimea strict mai mică decât m, conform ipotezei de inducţie $f_{n_i}(a_i) = h_{n_i}(a_i)$ pentru orice $1 \leq i \leq k$. Rezultă că

$f_n(a) = f_n(A_p(a_1, a_2, \dots, a_k)) = B_p(f_{n_1}(a_1), f_{n_2}(a_2), \dots, f_{n_k}(a_k)) =$
$B_p(h_{n_1}(a_1), h_{n_2}(a_2), \dots, h_{n_k}(a_k)) = h_n(A_p(a_1, a_2, \dots, a_k)) = h_n(a)$.

2.4 Existenţa algebrelor libere

Fie (S, Σ) o signatură şi $X = \{X_s\}_{s \in S}$ o mulţime S–sortată de variabile. Componentele mulţimii X sunt prin definiţie două câte două disjuncte.

Teorema 2.7 *Pentru orice signatură (S, Σ) şi orice mulţime S–sortată de variabile X există Σ–algebra liber generată de X.*

Demonstraţie: Fără a micşora generalitatea vom presupune că Σ şi X sunt disjuncte. Definim gramatica independentă de context

$$G = (S, \Sigma \cup X, P)$$

unde $P = \{s \longrightarrow \sigma s_1 s_2 \dots s_k \ : \ \sigma \in \Sigma_{s_1 s_2 \dots s_k, s}\} \cup \{s \longrightarrow x \ : \ s \in S \text{ şi } x \in X_s\}$.

Notăm cu

$$\mathcal{A} = (\{A_s\}_{s \in S}, \{A_{s \longrightarrow \sigma s_1 s_2 \dots s_k}\}_{\sigma \in \Sigma_{s_1 s_2 \dots s_k, s}}, \{A_{s \longrightarrow x}\}_{s \in S, x \in X_s})$$

G–algebra iniţială.

Observăm că

$$\mathcal{A}^* = (\{A_s\}_{s \in S}, \{A_{s \longrightarrow \sigma s_1 s_2 \dots s_k}\}_{\sigma \in \Sigma_{s_1 s_2 \dots s_k, s}})$$

este o Σ–algebră. Vom arăta că ea este liber generată de X.

Mulţimea variabilelor din X este injectată în $\{A_s\}_{s \in S}$ prin funcţia
$i : X \longrightarrow \{A_s\}_{s \in S}$ definită pentru orice $s \in S$ şi $x \in X_s$ prin

$$i_s(x) = A_{s \longrightarrow x}.$$

Fie $\mathcal{D} = (D_s, D_\sigma)$ o Σ–algebră şi $f : X \longrightarrow \{D_s\}_{s \in S}$ o funcţie S–sortată. Deoarece

$$(D_s, D_\sigma, \{f(x)\}_{s \in S, x \in X_s})$$

este o G–algebră există un unic morfism de G–algebre

$$h : \mathcal{A} \longrightarrow (D_s, D_\sigma, \{f(x)\}_{s \in S, x \in X_s}).$$

Observăm că h este morfism de G–algebre dacă şi numai dacă h este morfism de Σ–algebre şi $h_s(A_{s \longrightarrow x}) = f(x)$ pentru orice $s \in S$ şi $x \in X_s$, adică dacă şi numai dacă h este morfism de Σ–algebre şi $i; h = f$.

In concluzie există un unic morfism de Σ–algebre $h : \mathcal{A}^* \longrightarrow \mathcal{D}$ cu proprietatea $i; h = f$.

3 Tipuri abstracte de date - introducere

3.1 Izomorfisme de algebre multisortate

Definiţia 3.1 Morfismul de Σ-algebre $h : \mathcal{A} \longrightarrow \mathcal{B}$ se numeşte **izomorfism** dacă există morfismul $g : \mathcal{B} \longrightarrow \mathcal{A}$ cu proprietăţile $h; g = 1_{\mathcal{A}}$ şi $g; h = 1_{\mathcal{B}}$.

Dacă există, morfismul g din definiţia de mai sus este unic. Întradevăr dacă $f : \mathcal{B} \longrightarrow \mathcal{A}$ este un alt morfism cu proprietăţile $h; f = 1_{\mathcal{A}}$ şi $f; h = 1_{\mathcal{B}}$. Observăm că

$$g = g; 1_{\mathcal{A}} = g; (h; f) = (g; h); f = 1_{\mathcal{B}}; f = f.$$

Datorită unicităţii sale, conform uzanţelor, morfismul g, denumit şi inversul lui h, este notat în continuare cu h^{-1}. Prin urmare pentru orice izomorfism $h : \mathcal{A} \longrightarrow \mathcal{B}$ menţionăm egalităţile $h; h^{-1} = 1_{\mathcal{A}}$ şi $h^{-1}; h = 1_{\mathcal{B}}$.

Observăm că morfismele identitate sunt izomorfisme. In plus $(1_{\mathcal{A}})^{-1} = 1_{\mathcal{A}}$.

Propoziţie 3.2 *Un morfism este izomorfism dacă şi numai dacă are toate componentele bijective.*

Demonstraţie: Fie $h : \mathcal{A} \longrightarrow \mathcal{B}$ un morfism de Σ-algebre.

Presupunem că h este izomorfism, adică există morfismul $h^{-1} : \mathcal{B} \longrightarrow \mathcal{A}$ cu proprietăţile $h; h^{-1} = 1_{\mathcal{A}}$ şi $h^{-1}; h = 1_{\mathcal{B}}$. Rezultă că pentru orice sort $s \in S$ au loc egalităţile $h_s; h_s^{-1} = 1_{A_s}$ şi $h_s^{-1}; h_s = 1_{B_s}$, adică funcţia h_s este inversabilă pentru orice $s \in S$, deci toate componentele h_s ale lui h sunt bijecţii.

Reciproc, presupunem că toate componentele h_s ale lui h sunt bijecţii. Prin urmare pentru orice $s \in S$ există funcţia $h_s^{-1} : B_s \longrightarrow A_s$ cu proprietăţile $h_s; h_s^{-1} = 1_{A_s}$ şi $h_s^{-1}; h_s = 1_{B_s}$. De aici notând $h^{-1} = \{h_s^{-1}\}_{s \in S}$ rezultă pentru orice $s \in S$ că

$$(h; h^{-1})_s = h_s; h_s^{-1} = 1_{A_s} = (1_A)_s \quad \text{şi} \quad (h^{-1}; h)_s = h_s^{-1}; h_s = 1_{B_s} = (1_B)_s$$

deci $h; h^{-1} = 1_A$ şi $h^{-1}; h = 1_B$.

Pentru a încheia demonstraţia mai trebuie arătat că funcţia S-sortată $h^{-1} : B \longrightarrow A$ este un morfism de Σ-algebre $h^{-1} : \mathcal{B} \longrightarrow \mathcal{A}$.

Fie $s_1 s_2 \dots s_n \in S^*$, $s \in S$, $\sigma \in \Sigma_{s_1 s_2 \dots s_n, s}$ şi $b_1 \in B_{s_1}$, $b_2 \in B_{s_2}$, $\dots$, $b_n \in B_{s_n}$. Deoarece h este morfism pentru elementele $h_{s_1}^{-1}(b_1), h_{s_2}^{-1}(b_2), \dots, h_{s_n}^{-1}(b_n)$ din A deducem

$h_s(A_\sigma(h_{s_1}^{-1}(b_1), h_{s_2}^{-1}(b_2), \dots, h_{s_n}^{-1}(b_n))) = B_\sigma(h_{s_1}(h_{s_1}^{-1}(b_1)), h_{s_2}(h_{s_2}^{-1}(b_2)), \dots, h_{s_n}(h_{s_n}^{-1}(b_n))) =$
$= B_\sigma(b_1, b_2, \dots, b_n)$.

Aplicând funcţia h_s^{-1} ambilor membri deducem

$$A_\sigma(h_{s_1}^{-1}(b_1), h_{s_2}^{-1}(b_2), \dots, h_{s_n}^{-1}(b_n)) = h_s^{-1}(B_\sigma(b_1, b_2, \dots, b_n))$$

deci $h^{-1} : \mathcal{B} \longrightarrow \mathcal{A}$ este morfism de Σ-algebre.

Propoziţie 3.3 *Compunerea a două izomorfisme $f : \mathcal{A} \longrightarrow \mathcal{B}$ şi $g : \mathcal{B} \longrightarrow \mathcal{C}$ este un izomorfism $f; g : \mathcal{A} \longrightarrow \mathcal{C}$. În plus*

$$(f; g)^{-1} = g^{-1}; f^{-1}$$

Demonstraţie: Pentru a demonstra că un morfism de la $\mathcal{M}$ la $\mathcal{N}$ este izomorfism având ca invers un alt morfism de la $\mathcal{N}$ la $\mathcal{M}$ este suficient să probăm că prin compunerea celor două morfisme, în ambele sensuri posibile, se obţin identităţi. Prin urmare folosind egalităţile $f; f^{-1} = 1_{\mathcal{A}}$, $f^{-1}; f = 1_{\mathcal{B}}$, $g; g^{-1} = 1_{\mathcal{B}}$ şi $g^{-1}; g = 1_{\mathcal{C}}$ deducem

$$(f; g); (g^{-1}; f^{-1}) = f; (g; g^{-1}); f^{-1} = f; 1_{\mathcal{B}}; f^{-1} = f; f^{-1} = 1_{\mathcal{A}} \text{ şi}$$

$$(g^{-1}; f^{-1}); (f; g) = g^{-1}; (f^{-1}; f); g = g^{-1}; 1_{\mathcal{B}}; g = g^{-1}; g = 1_{\mathcal{C}}$$

ceea ce arată că $f; g$ este izomorfism având inversul $g^{-1}; f^{-1}$.

3.2 Unicitatea abstracţie de un izomorfism a algebrelor libere

Dacă A este o submulţime a lui B numim **funcţie incluziune** a lui A în B funcţia $i : A \longrightarrow B$ definită prin $i(a) = a$ pentru orice $a \in A$. În acest caz folosim şi notaţia $i : A \hookrightarrow B$.

Teorema 3.4 *Două algebre liber generate de aceeaşi mulţime sunt izomorfe.*

Demonstraţie: Fie $\mathcal{A} = (A_s, A_\sigma)$ şi $\mathcal{B} = (B_s, B_\sigma)$ două Σ-algebre liber generate de X. Notăm cu $i_A : X \longrightarrow A$ şi $i_B : X \longrightarrow B$ funcţiile incluziune ale lui X în A, respectiv în B. Demonstraţia are patru paşi.

1. Deoarece algebra $\mathcal{A}$ este liber generată de X există un morfism $f : \mathcal{A} \longrightarrow \mathcal{B}$ cu proprietatea $i_A; f = i_B$.

Pasul 2 este asemănător cu primul, doar că se inversează rolul algebrelor $\mathcal{A}$ şi $\mathcal{B}$. La fel vor fi paşii 3 şi 4.

2. Deoarece algebra $\mathcal{B}$ este liber generată de X există un morfism $g : \mathcal{B} \longrightarrow \mathcal{A}$ cu proprietatea $i_B; g = i_A$.

3. Deoarece $i_A; (f; g) = (i_A; f); g = i_B; g = i_A = i_A; 1_{\mathcal{A}}$ şi deoarece $\mathcal{A}$ este algebră liber generată de X deducem, folosind partea de unicitate din definiţia algebrei libere, că $f; g = 1_{\mathcal{A}}$.

4. Deoarece $i_B; (g; f) = (i_B; g); f = i_A; f = i_B = i_B; 1_{\mathcal{B}}$ şi deoarece $\mathcal{B}$ este algebră liber generată de X deducem, folosind partea de unicitate din definiţia algebrei libere, că $g; f = 1_{\mathcal{B}}$.

Deci f şi g sunt izomorfisme inverse unul altuia.

Corolar 3.5 *Σ-algebra iniţială este unică abstracţie făcând de un izomorfism.*

3.3 Conceptul abstract de tip de date

Un **tip de date** se numeşte **abstract** dacă este unic determinat abstracţie făcând de un izomorfism. Se vede prin urmare că dând o signatură am dat implicit, prin algebra iniţială corespunzătoare signaturii, un tip abstract de date.

Vom da un exemplu cunoscut din algebra de liceu. Se ştie că numerele întregi formează un inel iniţial. Vă invităm să reflectaţi asupra următoarei definiţii a idei de număr întreg.

Se numeşte **număr întreg** un element al inelului iniţial.

Abstract înseamnă de fapt că nu ne interesează cum sunt scrise sau memorate datele.

Una dintre metodele prin care se poate defini un tip abstract de date este cel al algebrei iniţiale. Mai clar : este suficient să dăm o signatură şi eventual nişte axiome(ecuaţii condiţionate sau nu) deoarece algebra iniţială, a cărei existenţă este garantată de teoremele care vor fi prezentate mai târziu, este unic determinată abstracţie de un izomorfism, prin urmare este un tip abstract de date.

Tipul numerelor naturale poate fi definit abstract ca fiind semiinelul iniţial.

Observăm că o astfel de definiţie nu spune nimic despre scrierea numerelor. Ele pot fi scrise cu cifre arabe, cu cifre romane, în baza 2 ca în calculatoare sau altfel.

Definiţiile de mai sus, deşi corecte sunt uneori ineficiente, deoarece nu face posibilă execuţia de calcule sau execuţia este ineficientă. Prin urmare dorim alte definiţii echivalente care permit calculatorului să facă calcule care să fie cât mai eficiente. Vom exemplifica pentru numere naturale fără a intra în prea multe detalii.

3.3.1 Tipul abstract al numerelor naturale

Semiinelul este o mulţime M înzestrată cu două operaţii binare notate cu $+$ şi $*$ cu următoarele proprietăţi

1. $(M, +, 0)$ este monoid comutativ,
2. $(M, *, 1)$ este monoid,
3. $*$ este distributivă faţă de $+$ şi
4. $(\forall m \in M) m * 0 = 0 * m = 0$.

Mulţimea numerelor naturale cu adunarea şi inmulţirea uzuale este un semiinel iniţial, adică conceptul de semiinel caracterizează numele naturale ca tip abstract de date.

Pentru a scrie un program se preferă axiomele lui Peano. Chiar dacă Peano nu s-a gândit la programarea prin rescriere, baza programării declarative, se pare să fi scris primul program de acest gen.

Cel căruia i se atribuie prima punere în evidenţă a ideilor abstracte privind numerele naturale este F.W. Lawvere.

Considerăm signatura cu un singur sort *nat*, o singură constantă de sort nat şi o singură operaţie unară cu argument şi rezultat de sort nat:

sort nat .
op 0 : $\longrightarrow$ nat .
op s : nat $\longrightarrow$ nat .

Elementele algebrei iniţiale sunt

0, s(0), s(s(0)), s(s(s(0))), s(s(s(s(0)))), ...

şi ele reprezintă numerele naturale 0 1 2 3 4 ...

Propoziţia următoare în care conceptul de număr natural este folosit în înţelesul său clasic arată că numerele naturale formează un model al definiţiei abstracte. Acest fapt arată corectitudinea definiţiei abstracte pentru modelul clasic.

Propoziţie 3.6 *Fie $\mathcal{N} = (N, 0_N, s_N)$ algebra definită prin: N este mulţimea numerelor naturale, 0_N este numărul natural zero şi $s_N(n) = n + 1$ pentru orice număr natural n. Algebra $\mathcal{N}$ este iniţială.*

Demonstraţie: Fie $\mathcal{A} = (A, 0_A, s_A)$ o altă algebră pentru signatura de mai sus.
Definim funţia $h : N \longrightarrow A$ prin inducţie

$h(0_N) = 0_A$
$h(n+1) = s_A(h(n))$ pentru orice număr natural n.

Prima egalitate de mai sus şi $h(s_N(n)) = s_A(h(n))$ pentru orice număr natural n dovedesc că
$h : \mathcal{N} \longrightarrow \mathcal{A}$ este un morfism.

Probăm unicitatea. Fie $g : \mathcal{N} \longrightarrow \mathcal{A}$ un alt morfism. Arătăm prin inducţie că $g(n) = h(n)$ pentru orice n natural.
$g(0_N) = 0_A = h(0_N)$ şi presupunând $g(n) = h(n)$ deducem
$g(n+1) = g(s_N(n)) = s_A(g(n)) = s_A(h(n)) = h(s_N(n)) = h(n+1)$. $\odot$

Propoziţia anterioară ne arată cum pot fi definite numerele naturale prim metoda algebrei iniţiale ca tip abstract de date. Ea dovedeşte corectitudinea definiţiei de mai sus.

Deocamdată prin signatura de mai sus calculatorul învaţă numerele naturale dar nu ştie încă să calculeze. Pentru început să-l învăţăm să adune. Dacă introducem în signatură o operaţie binară +

op _+_ : nat nat ⟶ nat

nu realizăm nimic altceva decât să adaugăm la mulţimea de mai sus a numerelor naturale foarte mult gunoi. De exemplu, deoarece calculatorul nu ştie încă să adune, 0 + 0 este un nou element de care nu avem nevoie. Il învăţăm să adune dându-i urmatoarele două reguli de rescriere precedate de o declarare de variabile

var X Y : nat .
eq X + 0 = 0 .
eq X + s(Y) = s(X+Y) .

Trebuie să remarcăm diferenţa esenţială dintre o regulă de rescriere şi o egalitate. O regulă de rescriere se aplică numai de la stânga la dreapta. Simetria este unul dintre marii duşmani ai programării prin rescriere conducând la neterminarea programelor.

Ce părere aveţi despre comutativitate?

Să vedem cum efectuează maşina adunarea 2 + 2, adică:

$$\text{s(s(0)) + s(s(0)).}$$

Calculatorul nu poate aplica decât a doua regulă pentru X=s(s(0)) şi Y=s(0) ajungând la

$$\text{s(s(s(0)) + s(0)).}$$

Trebuie din nou aplicată a doua regulă de rescriere pentru X=s(s(0)) şi Y=0 ajungând la

$$\text{s(s(s(s(0)) + 0)).}$$

Acum se poate aplica numai prima regulă pentru X=s(s(0)) obţinând rezultatul s(s(s(s(0)))), adică 4. Calculatorul se opreşte deoarece nu se mai pot face rescrieri.

Corectitudinea acestei definiţii precum şi a celei care urmează va fi probată mai târziu, în secţiunea 8.1.

Calculatorul va şti să şi înmulţească dacă mai introducem o operaţie binară şi două reguli de rescriere

op _*_ : nat nat ⟶ nat .
eq X * 0 = 0 .
eq X * s(Y) = X*Y + X .

Uneori, în programarea prin rescriere, dacă dorim să scriem un program, partea cea mai dificilă este definirea abstractă a tipurilor de date cu care lucrăm. Plecând de la noţiunea intuitivă dată de o algebră $\mathcal{D}$ a datelor, trebuie să găsim signatura şi eventual ecuaţiile pentru care $\mathcal{D}$ devine algebră iniţială.

Exemplul numerelor naturale este doar un început. Vă propunem de exemplu să definiţi relaţia de ordine ca operaţie $<$: nat nat ⟶ *bool*. Piatra de încercare va fi însă operaţia de împărţire.

4 SEMANTICA ALGEBREI INIŢIALE

Menţionăm aici pe marele informatician, regretatul Joseph Goguen, profesorul şi prietenul multor români, care spunea că "semantica algebrei iniţiale" este una dintre cele mai frumoase idei ale sale [**?**].

Metoda semanticii algebrei initiale este o simplificare a metodei mai clasice a semanticii denotaţionale, numită şi semantică matematică.

Metoda semanticii algebrei inţiale se aplică pentru limbaje definite printr-o gramatică independentă de context $G = (N, T, P, a)$, unde N este mulţimea neterminalelor, T muţimea terminalelor, P mulţimea producţiilor şi a axioma gramaticii. Ea spune că **pentru a defini semantica limbajului gramaticii G este suficient să dăm o G-algebră $\mathcal{S} = (\{S_n\}_{n\in N}, \{S_p\}_{p\in P})$**, numită în continuare **algebră semantică**.

Pentru a înţelege afirmaţia de mai sus trebuie să intrăm puţin în amănunte. Fie $\mathcal{A}$ algebra arborilor de derivare. Deoarece $\mathcal{A}$ este algebră inţială există un unic morfism de G-algebre

$$M : \mathcal{A} \longrightarrow \mathcal{S}.$$

Dat un cuvânt c din limbajul gramaticii G există un arbore de derivare arb cu rădăcina etichetată cu a a cărui frontieră este c. Semantica cuvântului c este prin definiţie $M_a(arb)$.

Menţionăm că metoda este bine definită numai pentru gramaticile neambigue, fapt care asigură unicitatea arborelui arb. Acest aspect este mai degrabă legat de analiza sintactică.

Trecem la exemple care vor clarifica şi mai mult ideile de mai sus.

4.1 Semantica unui şir de cifre ca număr natural

Vom considera o gramatică G care generează şirurile finite nevide de cifre zecimale, considerate ca terminale. Gramatica are două neterminale <cifra> şi <nat> ultima fiind şi axiomă a gramaticii. Descriem în continuare producţiile gramaticii cărora le dăm un nume

ci	<cifra>	$\longrightarrow$	i	pentru orice cifră zecimală i
n1	<nat>	$\longrightarrow$	<cifra>	
n2	<nat>	$\longrightarrow$	<nat> <cifra>	

Vom defini algebra semantică explicând de ce o definim astfel. Deoarece gramatica are două neterminale, signatura asociată are două sorturi, prin urmare algebra semantică trebuie să aibă ca suport două mulţimi. Semantica unei cifre, care nu este nimic altceva decât un semn, este numărul reprezentat de cifră. Prin urmare suportul corespunzător neterminalului <cifra> trebuie să conţină măcar numerele de la zero la nouă. Deci prin definiţie

$$S_{<cifra>} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Deoarece semantica unui şir finit de cifre este numărul natural pe care acest şir îl reprezintă, suportul corespunzător neterminalului <nat> trebuie să conţină măcar mulţimea numerelor naturale. Deci prin definiţie

$$S_{<nat>} = \text{este mulţimea numerelor naturale.}$$

Să trecem la definirea operaţiilor algebrei semantice.

Deoarece producţia ci corespunzătoare cifrei i nu are nici un neterminal în membrul drept, operaţia corespunzătoare ei este o operaţie fără argumente, adică un element din $S_{<cifra>}$

$$S_{ci} \text{ este numărul natural reprezentat de cifra } i.$$

Producţia n1 spune că orice cifră este un şir dr cifre. Deoarece valoarea numărului reprezentat de un şir de lungime unu este egală cu valoarea numărului reprezentat de singura cifră din şir şi semantica lor trebuie să fie aceeaşi. Deci prin definiţie

$$S_{n1} \text{ este funcţia incluziune de la } S_{<cifra>} \text{ la } S_{<nat>}.$$

Producţia n2 ne spune că prin adăugarea la dreapta unui şir de cifre, să spunem s1, a unei cifre, să spunem c, se obţine un nou şir de cifre s2 = s1c. Operaţia corespunzătoare producţiei n2 trebuie să fie o funcţie

$$S_{n2} : S_{<nat>} \times S_{<cifra>} \longrightarrow S_{<nat>}$$

definită pentru numele n şi m prin

$$S_{n2}(n, m) = 10n + m$$

deoarece valoarea ca număr a şirului s2 este egală cu de zece ori valoarea ca număr a şirului s1 plus valoarea ca număr a cifrei.

Cu aceasta semantica este complet definită. Vom exemplifica pentru şirul 023. Arborele de derivare pentru acest şir este

$$A_{n2}(A_{n2}(A_{n1}(A_{c0}), A_{c2}), A_{c3}).$$

Să-i aplicăm morfismul semantic acestui arbore.

$$M_{<nat>}(A_{n2}(A_{n2}(A_{n1}(A_{c0}), A_{c2}), A_{c3})) = S_{n2}(S_{n2}(S_{n1}(S_{c0}), S_{c2}), S_{c3}) = S_{n2}(S_{n2}(S_{n1}(0), 2), 3) =$$

$$= S_{n2}(S_{n2}(0, 2), 3) = S_{n2}(10 \times 0 + 2, 3) = S_{n2}(2, 3) = 10 \times 2 + 3 = 23$$

Ce sa întâmplă dacă în membrul drept al producţiei n2 scriem $< cifra >< nat >$?

4.2 Un calculator de buzunar

Vom da un alt exemplu, cel al unui minicalculator pe faţa căruia se află un mic ecran pe care se poate afişa un singur număr şi mai multe butoane:

ON pentru pornirea calculatorului
OFF pentru oprirea calculatorului
0 1 2 3 4 5 6 7 8 9 pentru cifrele zecimale
+ × pentru adunare şi înmulţire
M pentru unica celulă de memorie a calculatorului
( ) pentru parantezele necesare în scrierea expresiilor
IF , pentru o anumită expresie condiţionată
E pentru comanda de evaluarea a unei expresii

Remarcăm că butoanele diferite de ON, OFF şi E sunt folosite pentru introducerea expresiilor care vor fi evaluate la apăsarea butonului E. Sintaxa expresiilor este definită de producţiile r1-r6 ale gramaticii de mai jos. Expresiile sunt construite pornind de la şirurile finite de cifre zecimale reprezentând numere naturale(producţia r1), variabila M(producţia r2) şi recursiv, alte expresii cuprinse între paranteze(producţia r6). Pentru construţia expresiilor sunt folosite numai operaţiile de adunare şi înmulţire(producţiile r3 şi r4). Producţia r5 permite construirea unor expresii condiţionate unde condiţia reprezentată de prima expresie permite o alegere între celelalte două expresii.

Vom considera o gramatică G care generează limbajul programelor care pot fi executate de minicalculator. Ea extinde gramatica din exemplul precedent. Numele butoanelor de mai sus sunt chiar terminalele gramaticii.

Gramatica are cinci neterminale <cifra>, <nat>, <exp> pentru expresii, <inst> pentru anumite porţiuni de programe şi <program> care este şi axiomă a gramaticii. Descriem în continuare producţiile gramaticii cărora le dăm un nume pe care l-am scris la începutul rândului, în faţa producţiei.

ci	<cifra>	$\longrightarrow$	i	pentru orice cifră zecimală i
n1	<nat>	$\longrightarrow$	<cifra>	
n2	<nat>	$\longrightarrow$	<nat> <cifra>	
r1	<exp>	$\longrightarrow$	<nat>	
r2	<exp>	$\longrightarrow$	M	
r3	<exp>	$\longrightarrow$	<exp> + <exp>	
r4	<exp>	$\longrightarrow$	<exp> × <exp>	
r5	<exp>	$\longrightarrow$	IF <exp>,<exp>,<exp>	
r6	<exp>	$\longrightarrow$	(<exp>)	
r7	<inst>	$\longrightarrow$	<exp> E OFF	
r8	<inst>	$\longrightarrow$	<exp> E <inst>	
r9	<program>	$\longrightarrow$	ON <inst>	

Pentru a da semantica este necesar să explicăm cum functionează calculatorul. Se porneşte calculatorul apăsând butonul ON, moment în care se iniţializează unica celulă de memorie cu zero. Se introduce o expresie care la apăsarea butonului E se evaluează. Rezultatul evaluarii este afişat pe ecran şi introdus în unica celulă de memorie în locul vechii valori. Se introduce altă expresie, se apasă butonul E şi aşa mai departe. La final se apasă butonul OFF pentru închiderea calculatorului. Mai remarcăm că se calculează numai cu numere naturale.

Vom defini algebra semantică. Deoarece signatura asociată are cinci sorturi, algebra semantică trebuie să aibă ca suport cinci mulţimi, primele două fiind cele de mai sus $S_{<cifra>}$ şi $S_{<nat>}$.

Expresiile sunt făcute pentru a fi evaluate, prin urmare prima idee ar fi ca semantica unei expresii să fie rezultatul evaluării, adică un număr. Dar oare ce număr se obţine prin evaluarea expresiei M+3. Trebuie să menţionăm că fiecare apariţie a lui M este înlocuită în timpul evaluării cu valoarea care se află în unica celulă de memorie. Prin urmare rezultatul evaluării lui M+3 depinde de conţinutul celulei de memorie. Prin urmare semantica expresiei M+3 este funcţia $f : N \longrightarrow N$ defină prin $f(x) = x + 3$ care asociază numărului x care se află în celula de memorie rezultatul $x + 3$ al evaluării acesteia. Prin urmare prin definiţie $S_{<exp>}$ este mulţimea funcţiilor de la mulţimea numerelor naturale la ea însăşi.

O instrucţiune, generată de producţiile r7 şi r8, comandă evaluarea unui şir finit de expresii. Semantica instrucţiunii este, prin definiţie, şirul numerelor care apar pe ecran în timpul execuţiei instrucţiunii, adică un şir finit şi nevid de numere naturale. Mulţimea acestor şiruri o notăm cu N^+. Rezultatul evaluării depinde evident de valoarea din memorie din momentul începerii execuţiei instrucţiunii, prin urmare

$$S_{<inst>} = \{f : N \longrightarrow N^+ \mid f \text{ este funcţie.}\}$$

Un program, generat de producţia r9, diferă de o instrucţiune prin faptul că se introduce zero în celula de memorie înainte de execuţia instrucţiunii pe care o conţine, prin urmare şirul de numere care apar pe ecran în timpul execuţiei programului nu mai depinde de conţinutul memoriei, fiind un element din N^+, deci $S_{<program>} = N^+$.

Folosind următoarele notaţii

m este un număr natural reprezentând conţinutul celulei de memorie
n este un număr natural
e este o funcţie de la N la N reprezentând semantica unei expresii
ı este o funcţie de la N la N^+ reprezentând semantica unei instrucţiuni

definiţiile pentru operaţii, fără a le mai repeta pe cele de mai sus sunt:

$S_{r1}(n)(m) = n$
$S_{r2}(m) = m$
$S_{r3}(e1, e2)(m) = e1(m) + e2(m)$
$S_{r4}(e1, e2)(m) = e1(m) \times e2(m)$
$S_{r5}(e1, e2, e3)(m) =$ dacă $e1(m) = 0$ atunci $e2(m)$ altfel $e3(m)$
$S_{r6}(e) = e$
$S_{r7}(e)(m) = e(m)$
$S_{r8}(e, ı)(m) = e(m)ı(e(m))$
$S_{r9}(ı) = ı(0)$

4.3 Arbori de derivare

Reamintim proprietăţile unui arbore de derivare dintr-o gramatică independentă de context:
1. este un arbore finit şi local ordonat, adică succesorii fiecărui nod sunt într-o ordine totală;
2. are noduri etichetate cu terminale sau neterminale, rădăcina fiind etichetată cu un neterminal;
3. orice nod etichetat cu un terminal nu are nici un succesor;
4. pentru orice nod, dacă este etichetat cu un neterminal n, atunci perechea formată din n şi cuvântul format de etichetele $s_1, s_2, \ldots, s_k$ ale succesorilor săi formează o producţie $(n, s_1 s_2 \ldots s_{k-1} s_k) \in P$.

4.3.1 O gramatică pentru expresii

Definim o gramatică independentă de context pentru expresiile construite cu variabilele x, y şi z, cu operaţiile binare de adunare şi înmulţire şi cu paranteze. Gramatica trebuie constrită respectand urmatoarele condiţii privind expresiile:

1. înmulţirile se fac înaintea adunărilor
2. adunările se fac de la stânga la dreapta
3. înmulţirile de la dreapta la stânga.

Mulţimea terminalelor este $\{x, y, z, (,), +, *\}$. Fie $N = \{V, F, T, E, P\}$ mulţimea neterminalelor. Semnificaţia lor este următoarea:

V reprezintă variabilele
F de la factor, reprezintă cele mai "mici" elemente folosite în construcţia expresiilor
T de la termen, reprezintă un produs de factori
E de la expresie, reprezintă o sumă de termeni
P de la program.

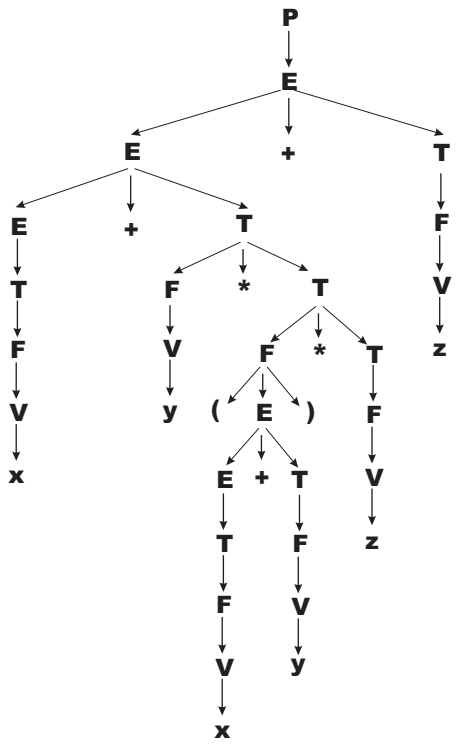
Continuăm cu mulţimea **regulilor(producţiilor)** gramaticii:

0. $P \to E$ **1.** $V \to x$ **2.** $V \to y$ **3.** $V \to z$ **4.** $F \to V$
5. $F \to (E)$ **6.** $T \to F$ **7.** $T \to F * T$ **8.** $E \to T$ **9.** $E \to E + T$

Observaţi producţiile 7 şi 9 pentru a vedea cum se precizează ordinea de execuţie a operaţiilor de acelaşi fel. Neterminalul din membrul stâng se află în dreapta, respectiv în stânga semnului operaţiei.

4.3.2 Un exemplu

Să se construiască arborele de derivare pentru expresia $x + y * (x + y) * z + z$.



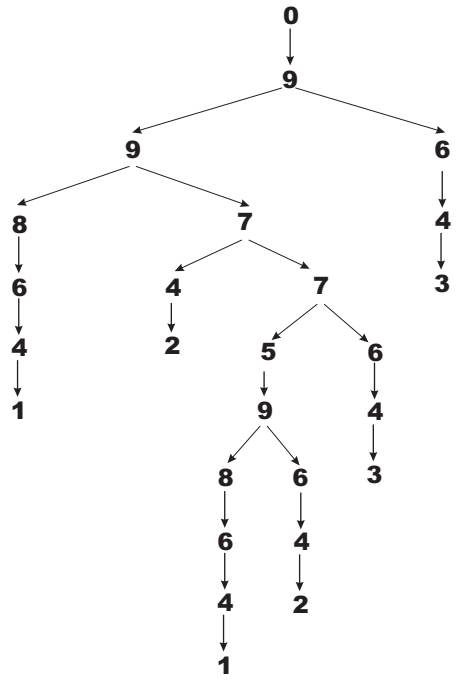
Pentru a reface expresia, arborele se parcurge în inordine.

Dacă veţi incerca să construiţi alt arbore de derivare pentru aceeaşi expresie veţi constata ca acest lucru nu este posibil. Prin urmare modul de construcţie al expresiei va impune o anumită ordine în evaluarea acesteia.

Folosind numerele care denumesc producţiile ca nume ale operaţiilor din algebra arborilor de derivare arborele de mai sus este

$$A = 0(9[9\{a, 7(4[2], 7[5\{9(a,6[4(2)])\},b]) \},b])$$

unde a = 8(6[4(1)]) şi b = 6(4(3)).



4.4 Scrierea poloneză inversă

Semantica pe care o dăm unui limbaj depinde de scopul pe care-l urmărim. Pentru a ilustra această idee vom da două semantici pentru expresiile definite mai sus.

Prima semantică va da scrierea poloneză inversă pentru expresii. A doua semantică va fi dată în secţiunea următoare.

Algebra semantică va avea toate suporturile egale cu semigrupul liber generat de $\{x, y, z, +, *\}$, sau mai popular mulţimea şirurilor finite nevide de semne din $\{x, y, z, +, *\}$. Operaţiile sunt următoarele:

$0 = 4 = 5 = 6 = 8$ sunt aplicaţia identitate,

$1 = x$, $2 = y$ şi $3 = z$,

$7(\alpha, \beta) = \alpha\beta*$ şi $9(\alpha, \beta) = \alpha\beta+$.

Notăm cu $\mathcal{S}$ algebra semantică de mai sus, cu $\mathcal{A}$ algebra arborilor de derivare şi cu $C : \mathcal{A} \longrightarrow \mathcal{S}$ unicul morfism de algebre existent. Morfismul C este modelarea algebrică a rescrierii expresiei în scrierea poloneză inversă.

Conform metodei semanticii algebrei iniţiale rezultă că C(A) este scrierea poloneză înversă a expresiei $x + y * (x + y) * z + z$.

Calculele sunt următoarele

C(a) $= 8(6[4(1)]) = x$ şi C(b) $= 6(4(3)) = z$

C(A) $= C(0(9[9\{$a, $7(4[2], 7[\ 5\{9($a,$6[4(2)])\}$,b $])\ \},$b$]))$ $= 9[9\{x,\ 7(\ y\ ,\ 7[\ 9(x,y)\ ,z\ ])\ \},z] =$

$9[9\{x,\ 7(\ y\ ,\ 7[xy+,z])\ \},z] = 9[9\{x,\ 7(y,xy + z*)\ \},z] = 9[9\{x,\ yxy + z**\ \},z] = 9[xyxy + z**+,z] =$

$xyxy + z** + z +$.

Rezultatul final confirmă corectitudinea şi eficacitatea metodei.

4.5 Compilare

Un compilator care va produce un program pentru evaluarea expresiilor si tipărirea rezultatului evalării va fi modelat printr-un morfism.

Vom pune în evidenţă elementele care apar în programele obţinute prin compilare. **R** este un registru din memorie. **P** este un pointer către primul loc liber al stivei utilizate în timpul evaluării. Ori de câte ori se pune un element în

stivă pointerul creşte automat cu o unitate. Ori de câte ori se scoate un element din stivă pointerul scade automat cu o unitate. Presupunem că la începutul execuţiei programului pointerul indică primul loc liber din stivă.

Presupunem că limbajul de asamblare include următoarele instrucţiuni:

Ad R	scoate valoarea din vârful stivei, o adună cu conţinutul lui R şi pune rezultatul în R
Mu R	scoate valoarea din vârful stivei, o inmulţeşte cu conţinutul lui R şi pune rezultatul în R
ld R	scoate valoarea din vârful stivei şi o pune în R
st R	valoarea din R este pusă în vârful stivei
print	tipăreşte valoarea din vârful stivei

Semantica unei expresii va fi programul care evaluează expresia şi imprimă rezultatul evaluării. Programul care evaluează o expresie are ca efect plasarea în vârful stivei a rezultatului evaluării acesteia şi avansarea pointerului. Pentru definirea acesteia se va folosi metoda semanticii algebrei iniţiale.

Toate cele 5 suporturi ale algebrei semantice, corespunzătoare neterminalelor, sunt identice şi coincid cu mulţimea bucăţilor de program(şiruri de instrucţiuni separate prin ;). Astfel de bucăţi de programe vor fi notate cu litere greceşti.

Cele 10 operaţii ale algebrei semantice, corespunzătoare producţiilor sunt:

1_S	=	st x
2_S	=	st y
3_S	=	st z
$4_S(\alpha)$	=	α
$5_S(\alpha)$	=	α
$6_S(\alpha)$	=	α
$7_S(\alpha,\beta)$	=	α ; β ; ld R ; Mu R ; st R
$8_S(\alpha)$	=	α
$9_S(\alpha,\beta)$	=	α ; β ; ld R ; Ad R ; st R
$0_S(\alpha)$	=	α ; print

Notăm cu $\mathcal{S}$ algebra semantică de mai sus, cu $\mathcal{A}$ algebra arborilor de derivare şi cu $C : \mathcal{A} \longrightarrow \mathcal{S}$ unicul morfism de algebre existent. Morfismul C este modelarea algebrică a compilatorului.

Conform metodei semanticii algebrei iniţiale rezultă că C(A) este programul care evaluează şi tipăreşte rezultatul pentru expresia $x + y * (x + y) * z + z$.

4.5.1 Program

Prezentăm calculele care dovedesc afirmaţia de mai sus.

$$C(\text{a}) = 8_S(6_S[4_S(1_S)]) = 1_S \quad \text{şi} \quad C(\text{b}) = 6_S(4_S(3_S)) = 3_S$$

Notând c = 7[5{9(a,6[4(2)])},b] deducem

C(c) = 7_S[5_S{9_S(1_S,6_S[4_S(2_S)])},3_S] =
7_S[9_S(1_S,2_S),3_S] =
9_S(1_S,2_S) ; 3_S ; ld R ; Mu R ; st R =
1_S ; 2_S ; ld R ; Ad R ; st R ; 3_S ; ld R ; Mu R ; st R =
st x ; st y ; ld R ; Ad R ; st R ; st z ; ld R ; Mu R ; st R

Deoarece

$$A = 0(9[9\{a, 7(4[2], c)\}, b])$$

deducem

C(A) = 0_S(9_S[9_S{1_S, 7_S(2_S,C(c))},3_S]) =
9_S[9_S{1_S, 7_S(2_S,C(c)) },3_S] ; print =
9_S{1_S, 7_S(2_S,C(c)) } ; 3_S ; ld R ; Ad R ; st R ; print =
1_S ; 7_S(2_S,C(c)) ; ld R ; Ad R ; st R ; 3_S ; ld R ; Ad R ; st R ; print =
1_S ; 2_S ; C(c) ; ld R ; Mu R ; st R ; ld R ; Ad R ; st R ; 3_S ; ld R ; Ad R ; st R ; print =

st x ; st y ; st x ; st y ; ld R ; Ad R ; st R ; st z ; ld R ; Mu R ;
st R ; ld R ; Mu R ; st R ; ld R ; Ad R ; st R ; st z ; ld R ; Ad R ; st R ; print

4.5.2 Optimizare cod

Programul de mai sus poate fi simplificat. Observăm că grupul de instrucţiuni "st R ; ld R" are efect cumulat nul ceea ce conduce la programul

st x ; st y ; st x ; st y ; ld R ; Ad R ; st R ; st z ; ld R ; Mu R ;
Mu R ; Ad R ; st R ; st z ; ld R ; Ad R ; st R ; print

5 Introducere în semantica logicii ecuaţionale

Logica ecuaţională, în varianta fără sorturi, a fost intens studiată în prima jumătate a secolului XX când i-a fost demonstrată completitudinea.

5.1 Ecuaţii

Să analizăm conceptul de axiomă aşa cum apare el în algebră. De exemplu comutativitatea şi asociativitatea se scriu

$$(\forall x \forall y) x?y = y?x \qquad (\forall x \forall y \forall z) x?(y?z) = (x?y)?z.$$

Ce sunt acestea? Sunt egalităţi de două expresii cuantificate universal prin mulţimea variabilelor conţinute în cele două expresii. Deci o astfel de axiomă are forma

$$(\forall X) l \stackrel{\circ}{=} r$$

unde l şi r sunt elemente din algebra liber generată de mulţimea X de variabile. Semnul $\stackrel{\circ}{=}$ este utilizat în loc de $=$, între două expresii, pentru a marca faptul că egalitatea poate fi şi falsă, fapt pentru care vom folosi denumirea de **egalitate formală** în loc de egalitate. În cazul multisortat, cei doi membri ai unei egalităţi formale trebuie să fie de acelaş sort. Uneori scriem $l \stackrel{\circ}{=}_s r$ pentru a marca sortul comun s al celor doi membri.

Ce înseamnă că o axiomă este adevărată într-o algebră $\mathcal{D}$? Intuitiv este necesar ca rezultatul evaluării celor două expresii l şi r în algebra $\mathcal{D}$ să fie acelaşi indiferent de valorile date în $\mathcal{D}$ variabilelor din X. Ţinând cont că a da valori variabilelor din X în algebra $\mathcal{D}$ este echivalent cu a da un morfism de la $T_\Sigma(X)$ la $\mathcal{D}$, această idee intuitivă conduce la:

Definiţia 5.1 Axioma $(\forall X) l \stackrel{\circ}{=} r$ este satisfăcută în algebra $\mathcal{D}$ dacă şi numai dacă pentru orice morfism $h : T_\Sigma(X) \longrightarrow \mathcal{D}$ este adevărată egalitatea $h(l) = h(r)$.

Dacă $(\forall X) l \stackrel{\circ}{=} r$ este satisfăcută în algebra $\mathcal{D}$, vom folosi notaţia

$$\mathcal{D} \models_\Sigma (\forall X) l \stackrel{\circ}{=} r$$

şi mai spunem că algebra $\mathcal{D}$ satisface $(\forall X) l \stackrel{\circ}{=} r$. $\odot$

În continuare vom folosi pentru $(\forall X) l \stackrel{\circ}{=} r$ termenul de **ecuaţie** în locul celui de axiomă, pentru a ne conforma cu terminologia internaţională.

În plus vor intra în joc şi aşa zisele **ecuaţii condiţionate**. De exemplu

$$(\forall x \forall y \forall z)(x * y = x * z \Rightarrow y = z)$$

ceea ce corespunde axiomei de simplificare la stânga care este adevărată în orice grup sau în orice monoid liber.

5.2 Ecuaţii condiţionate

În logica ecuaţională o axiomă poate fi o implicaţie

$$a_1 = c_1, a_2 = c_2, \ldots, a_n = c_n \Rightarrow a = c$$

unde ipoteza este o conjuncţie de egalităţi formale şi concluzia o egalitate formală. Toată implicaţia este cuantificată universal, fapt care nu apare scris mai sus. În acest cadru o axiomă, numită în continuare şi ecuaţie condiţionată a logicii ecuaţionale, poate fi scrisă sub forma

$$(\forall X)\ a \stackrel{\circ}{=}_s c \textbf{ if } H$$

unde a şi c sunt elemente de acelaşi sort s, iar H este o mulţime **finită** de egalităţi formale din algebra liber generată de mulţimea X de variabile. Ipoteza implicaţiei este dată de conjuncţia tuturor egalităţilor formale din mulţimea H.

Pentru a verifica dacă o algebră $\mathcal{D}$ satisface axioma de mai sus se dau valori arbitrare variabilelor din X în $\mathcal{D}$ fapt ce poate fi făcut printr-o funcţie arbitrară $f : X \longrightarrow \mathcal{D}$ sau echivalent printr-un morfism arbitrar $h : T_\Sigma(X) \longrightarrow \mathcal{D}$. Apoi se evaluează expresiile din H pentru a se verifica dacă rezultatul evaluarii conduce la egalităţi adevărate, caz în care trebuie ca $h_s(a) = h_s(c)$. Deci Σ-algebra $\mathcal{D}$ satisface ecuaţia condiţionată $(\forall X) a \stackrel{\circ}{=}_s c$ **if** H fapt notat prin

$$\mathcal{D} \models_\Sigma (\forall X)\ a \stackrel{\circ}{=}_s c \textbf{ if } H$$

dacă şi numai dacă

$$(\forall h : T_\Sigma(X) \longrightarrow \mathcal{D})\ \{(\forall u \stackrel{\circ}{=}_t v \in H) h_t(u) = h_t(v)\} \text{ implică } h_s(a) = h_s(c).$$

Credem că este bine să menţionăm din nou diferenţa esenţială între semnele $\stackrel{\circ}{=}_s$ şi $=$, diferenţă care va fi menţinută constant pe parcursul întregului text. Egalul peste care s-a pus un cerculeţ ($\stackrel{\circ}{=}_s$) indică o egalitate formală care poate fi adevărată sau falsă. Egalul $=$ are semnificaţia uzuală indicând deobicei o egalitate adevărată.

Formalizăm cele de mai sus în definiţiile următoare.

Definiţia 5.2 O *ecuaţie condiţionată* este

$$(\forall X) l \stackrel{\circ}{=}_s r \textbf{ if } H$$

unde X este o mulţime S-sortată de variabile, l şi r sunt două elemente de sort s din $T_\Sigma(X)$ iar H o mulţime **finită** de egalităţi formale din $T_\Sigma(X)$. $\odot$

O ecuaţie condiţionată în care $H = \emptyset$ devine necondiţionată şi este numită pe scurt *ecuaţie*. În acest caz scriem doar $(\forall X) l \stackrel{\circ}{=}_s r$ în loc de $(\forall X) l \stackrel{\circ}{=}_s r$ **if** $\emptyset$.

Definiţia 5.3 Algebra $\mathcal{D}$ *satisface* ecuaţia condiţionată $(\forall X) l \stackrel{\circ}{=}_s r$ **if** H, fapt notat prin

$$\mathcal{D} \models_\Sigma (\forall X) l \stackrel{\circ}{=}_s r \textbf{ if } H$$

dacă pentru orice morfism $h : T_\Sigma(X) \longrightarrow \mathcal{D}$ pentru care $h_{s'}(u) = h_{s'}(v)$ pentru orice $u \stackrel{\circ}{=}_{s'} v \in H$, are loc egalitatea $h_s(l) = h_s(r)$. $\odot$

În cele ce urmează indicele Σ din $\models_\Sigma$ va fi omis. El va fi menţionat atunci când este pericol de confuzie.

Observăm că $\mathcal{D} \models (\forall X) l \stackrel{\circ}{=}_s r$ dacă şi numai dacă $h_s(l) = h_s(r)$ pentru orice morfism $h : T_\Sigma(X) \longrightarrow \mathcal{D}$.

5.3 Γ–algebre

În continuare fixăm o mulţime Γ de ecuaţii condiţionate, numite axiome.

Definiţia 5.4 Spunem că algebra $\mathcal{D}$ satisface Γ sau că $\mathcal{D}$ este o Γ-algebră şi scriem $\mathcal{D} \models \Gamma$ dacă $\mathcal{D}$ satisface toate ecuaţiile condiţionate din Γ.

Un morfism de Σ-algebre între două Γ-algebre se numeşte morfism de Γ-algebre sau mai scurt Γ-morfism. $\odot$

6 CONGRUENŢE, ALGEBRE CÂT, PROIECTIVITATE

Dacă $f : A \longrightarrow B$ este o funcţie relaţia $Ker(f)$ definită prin

$$Ker(f) = \{(a, b) \in A \times A : f(a) = f(b)\}$$

se numeşte **echivalenţa nucleară** a lui f, sau mai pe scurt nucleul lui f.

Echivalenţa nucleară a unei funcţii este o relaţie de echivalenţă. În cazul multisortat nucleul este luat pe componente, adică pentru fiecare sort în parte.

6.1 Congruenţe

O relaţie de echivalenţă într-o Σ-algebră (A_s, A_σ) este de fapt o familie de relaţii de echivalenţe, câte una pentru fiecare mulţime A_s. O congruenţă este o relaţie de echivalenţă care este compatibilă cu operaţiile algebrei. De fapt compatibilatea trebuie cerută numai pentru operaţiile cu argumente căci pentru o constantă $A_\sigma \sim A_\sigma$ rezultă din reflexivitate.

Definiţia 6.1 O congruenţă $\sim = \{\sim_s\}_{s\in S}$ în Σ-algebra (A_s, A_σ) este o relaţie de echivalenţă cu proprietatea pentru orice $s_1 s_2 \dots s_n \in S^*$, pentru orice $s \in S$, pentru orice $\sigma \in \Sigma_{s_1 s_2 \dots s_n, s}$, dacă $a_i \sim_{s_i} b_i$ în A_{s_i} pentru orice $1 \le i \le n$, atunci

$$A_\sigma(a_1, a_2, \dots, a_n) \sim_s A_\sigma(b_1, b_2, \dots, b_n).$$

Cea mai mică congruenţă este relaţia de egalitate. Cea mai mare congruenţă este relaţia totală.

Propoziţie 6.2 *Dacă $h : \mathcal{A} \longrightarrow \mathcal{B}$ este un Σ-morfism, atunci $Ker(h)$ este o congruenţă.*

Demonstraţie: *Fie $s_1 s_2 \dots s_n \in S^*$, $s \in S$ şi $\sigma \in \Sigma_{s_1 s_2 \dots s_n, s}$. Presupunem că $a_i \; Ker(h_{s_i}) \; b_i$ în A_{s_i} pentru orice $1 \le i \le n$. Rezultă că $h_{s_i}(a_i) = h_{s_i}(b_i)$ pentru orice $1 \le i \le n$. Prin urmare*

$h_s(A_\sigma(a_1, a_2, \dots, a_n)) = B_\sigma(h_{s_1}(a_1), h_{s_2}(a_2), \dots, h_{s_n}(a_n)) =$
$B_\sigma(h_{s_1}(b_1), h_{s_2}(b_2), \dots, h_{s_n}(b_n)) = h_s(A_\sigma(b_1, b_2, \dots, b_n))$

aşadar $A_\sigma(a_1, a_2, \dots, a_n) \; Ker(h_s) \; A_\sigma(b_1, b_2, \dots, b_n)$. Deci $Ker(h)$ este congruenţă.

Propoziţie 6.3 *Orice intersecţie de congruenţe este tot o congruenţă.*

Demonstraţie: *Fie $\{\sim^k\}_{k\in K}$ o mulţime de congruenţe şi*

$$\sim \; = \bigcap_{k\in K} \sim^k .$$

Probăm că $\sim$ este congruenţă.

Fie $s_1 s_2 \dots s_n \in S^*$, fie $s \in S$, fie $\sigma \in \Sigma_{s_1 s_2 \dots s_n, s}$ şi $a_i \sim b_i$ în A_{s_i} pentru orice $1 \le i \le n$. Rezultă că $a_i \sim^k b_i$ pentru orice $k \in K$ şi orice $1 \le i \le n$.

Pentru orice $k \in K$ deoarece $\sim^k$ este congruenţă, fiindcă $a_i \sim^k b_i$ în A_{s_i} pentru orice $1 \le i \le n$ deducem

$$A_\sigma(a_1, a_2, \dots a_n) \sim^k A_\sigma(b_1, b_2, \dots, b_n).$$

Deci

$$A_\sigma(a_1, a_2, \dots a_n) \sim A_\sigma(b_1, b_2, \dots, b_n). \; \odot$$

6.2 Algebre cât

6.2.1 Proprietatea de universalitate a mulţimii cât.

Propoziţie 6.4 *Fie $\sim$ o relaţie de echivalenţa în mulţimea A. Fie $A/\sim$ mulţimea cât şi $\rho : A \longrightarrow A/\sim$ surjecţia naturală de factorizare. Pentru orice funcţie $f : A \longrightarrow B$*

dacă $\sim \subseteq Ker(f)$, atunci există o unică funcţie $f^\# : A/\sim \longrightarrow B$ cu proprietatea $\rho; f^\# = f$.

Demonstraţie: Definim funcţia $f^\#$ pentru orice $a \in A$ prin

$$f^\#(\rho(a)) = f(a).$$

Definiţia este corectă deoarece $b \sim a$ implică, folosind ipoteza $\sim \subseteq Ker(f)$, că $bKer(f)a$, adică $f(a) = f(b)$.

Egalitatea din enunţ $\rho; f^\# = f$ rezultă direct din definiţie.

Unicitatea rezultă din surjectivitatea funcţiei ρ de factorizare. Fie $g : A/\sim \longrightarrow B$ o funcţie cu proprietatea $\rho; g = f$. Rezultă că pentru orice $a \in A$

$$g(\rho(a)) = f(a).$$

Prin urmare $g(\rho(a)) = f^\#(\rho(a))$ pentru orice $a \in A$. Deoarece orice element din $A/\sim$ este de forma $\rho(a)$ cu $a \in A$ rezultă că $g = f^\#$. $\odot$

6.2.2 Proprietatea de universalitate a algebrei cât.

Fie (A_s, A_σ) o Σ-algebră şi $\sim$ o congruenţă. Definim operaţiile algebrei cât

$$\mathcal{A}/\sim \ = (\{A_s/\sim\}_{s\in S}, A/\sim_\sigma)$$

pentru orice $s_1 s_2 \dots s_n \in S^*$, $s \in S$, $\sigma \in \Sigma_{s_1 s_2 \dots s_n, s}$ şi $a_i \in A_{s_i}$ pentru orice $1 \leq i \leq n$ prin

$$A/\sim_\sigma(\rho(a_1), \rho(a_2), \dots, \rho(a_n)) = \rho(A_\sigma(a_1, a_2, \dots, a_n)).$$

Trebuie să probăm corectitudinea acestei definiţii. Cu notaţiile de mai sus presupunem pentru orice $1 \leq i \leq n$ că $b_i \in A_{s_i}$ şi $\rho(a_i) = \rho(b_i)$. Cu aceste ipoteze este suficient să probăm că $\rho(A_\sigma(a_1, a_2, \dots, a_n)) = \rho(A_\sigma(b_1, b_2, \dots, b_n))$. Pentru orice $1 \leq i \leq n$ din $\rho(a_i) = \rho(b_i)$ deducem $a_i \sim b_i$. Deoarece $\sim$ este congruenţă deducem $A_\sigma(a_1, a_2, \dots, a_n) \sim A_\sigma(b_1, b_2, \dots, b_n)$, deci $\rho(A_\sigma(a_1, a_2, \dots, a_n)) = \rho(A_\sigma(b_1, b_2, \dots, b_n))$.

Teorema 6.5 *Pentru orice morfism de Σ-algebre $f : \mathcal{A} \longrightarrow \mathcal{B}$ dacă $\sim \subseteq Ker(f)$, atunci există un unic morfism de Σ-algebre $f^\# : \mathcal{A}/\sim \longrightarrow \mathcal{B}$ cu proprietatea $\rho; f^\# = f$.*

Demonstraţie: Din proprietatea de universalitate a mulţimii cât deducem existenţa unei unice funcţii $f^\# : A/\sim \longrightarrow B$ cu proprietatea $\rho; f^\# = f$. Observăm că $f^\#_s(\rho(a)) = f_s(a)$ pentru orice $s \in S$ şi $a \in A_s$. Mai trebuie să demonstrăm că funcţia $f^\#$ este un morfism de Σ-algebre.

Fie $s_1 s_2 \dots s_n \in S^*$, $s \in S$, $\sigma \in \Sigma_{s_1 s_2 \dots s_n, s}$ şi $a_i \in A_{s_i}$ pentru orice $1 \leq i \leq n$. Observăm că

$$f^\#_s(A/\sim_\sigma(\rho(a_1), \rho(a_2), \dots, \rho(a_n))) = f^\#_s(\rho(A_\sigma(a_1, a_2, \dots, a_n))) = f_s(A_\sigma(a_1, a_2, \dots, a_n)) =$$

$$B_\sigma(f_{s_1}(a_1), f_{s_2}(a_2), \dots, f_{s_n}(a_n)) = B_\sigma(f^\#_{s_1}(\rho(a_1)), f^\#_{s_2}(\rho(a_2)), \dots, f^\#_{s_n}(\rho(a_n))).$$

Deci $f^\#$ este morfism de Σ-algebre. $\odot$

Să se arate că pentru orice Σ-morfism $h : \mathcal{A} \longrightarrow \mathcal{B}$ algebrele $\mathcal{A}/Ker(h)$ şi $h(\mathcal{A})$ sunt izomorfe.

6.3 Proiectivitatea algebrelor libere

Definiţia 6.6 O Σ-algebră $\mathcal{P}$ se numeşte *proiectivă* dacă pentru orice Σ-morfism cu toate componentele surjective $e : \mathcal{A} \longrightarrow \mathcal{B}$ si pentru orice morfism $f : \mathcal{P} \longrightarrow \mathcal{B}$ există un morfism $g : \mathcal{P} \longrightarrow \mathcal{A}$ astfel încât $g; e = f$.

Propoziţie 6.7 *Orice Σ-algebră liberă este proiectivă.*

Demonstraţie: Fie X o mulţime S-sortată de variabile. Vom proba că Σ-algebra $T_\Sigma(X)$ liber generată de X este proiectivă.

Fie $\mathcal{A}$, $\mathcal{B}$ două Σ-algebre şi $e : \mathcal{A} \longrightarrow \mathcal{B}$ un morfism cu toate componentele surjective. Dacă $f : T_\Sigma(X) \longrightarrow \mathcal{B}$ este un morfism oarecare trebuie să demonstrăm că există un morfism $g : T_\Sigma(X) \longrightarrow \mathcal{A}$ astfel încât $g; e = f$.

Pentru a defini morfismul g este suficient să dăm acţiunea lui pe variabile.

Fie $s \in S$ şi $x \in X_s$. Deoarece e_s este surjectiv, deci există $a_x \in A_s$ astfel încât $f_s(x) = e_s(a_x)$. Definim g ca fiind unicul morfism cu proprietatea că $g_s(x) = a_x$ pentru orice $s \in S$ şi orice $x \in X_s$. Este evident faptul că $(g; e)_s(x) = f_s(x)$ pentru orice $s \in S$ şi orice $x \in X_s$. Deoarece morfismele $g; e$ şi f coincid pe generatorii algebrei libere $T_\Sigma(X)$ rezultă că $g; e = f$ şi demonstraţia este încheiată. $\odot$

Comentariu. Faptul că $g_s(x)$ poate fi ales arbitrar în $e_s^{-1}(\{f_s(x)\})$ nu garantează unicitatea lui g.

7 Semantica logicii ecuaţionale

Privind mulţimea de ecuaţii condiţionate Γ ca o mulţime de axiome, ne interesează toate consecinţele ei semantice. Prin consecinţă semantică se înţelege orice fapt adevărat în orice Γ–algebră. Logica ecuaţională este interesată de consecinţele semantice care sunt scrise ca egalităţi formale.

Prin urmare ecuaţia $(\forall X) l \stackrel{\circ}{=}_s r$ este o consecinţă semantică a lui Γ, denumită şi tautologie sau propoziţie valabilă sau validă a logicii ecuaţionale, dacă ea este adevărată în orice Γ–algebră, adică $h_s(l) = h_s(r)$ pentru orice Γ–algebră $\mathcal{M}$ şi pentru orice morfism $h : T_\Sigma(X) \longrightarrow \mathcal{M}$.

Pentru conceptul de consecinţă semantică folosim notaţia

$$\models (\forall X) l \stackrel{\circ}{=}_s r.$$

Sintetizând observăm că

$$\models (\forall X) l \stackrel{\circ}{=}_s r \text{ dacă şi numai dacă } (\forall h : T_\Sigma(X) \longrightarrow \mathcal{M} \models \Gamma) h_s(l) = h_s(r).$$

Conceptul poate fi generalizat înlocuind algebra liberă $T_\Sigma(X)$ cu o algebră arbitrară $\mathcal{A}$. Prin urmare pentru elementele l şi r de sort s din algebra $\mathcal{A}$ putem scrie

$$\models l \stackrel{\circ}{=}_s r \text{ dacă şi numai dacă } (\forall h : \mathcal{A} \longrightarrow \mathcal{M} \models \Gamma) h_s(l) = h_s(r).$$

Pentru orice algebră $\mathcal{A} = (A_s, A_\sigma)$ notăm cu

$$Sen(\mathcal{A}) = \{l \stackrel{\circ}{=}_s r | s \in S,\ l, r \in A_s\}$$

mulţimea *propoziţilor* sale. Propoziţiile sunt de fapt egalităţi formale care pot fi adevărate sau false.

Pentru orice algebră $\mathcal{A}$, putem identifica $Sen(\mathcal{A})$ cu produsul cartezian $A \times A$. Poate cea mai bună reprezentare a unei propoziţii din $\mathcal{A}$ este un triplet format dintr-un sort s si două elemente de sort s din $\mathcal{A}$.

Având în vedere cele de mai sus nu vom mai face nici o distincţie între relaţiile lui $\mathcal{A}$ şi mulţimile de egalităţi formale între elementele lui $\mathcal{A}$.

7.1 Congruenţa semantică

Vom lucra într-o Σ-algebră $\mathcal{A}$ şi vom grupa într-o relaţie toate tautologiile(propoziţiile valide) logicii ecuaţionale din algebra $\mathcal{A}$.

Fie $\equiv_\Gamma^{\mathcal{A}}$ relaţia pe A definită prin

$$a \equiv_\Gamma^{\mathcal{A}} c \text{ dacă şi numai dacă } (\forall h : \mathcal{A} \longrightarrow \mathcal{M} \models \Gamma) h_s(a) = h_s(c).$$

Dacă nu există pericol de confuzie vom prefera să scriem $\equiv_\Gamma$ în loc de $\equiv_\Gamma^{\mathcal{A}}$.

Observăm că

$$\equiv_\Gamma = \bigcap \{Ker(h) \mid h : \mathcal{A} \longrightarrow \mathcal{M} \models \Gamma\}.$$

Deoarece nucleul unui morfism este o relaţie de congruenţă şi deoarece orice intersecţie de relaţii de congruenţe este o relaţie de congruenţă, deducem că $\equiv_\Gamma$ este o relaţie de congruenţă.

$\equiv_\Gamma$ este numită **congruenţă semantică.**

Definim **regula de deducţie a substituţiei** utilizată atât în logica ecuaţională cât şi în rescrierea termenilor.

Sub$_\Gamma$ Pentru orice $(\forall X)\, l \stackrel{\circ}{=}_s r$ **if** $H \in \Gamma$ şi orice morfism $h :\ T_\Sigma(X) \longrightarrow \mathcal{A}$
$(\forall u \stackrel{\circ}{=}_t v \in H) h_t(u) \stackrel{\circ}{=}_t h_t(v)$ implică $h_s(l) \stackrel{\circ}{=}_s h_s(r)$.

Definiţia 7.1 O submulţime M a lui $Sen(\mathcal{A})$ se numeşte **închisă la substituţie** sau închisă la **Sub**$_\Gamma$ dacă pentru orice $(\forall X) l \stackrel{\circ}{=}_s r$ **if** H în Γ şi orice $h : T_\Sigma(X) \longrightarrow \mathcal{A}$
$(\forall u \stackrel{\circ}{=}_t v \in H) h_t(u) \stackrel{\circ}{=}_t h_t(v) \in M$ implică $h_s(l) \stackrel{\circ}{=}_s h_s(r) \in M$.

Observăm că intersecţia unor mulţimi închise la substituţie este tot o mulţime închisă la substituţie.

Lemă 7.2 *Pentru orice morfism $f : \mathcal{A} \longrightarrow \mathcal{M} \models \Gamma$, $Ker(f)$ este închis la* **Sub**$_\Gamma$.

Demonstraţie: Fie $(\forall X) l \stackrel{\circ}{=}_s r$ **if** H în Γ şi $h : T_\Sigma(X) \longrightarrow \mathcal{A}$ un morfism cu proprietatea $h_t(u)\ Ker(f)\ h_t(v)$ pentru orice $u \stackrel{\circ}{=}_t v \in H$. Prin urmare $(h; f)_t(u) = (h; f)_t(v)$ pentru orice $u \stackrel{\circ}{=} v \in H$. Deoarece $h; f :\ T_\Sigma(X) \longrightarrow \mathcal{M} \models (\forall X) l \stackrel{\circ}{=}_s r$ **if** H rezultă că $(h; f)_s(l) = (h; f)_s(r)$, deci $h_s(l)\ Ker(f)\ h_s(r)$.$\odot$

Propoziţie 7.3 *Congruenţa semantică este închisă la substituţie.*

Demonstraţie: Congruenţa semantică este o intersecţie de congruenţe închise la substituţie. Deoarece o intersecţie de congruenţe închise la substituţii este o congruenţă închisă la substituţii rezultă că $\equiv_\Gamma$ este o congruenţă închisă la substituţie. $\odot$

Propoziţie 7.4 *Dacă $\sim$ este o congruenţă închisă la substituţii, atunci $\mathcal{A}/\sim \models \Gamma$.*

Demonstraţie: Notăm cu $\rho : \mathcal{A} \longrightarrow \mathcal{A}/\sim$ morfismul de factorizare canonic.

Fie $(\forall X) l \stackrel{\circ}{=}_s r$ **if** H în Γ. Fie $h : T_\Sigma(X) \longrightarrow \mathcal{A}/\sim$ un morfism astfel încât $h_t(u) = h_t(v)$ pentru orice $u \stackrel{\circ}{=}_t v \in H$. Deoarece orice algebră liberă este proiectivă rezultă că $T_\Sigma(X)$ este algebră proiectivă, prin urmare există un morfism $f : T_\Sigma(X) \longrightarrow \mathcal{A}$ astfel încât $f; \rho = h$. Pentru orice $u \stackrel{\circ}{=}_t v \in H$ deoarece $\rho_t(f_t(u)) = \rho_t(f_t(v))$ deducem $f_t(u) \sim f_t(v)$.

Deoarece $\sim$ este o congruenţă închisă la substituţii obţinem $f_s(l) \sim f_s(r)$. Prin urmare $\rho_s(f_s(l)) = \rho_s(f_s(r))$, de unde $h_s(l) = h_s(r)$. $\odot$

Propoziţie 7.5 *Congruenţa semantică $\equiv_\Gamma$ este cea mai mică congruenţă închisă la substituţie.*

Demonstraţie: Fie $\sim$ o congruenţă închisă la substituţie. Conform propoziţiei 7.4 $\mathcal{A}/\sim \models \Gamma$

Deoarece

$$\equiv_\Gamma = \bigcap \{Ker(h) \mid h : \mathcal{A} \longrightarrow \mathcal{M} \models \Gamma\}.$$

rezultă incluziunea

$$\equiv_\Gamma \subseteq Ker(h)$$

pentru orice morfism $h : \mathcal{A} \longrightarrow \mathcal{M} \models \Gamma\}$.

Deoarece morfismul de factorizare canonic de la $\mathcal{A}$ la $\mathcal{A}/\sim$ este un astfel de morfism si nucleul lui este $\sim$ deducem $\equiv_\Gamma \subseteq \sim$. $\odot$

7.2 ??

Fie $\mathcal{A}_\Gamma$ factorizarea lui $\mathcal{A}$ prin congruenţa $\equiv_\Gamma$ şi fie $\eta : \mathcal{A} \longrightarrow \mathcal{A}_\Gamma$ morfismul cât.

Teorema 7.6 $\mathcal{A}_\Gamma \models \Gamma$

Demonstraţie: Se aplică propoziţiile 7.3 şi 7.4. $\odot$

Teorema 7.7 *Pentru orice Γ-algebră $\mathcal{B}$ şi pentru orice morfism $h : \mathcal{A} \longrightarrow \mathcal{B}$ există şi este unic un morfism $h^\# : \mathcal{A}_\Gamma \longrightarrow \mathcal{B}$ astfel încât $\eta; h^\# = h$.*

Demonstraţie: Este suficient să arătăm că $a \equiv_\Gamma c$ implică $h(a) = h(c)$ şi să aplicăm proprietatea de universalitate a algebrei cât. Într-adevăr $a \equiv_\Gamma c$ implică $h(a) = h(c)$ deoarece $h : \mathcal{A} \longrightarrow \mathcal{B} \models \Gamma$. $\odot$

Corolar 7.8 *Dacă $\mathcal{A}$ este Σ-algebră iniţială, atunci $\mathcal{A}_\Gamma$ este Γ-algebră iniţială.*

Corolar 7.9 *Pentru orice signatură Σ şi pentru orice mulţime Γ de ecuaţii condiţionate există o Γ-algebră iniţială.*

Demonstraţie: Se foloseşte existenţa Σ–algebrei initiale şi corolarul precedent.

7.3 Problema programării prin rescriere

Principala problemă este : "poate o maşină să demonstreze că $a \equiv_\Gamma c$?".

Se ştie că în unele cazuri rescrierile ne dau o soluţie.

8 TIPURI ABSTRACTE de DATE

Am văzut în lecţiile precedente că orice signatură determină prin algebra sa iniţială un tip abstract de date. Tipul abstract de date al numerelor naturale a fost determinat de signatura formată din constanta 0 şi operaţia unară cunoscută sub numele de succesor.

Acum am pus în evidenţă un instrument mai puternic deoarece orice signatură împreună cu o mulţime Γ de ecuaţii condiţionate determină prin Γ-algebra iniţială, a cărei existenţă am dovedit-o mai sus, un tip abstract de date.

8.1 Tipul abstract al numerelor naturale - continuare

Considerăm signatura cu un singur sort *nat*, o singură constantă de sort nat şi o singură operaţie unară cu argument şi rezultat de sort nat:

sort nat .
op 0 : $\longrightarrow$ nat .
op s : nat $\longrightarrow$ nat .

Elementele algebrei iniţiale sunt

$$0,\ s(0),\ s(s(0)),\ s(s(s(0))),\ s(s(s(s(0)))),\ ...$$

şi ele reprezintă numerele naturale 0 1 2 3 4 ...

Propoziţie 8.1 *Algebra $(N, 0_N, s_N)$ definită prin: N este mulţimea numerelor naturale, 0_N este numărul natural zero şi $s_N(n) = n + 1$ pentru orice număr natural n; este iniţială. (vezi şi propoziţia 3.6)*

Propoziţia anterioară ne arată cum pot fi definite numerele naturale prim metoda algebrei iniţiale ca tip abstract de date.

Deocamdată prin signatura de mai sus calculatorul învaţă numerele naturale dar nu ştie încă să calculeze. Să-l învăţăm deocamdată să adune şi să înmulţească.

Adunarea poate fi introdusă prin declaraţia

$$\text{op } \_+\_ : \text{nat nat} \longrightarrow \text{nat} .$$

dar aceasta, singură, nu face decât să strice ceea ce am construit deja. Deoarece nu ştie să adune calculatorul interpretează de exemplu $0 + 0$ ca un nou element diferit de 0, ceea ce evident nu dorim.

La cele de mai sus adăugăm prea cunoscutele axiome ale lui Peano.

op _+_ : nat nat $\longrightarrow$ nat .
var X Y : nat .
eq X + 0 = 0 .
eq X + s(Y) = s(X+Y) .

Inmulţirea este introdusă prin

op _*_ : nat nat $\longrightarrow$ nat .
eq X * 0 = 0 .
eq X * s(Y) = (X*Y) + X .

Parantezele din ultimul rând sunt puse pentru ca maşina să înţeleagă că înmulţirea se efectuează inaintea adunării. Există metode mai eficiente care fac maşina să înţeleagă acest fapt, dar acesta nu este scopul prezentului text. În demonstraţia de mai jos nu mai punem aceste paranteze deoarece textul nu se mai adresează calculatorului.

Trebuie să menţionăm că egalităţiile de mai sus sunt folosite în două moduri:

a) ca ecuaţii care împreună cu signatura de patru operaţii formează o mulţime Γ pentru a defini o structură algebrică şi

b) ca reguli de rescriere în timpul execuţiei programelor, adică se aplică numai de la stânga la dreapta.

Propoziţie 8.2 *Algebra $\mathcal{N} = (N, 0_N, s_N, +_N, *_N)$ este Γ-algebră iniţială.*

Demonstraţie: Fie $\mathcal{A} = (A, O_A, s_A, +_A, *_A)$ o Γ-algebră. Menţionăm că algebra $\mathcal{A}$ satisfac ecuaţiile din Γ, adică

1) $a +_A 0_A = a$ pentru orice a din A,
2) $a +_A s_A(b) = s_A(a + b)$ pentru orice a, b din A,
3) $a *_A 0_A = 0_A$ pentru orice a din A,
4) $a *_A s_A(b) = a *_A b +_A a$ pentru orice a, b din A.

Vom proba că există un unic morfism de Γ-algebre de la $\mathcal{N}$ la $\mathcal{A}$.

Să începem cu unicitatea. Dacă $h : \mathcal{N} \longrightarrow \mathcal{A}$ este un Γ-morfism, atunci $h : (N, 0_N, s_N) \longrightarrow (A, 0_A, s_A)$ este morfism prin urmare coincide cu unicul morfism dat de propoziţia de mai sus.

Pentru a demonstra existenţa nu avem decât o singură şansă şi anume să dovedim că unicul morfism $h : (N, 0_N, s_N) \longrightarrow (A, 0_A, s_A)$ este şi morfism de Γ-algebre. Reamintim că

$$h(0_N) = 0_A \text{ şi pentru orice } n \text{ număr natural } h(n+1) = s_A(h(n)).$$

Probăm că $h(n +_N m) = h(n) +_A h(m)$ prin inducţie după m

$h(n +_N 0_N) = h(n) = h(n) +_A 0_A = h(n) +_A h(0_N)$ şi
$h(n+_N(m+1)) = h(s_N(n+_N m)) = s_A(h(n+_N m)) = s_A(h(n)+_A h(m)) = h(n)+_A s_A(h(m)) = h(n)+_A h(m+1)$.

Probăm că $h(n *_N m) = h(n) *_A h(m)$ prin inducţie după m

$h(n *_N 0_N) = h(0_N) = 0_A = h(n) *_A 0_A = h(n) *_A h(0_N)$ şi
$h(n *_N (m+1)) = h(n *_N m +_N n) = h(n *_N m) +_A h(n) = (h(n) *_A h(m)) +_A h(n) = h(n) *_A s_A(h(m)) = h(n) *_A h(m+1)$. $\odot$

Propoziţia anterioară demonstrează corectitudinea definiţiei de mai sus. Deoarece algebra $\mathcal{N}$ este iniţială rezultă că ea este izomorfă cu Γ-algebra iniţială, deci specificaţia de mai sus caracterizează, prin Γ-algebra sa iniţială, tipul de date al numerelor naturale.

Cred că este util să menţionăm aici că dat un tip de date nu este suficient să găsim o structură algebrică care să-l definească ca tip abstract de date. De exemplu ştim din algebră că numerele naturale formează un semiinel iniţial. Caracterizarea numerelor naturale ca semiinel iniţial nu ne mulţumeşte deoarece axiomele conceptului de semiinel nu ne dau un program(calculatorul este incapabil să facă operaţii cu numerele naturale folosind numai axiomele semiinelului). Axiomele lui Peano dau un program în programarea prin rescriere.

9 TEORII DEDUCTIVE À la MOISIL

Fie E o mulţime de propoziţii. O *regulă de deducţie* pe E este o pereche (H, e) unde H este o submulţime *finită* a lui E (şi se numeşte ipoteza regulii) şi $e \in E$ (şi se numeşte concluzia regulii). Poate mai sugestiv o regulă (H, e) ar fi trebuit scrisă $H \longrightarrow e$ deoarece semnificaţia ei este "propoziţia e este o consecinţa a propoziţiilor din H". Mai menţionăm că în cazul particular $H = \emptyset$ regula $(\emptyset, e)$ spune că e este o axiomă.

O submulţime D a lui E se numeşte **închisă** la regula (H, e) dacă $H \subseteq D$ implică $e \in D$. In acest caz se mai spune că regula (H, e) este corectă pentru D deoarece regula nu ne scoate din D.

Fie R o mulţime de reguli de deducţie. O submulţime D a lui E se numeşte inchisă la R dacă şi numai dacă D este inchisă la orice regulă din R.

Definiţia închiderii la substituţie, dată în secţiunea 7.1, este un caz particular al definiţiei de mai sus.

Observaţia 9.1 *Dacă $D_i \subseteq E$ este inchisă la R pentru fiecare $i \in I$, atunci $\bigcap_{i\in I} D_i$ este închisă la R.*

Demonstraţie: Fie $(H, e) \in R$ astfel încât $H \subseteq \bigcap_{i\in I} D_i$. Pentru orice $i \in I$, deoarece D_i este inchisă la R şi $H \subset D_i$ deducem $e \in D_i$. Deci $e \in \bigcap_{i\in I} D_i$. $\odot$

Prin definiţie $[n] = \{1, 2, \ldots, n\}$ pentru orice număr natural n.

O propoziţie $e \in E$ poate fi demonstrată folosind R dacă şi numai dacă există o secvenţă finită de propoziţii $e_1e_2 \ldots e_n$ astfel încât $e_n = e$ şi pentru orice $i \in [n]$, există $(H, e_i) \in R$ astfel încât $H \subseteq \{e_1, e_2, \ldots, e_{i-1}\}$. Vom spune că $e_1e_2 \ldots e_n$ este o *demonstraţie* pentru $e_n = e$.

Observaţia 9.2 *Dacă α şi β sunt demonstraţii atunci $\alpha\beta$ este demonstraţie.*

Notăm cu $\mathbf{Teor}(R)$ mulţimea propoziţiilor care pot fi demonstrate folosind R.

Lemă 9.3 *Fie $D \subseteq E$. Dacă D este inchisă la R atunci $\mathbf{Teor}(R) \subseteq D$.*

Demonstraţie: Fie $e \in \mathbf{Teor}(R)$. Prin urmare există o secvenţă finită de propoziţii $e_1e_2 \ldots e_n$ astfel încât $e_n = e$ şi pentru oricare i din [n] există $(H, e_i) \in R$ astfel încât $H \subseteq \{e_1, e_2, \ldots, e_{i-1}\}$.

Probăm prin inducţie că $e_i \in D$ pentru orice i din [n]. Ipoteza de inducţie este $(\forall k < i)\ e_k \in D$. Deoarece şirul $e_1e_2 \ldots e_n$ este o demonstraţie există $(H, e_i) \in R$ cu $H \subseteq \{e_1, e_2, \ldots, e_{i-1}\}$. Din ipoteza de inducţie deducem $H \subseteq D$. Deoarece D este închisă la R rezultă că $e_i \in D$.

În particular pentru $i = n$ deducem $e \in D$.

Dar e a fost luată arbitrar în $\mathbf{Teor}(R)$, deci $\mathbf{Teor}(R) \subseteq D$. $\odot$

Această lemă ne furnizează o metodă prin care putem arăta că toate propoziţiile demonstrabile folosind R au o anumită proprietate şi anume este suficient să arătăm că mulţimea propoziţiilor cu proprietatea dată este închisă la R.

Teorema 9.4 $\mathbf{Teor}(R)$ *este cea mai mică mulţime(în raport cu $\subseteq$) care este închisă la R.*

Demonstraţie: Demonstrăm că $\mathbf{Teor}(R)$ este închisă la R. În acest scop presupunem $(H, e) \in R$, unde $H = \{f_1, \ldots, f_n\}$ şi $H \subseteq \mathbf{Teor}(R)$. Pentru orice $i \in [n]$ din $f_i \in \mathbf{Teor}(R)$ deducem existenţa unei demonstraţii $\alpha_i f_i$ unde α_i este un şir de propoziţii din E. Atunci $\alpha_1 f_1 \ldots \alpha_n f_n$ este o demonstraţie deoarece este o concatenare de demonstraţii şi pentru că $(\{f_1, \ldots, f_n\}, e) \in R$ deducem că $\alpha_1 f_1 \ldots \alpha_n f_n e$ este o demonstraţie pentru e. Deci $e \in \mathbf{Teor}(R)$. Rezultă că $\mathbf{Teor}(R)$ este închisă la R. Din lema 9.3 rezultă că $\mathbf{Teor}(R)$ este cea mai mică în sensul incluziunii care este închisă la R. $\odot$

10 LOGICĂ ECUAŢIONALA

Fie Γ o mulţime de ecuaţii condiţionale şi o Σ-algebra $\mathcal{A}$ fixată.

Mulţimea propoziţiilor adevărate, tautologiile, din $\mathcal{A}$ este chiar congruenţa semantică

$$\equiv_\Gamma = \{a \stackrel{\circ}{=}_s b \in Sen(A) : \ (\forall M \models \Gamma)(\forall f : \ A \longrightarrow M) f_s(a) = f_s(b)\}.$$

Conform tradiţiei mai scriem

$$\models a \stackrel{\circ}{=}_s b \text{ dacă şi numai dacă } a \equiv_\Gamma b$$

Se caută o mulţime corectă şi completă de reguli de deducţie pentru $\equiv_\Gamma^{\mathcal{A}}$.

10.1 Necesitatea utilizării cuantificatorilor în ecuaţii

Vom ilustra printr-un exemplu necesitatea utilizării cuantificatorilor în ecuaţiile logicii ecuaţionale multisortate.

Fie signatura $S = \{a, bool\}$ şi $\Sigma = \{g, F, T\}$. Rangurile simbolurilor de operaţii sunt date prin desenul următor :

$$a \xrightarrow{\ g\ } bool \quad \begin{array}{c} \xleftarrow{\ F\ } \\ \xleftarrow{\ T\ } \end{array}$$

unde se vede că sortul rezultat al celor trei simboluri de operaţie g, F şi T este *bool*; operaţiile F şi T nu au argumente şi g are un singur argument de sort a.

Vom lucra cu două Σ–algebre.

Σ–algebra iniţială este $\mathcal{I} = (\emptyset, \{F, T\}; I_g, I_F, I_T)$ unde $F \neq T$; $I_F = F$, $I_T = T$ şi $I_g : \emptyset \longrightarrow \{F, T\}$ este funcţia incluziune.

Pentru orice variabilă x de sort a, Σ-algebra liber generată de această variabilă $T_\Sigma(\{x\}, \emptyset)$ are suporturile $\{x\}$ pentru sortul a şi {g(x),F,T} pentru sortul *bool*.

Are loc relaţia $\mathcal{I} \models_\Sigma F \stackrel{\circ}{=} T$? Sau mai intuitiv: este egalitatea formală $F \stackrel{\circ}{=} T$ adevărată în algebra $\mathcal{I}$? Vom arăta că raspunsul depinde de algebra liberă în care este scrisă egalitatea.

Sunt posibile cel puţin două variante.

1. $\mathcal{I} \models_\Sigma (\forall \emptyset) T \stackrel{\circ}{=} F$ dacă şi numai dacă pentru orice morfism $h : \mathcal{I} \longrightarrow \mathcal{I}$, $h_{bool}(F) = h_{bool}(T)$, ceea ce este **fals** deoarece $h_{bool}(F) = F \neq T = h_{bool}(T)$.

2. $\mathcal{I} \models_\Sigma (\forall x) T \stackrel{\circ}{=} F$ dacă şi numai dacă pentru orice morfism $h : T_\Sigma(\{x\}, \emptyset) \longrightarrow \mathcal{I}$, $h_{bool}(F) = h_{bool}(T)$, ceea ce este **adevărat** deoarece nu există nici un morfism $h : T_\Sigma(\{x\}, \emptyset) \longrightarrow \mathcal{I}$.

Am arătat că $\mathcal{I} \models_\Sigma (\forall \emptyset) T \stackrel{\circ}{=} F$ este falsă şi că $\mathcal{I} \models_\Sigma (\forall x) T \stackrel{\circ}{=} F$ este adevărată. Dacă omitem cuantificatorii obţinem: "$\mathcal{I} \models_\Sigma T \stackrel{\circ}{=} F$ este falsă şi $\mathcal{I} \models_\Sigma T \stackrel{\circ}{=} F$ este adevărată."

Contradicţia obţinută prin omiterea cuantificatorilor din faţa egalităţii formale $F \stackrel{\circ}{=} T$ arată că în logica ecuaţională multisortată prezenţa cuantificatorilor în ecuaţii este necesară.

In multe cazuri, de exemplu în limbajele de programare PROLOG, OBJ, Maude, CafeOBJ, etc cuantificatorii din faţa ecuaţiilor sunt omişi dar se presupune implicit că ecuaţiile sunt cuantificate prin mulţimea variabilelor care apar în ecuaţie. Egalitatea formală $T \stackrel{\circ}{=} F$ este implicit interpretată $(\forall \emptyset) T \stackrel{\circ}{=} F$.

10.2 Punctul de vedere local

Fixăm o algebră $\mathcal{A}$ şi lucrăm cu propoziţii din $Sen(\mathcal{A})$. Acesta este punctul *local* de vedere al logicii ecuaţionale. Cazul în care ne interesează propoziţii din algebre diferite este numit *global* dar va fi puţin folosit în acestă carte. În cazul global o propoziţie din $\mathcal{D}$ va fi scrisă $(\forall \mathcal{D}) a \stackrel{\circ}{=}_s c$ în loc de $a \stackrel{\circ}{=}_s c$. Notaţia fără cuantificatori folosită în cazul local nu contrazice ceea ce am scris despre necesitatea utilizării cuantificatorilor în ecuaţii. Pur si simplu nu scriem cuantificatorul pentru că fixând algebra $\mathcal{A}$ el va fi mereu acelaşi $(\forall \mathcal{A})$ şi deci îl ştim chiar dacă nu-l vedem scris.

O altă idee pe care dorim să o subliniem este folosirea unei algebre arbitrare $\mathcal{A}$ în locul unei algebre libere. Principalele rezultate privind rescrierile, care se referă atât la rescrierile de termeni cât şi la rescrierile modulo ecuaţii, rămân valabile în acest cadru mai general. Aceasta prezentare unitară a diverselor tipuri de rescriere este de fapt contribuţia autorului la modernizarea lecţiilor despre rescriere. Există în lume trei cărţi privind rescrieriile [**?**, **?**, **?**]. Nici una nu prezintă stilul local.

Menţionăm că algebra $\mathcal{A}$ nu are legătură cu algebra $T_\Sigma(X)$ folosită în vreo axiomă $(\forall X) l \stackrel{\circ}{=}_s r$ **if** H din Γ. Menţionăm şi faptul că în axiome diferite putem folosi algebre libere diferite. Aceasta corespunde cazului practic, de exemplu scriem comutativitatea $xy = yx$ într-o algebră liberă cu doi generatori x şi y iar asociativitatea $(xy)z = x(yz)$ într-o algebră liberă cu trei generatori x, y şi z.

10.3 Reguli de deducţie, corectitudine

Notăm cu $\mathbf{R_E}$ mulţimea următoarelor reguli de deducţie pentru logica ecuaţională multisortată:

R	$a \stackrel{\circ}{=}_s a$
S	$a \stackrel{\circ}{=}_s b$ implică $b \stackrel{\circ}{=}_s a$
T	$a \stackrel{\circ}{=}_s b$ şi $b \stackrel{\circ}{=}_s c$ implică $a \stackrel{\circ}{=}_s c$
CΣ	Pentru orice $\sigma \in \Sigma_{s_1 s_2 \ldots s_n, s}$:
	$a_i \stackrel{\circ}{=}_{s_i} b_i$ pentru $1 \leq i \leq n$ implică $A_\sigma(a_1, a_2, \ldots, a_n) \stackrel{\circ}{=}_s A_\sigma(b_1, b_2, \ldots, b_n)$.
Sub$_\Gamma$	Pentru orice $(\forall X)\, l \stackrel{\circ}{=}_s r$ **if** $H \in \Gamma$ şi pentru orice $h : T_\Sigma(X) \longrightarrow \mathcal{A}$
	$h_t(u) \stackrel{\circ}{=}_t h_t(v)$ pentru orice $u \stackrel{\circ}{=}_t v \in H$ implică $h_s(l) \stackrel{\circ}{=}_s h_s(r)$

Conform tradiţiei notăm prin $\vdash a \stackrel{\circ}{=}_s b$ faptul că egalitatea formală $a \stackrel{\circ}{=}_s b$ este demonstrabilă cu regulile de mai sus.

Observaţia 10.1 *Observăm că o mulţime de egalităţi formale este închisă la regula* **R** *dacă şi numai dacă este o relaţie reflexivă.*

Din această cauză **R** este numită regula reflexivităţii.

Observaţia 10.2 *Observăm că o mulţime de egalităţi formale este închisă la regula* **S** *dacă şi numai dacă este o relaţie simetrică.*

Din această cauză **S** este numită regula simetriei.

Observaţia 10.3 *Observăm că o mulţime de egalităţi formale este închisă la regula* **T** *dacă şi numai dacă este o relaţie tranzitivă.*

Din această cauză **T** este numită regula tranzitivităţii.

Observaţia 10.4 *Observăm că o mulţime de egalităţi formale este închisă la regula* **C**Σ *dacă şi numai dacă este o relaţie compatibilă cu toate operaţiile algebrei* $\mathcal{A}$.

Din această cauză **C**Σ este numită regula compatibilităţii cu operaţiile din Σ.

Teorema 10.5 *Regulile de deducţie* $\mathbf{R_E}$ *sunt corecte pentru* $\equiv_\Gamma$.

Demonstraţie: Deoarece $\equiv_\Gamma$ este congruenţă rezultă ea este închisă la primele patru reguli, prin urmare ele sunt corecte pentru $\equiv_\Gamma$. Corectitudinea ultimei reguli rezultă din închiderea congruenţei semantice la substituţii în conformitate cu propoziţia 7.3. $\odot$

Corolar 10.6 $\vdash a \stackrel{\circ}{=}_s b$ *implică* $\models a \stackrel{\circ}{=}_s b$. $\odot$

Demonstraţie: Congruenţa semantică, adică mulţimea egalităţilor formale cu proprietatea $\models a \stackrel{\circ}{=}_s b$, este închisă la $\mathbf{R_E}$ conform teoremei 10.5

Mulţimea egalităţilor formale demonstrabile, adică mulţimea egalităţilor formale cu proprietatea $\vdash a \stackrel{\circ}{=}_s b$, este cea mai mică mulţime închisă la $\mathbf{R_E}$ conform teoremei 9.4.

Prin urmare mulţimea egalităţilor formale cu proprietatea $\vdash a \stackrel{\circ}{=}_s b$ este inclusă în mulţimea egalităţilor formale cu proprietatea $\models a \stackrel{\circ}{=}_s b$, de unde rezultă concluzia.

10.4 Completitudine

Pentru orice $s \in S$ şi $a, b \in A_s$ se defineşte relatia $\sim_\Gamma$ in $\mathcal{A}$ prin:

$$a \sim_\Gamma b \Longleftrightarrow \vdash a \stackrel{\circ}{=}_s b.$$

Deoarece regulile **R, S** şi **T** sunt în $\mathbf{R_E}$ deducem că $\sim_\Gamma$ este închisă la **R, S** şi **T** deci este o echivalenţă. Deoarece **C**Σ este în $\mathbf{R_E}$ rezultă că $\sim_\Gamma$ este închisă la **C**Σ deci este congruenţă. Mai mult, deoarece **Sub**$_\Gamma$ este în $\mathbf{R_E}$ rezultă că $\sim_\Gamma$ este închisă la substituţii.

Fie $\mathcal{A}_\Gamma$ câtul lui $\mathcal{A}$ prin $\sim_\Gamma$.

Observaţia 10.7 $\mathcal{A}_\Gamma \models \Gamma$

Demonstraţie: Deoarece $\sim_\Gamma$ este închisă la substituţii, aplicând propoziţia 7.4 rezultă concluzia.

Teorema 10.8 *Teoremă de completitudine:* $\models a \stackrel{\circ}{=}_s b$ *implică* $\vdash a \stackrel{\circ}{=}_s b$.

Demonstraţie: Deoarece congruenţa $\sim_\Gamma$ este închisă la substituţii şi conform propoziţiei 7.5 congruenţa semantică este cea mai mică congruenţă închisă la substituţii rezultă incluziunea

$$\equiv_\Gamma \subseteq \sim_\Gamma .$$

Prin urmare $\models a \stackrel{\circ}{=}_s b$ implică $\vdash a \stackrel{\circ}{=}_s b$. $\odot$

În concluzie congruenţa $\sim_\Gamma$ şi congruenţa semantică $\equiv_\Gamma$ coincid. Prin urmare algebra $\mathcal{A}_\Gamma$ coincide cu cea introdusă într-un mod numai aparent diferit în secţiunea **??**, teoremele 7.6 şi 7.7. Proprietatea de universalitate a algebrei $\mathcal{A}_\Gamma$ demonstrată atunci, teorema 7.7, rămâne adevărată şi în noul context.

Următorul pas după o teoremă de completitudine care ne asigură existenţa unei demonstraţii pentru orice tautologie este de a găsi aceste demonstraţii. Informaticienii sunt ceva mai pretenţioşi deoarece doresc ca aceste demonstraţii să fie găsite de un calculator. Pentru acest scop rescrierea este foarte utilă.

11 RESCRIERE LOCALĂ

Rescrierile sunt un fapt pe care-l întâlnim din primii ani de şcoală. De exemplu în şirul de egalităţi

$$2 * (3 + 4) = 2 * 7 = 14$$

facem două rescrieri(înlocuiri): 3+4 esre rescris în 7 şi apoi $2 * 7$ este rescris în 14. Aceste rescrieri sunt permise de regulile adunării şi înmulţirii. Partea expresiei care rămâne neschimbată deoarece rescrierea are loc în interiorul ei se numeşte context. La prima rescriere contextul este $2 * \bullet$, unde $\bullet$ este un semn special care arată locul în care se face rescrierea. Deoarece a doua rescriere se face la vârf contextul este $\bullet$.

De obicei rescrierile se fac de la expresii mai complicate spre expresii mai simple. Rescrierea lui 3+4 în 7 este ceva firesc, pe când rescrierea lui 7 în 3+4 este ceva artificial. Dece 3+4 şi nu 2+5? Prin urmare spre deosebire de egalitate care este simetrică, **rescrierea nu este simetrică**.

Deoarece se doreşte ca rescrierile să fie făcute de calculator, practica programării ne dă un argument foarte puternic împotriva simetriei. Simetria este o regula care conduce la neterminarea programelor, deoarece după ce am rescris a în b putem rescrie pe b în a, pe a în b şi aşa mai departe.

Eliminarea simetriei dintre regulile de deducţie este principala diferenţă dintre rescriere şi calculul cu egalităţi eflectat de logica ecuaţională. Deoarece eliminarea simetriei duce la pierderea completitudinii, simetria va trebui înlocuită cu altceva pentru a reobţine completitudinea.

11.1 Preliminarii

Signatura Σ, mulţimea Γ de axiome şi algebra $\mathcal{A}$ în care se fac rescrierile(localizare) sunt fixate.

Fie X o mulţime S–sortată de variabile şi $T_\Sigma(X)$ Σ–algebra liber generată de X. Pentru orice $x \in X$ şi $\alpha \in T_\Sigma(X)$, vom nota cu

$$nr_x(\alpha) \text{ numărul de apariţii ale lui } x \text{ în } \alpha.$$

Observăm că $nr_x(x) = 1$, $nr_x(y) = 0$ pentru orice $y \in X - \{x\}$ şi
$nr_x(\sigma(e_1, e_2, \ldots, e_k)) = nr_x(e_1) + nr_x(e_2) + \ldots nr_x(e_k)$.

Fie $\mathcal{A}$ o Σ–algebră, $\bullet$ o variabilă de sort s, $\bullet \notin A_s$. Considerăm algebra liber generată de $A \cup \{\bullet\}$, şi anume $T_\Sigma(A \cup \{\bullet\})$. Vom presupune fără a micşora generalitatea că

$$T_\Sigma(A) \subseteq T_\Sigma(A \cup \{\bullet\}).$$

Un element c din $T_\Sigma(A \cup \{\bullet\})$ se numeşte *context* dacă $nr_\bullet(c) = 1$. Dacă $c = \sigma(c_1, c_2, \ldots, c_n)$ este un context, atunci există un $1 \leq i \leq n$ astfel încât c_i este context şi $c_j \in T_\Sigma(A)$ pentru orice $j \neq i$.

Pentru $d \in A_s$, vom nota cu $\bullet \leftarrow d : T_\Sigma(A \cup \{\bullet\}) \longrightarrow \mathcal{A}$ unicul morfism de Σ–algebre definit prin $(\bullet \leftarrow d)(\bullet) = d$ şi $(\bullet \leftarrow d)(a) = a$ pentru orice $a \in A$. Menţionăm că restricţia lui $\bullet \leftarrow d$ la $T_\Sigma(A)$ este un morfism care evaluează în $\mathcal{A}$ o expresie în care elementele din A erau considerate variabile.

Pentru orice t din $T_\Sigma(A \cup \{\bullet\})$ şi $a \in A_s$, vom prefera să scriem

$$t[a]$$

în loc de $(\bullet \leftarrow a)(t)$, deoarece $(\bullet \leftarrow a)(t)$ este de fapt rezultatul înlocuirii lui $\bullet$ prin a în t şi al evalării în $\mathcal{A}$ al expresiei obţinute în $T_\Sigma(A)$.

Dacă în t nu apare $\bullet$, adică t este din Σ-algebra liber generată de A, atunci $t[a] = t[d]$ pentru orice $a, d \in A_s$, deoarece ambele sunt egale cu evaluarea în $\mathcal{A}$ a lui t.

Compunerea relaţiilor $R \subseteq A \times B$ şi $Q \subseteq B \times C$ este relaţia $R; Q \subseteq A \times C$ definită prin

$$R; Q = \{(a, c) \mid (\exists b \in B)(a, b) \in R \text{ şi } (b, c) \in Q\}.$$

Compunerea relaţiilor este asociativă.

Relaţiile între elementele aceleeaşi mulţimi A, adică părţile lui $A \times A$, formează cu operaţia de compunere un monoid al cărui element neutru este relaţia de egalitate. Pentru astfel de relaţii sunt definite puterile lor naturale. Prin definiţie R^0 este chiar relaţia de egalitate. Prin definiţie

$$R^* = \bigcup_{n \geq 0} R^n$$

este închiderea reflexivă şi tranzitivă a relaţiei R, adică R^* este cea mai mică relaţie reflexivă şi tranzitivă care include R. Menţionăm că * este operator de închidere în mulţimea părţilor lui $A \times A$.

Pentru o relaţie notată $\longrightarrow$ în loc de notaţia $\longrightarrow^*$ se preferă notaţia $\xrightarrow{*}$.

11.2 Închiderea la contexte

Definiţia 11.1 O relaţie Q pe A se numeţe **închisă la contexte** dacă pentru orice context c şi pentru orice pereche de elemente a, d din A_s, $a\, Q\, d$ implică $c[a]\, Q\, c[d]$.

Observăm că relaţia de egalitate este închisă la contexte.

Introducem regula de deducţie

CAΣ $a \overset{\circ}{=}_{s_i} d$ implică $A_\sigma(a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n) \overset{\circ}{=}_s A_\sigma(a_1, \ldots, a_{i-1}, d, a_{i+1}, \ldots, a_n)$
$(\forall \sigma \in \Sigma_{s_1 \ldots s_n, s})$, orice $1 \leq i \leq n$, unde $a_j \in A_{s_j}$ pentru orice $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$ şi $a, d \in A_{s_i}$.

Definiţia 11.2 O relaţie $\rho \subset A \times A$ se numeşte **compatibilă pe argumente cu operaţiile** algebrei $\mathcal{A}$ dacă este închisă la **CA**Σ.

Propoziţie 11.3 *O relaţie compatibilă pe argumente cu operaţiile algebrei este închisă la contexte.*

Demonstraţie:

Se arată prin inducţie structurală în $T_\Sigma(A \cup \{\bullet\})$.

Pasul 0: $c = \bullet$. Pentru orice $(a, d) \in Q$, $c[a] = a$, $c[d] = d$, deci $(c[a], c[d]) \in Q$.

Pentru un context $c = \sigma(a_1, \ldots, a_{i-1}, c', a_{i+1}, \ldots, a_n)$ unde $c' \in T_\Sigma(A \cup \{\bullet\})$ este un context şi $a_i \in T_\Sigma(A)$

$$c[a] = (\bullet \leftarrow a)(c) = A_\sigma(a_1[a], \ldots, a_{i-1}[a], c'[a], a_{i+1}[a], \ldots, a_n[a]).$$

La fel $c[d] = A_\sigma(a_1[d], \ldots, a_{i-1}[d], c'[d], a_{i+1}[d], \ldots, a_n[d])$. Mai observăm că $a_j[a] = a_j[d]$ pentru orice $j \neq i$.
Din ipoteza de inducţie $c'[a]\, Q\, c'[d]$ şi ţinând cont de compatibilitatea pe argumente obţinem $c[a]\, Q\, c[d]$.

11.3 Închiderea la preordini compatibile cu operaţiile

Preordine semnifică o relaţie reflexivă şi tranzitivă.

Reamintim regulile de deducţie în mulţimea egalităţilor formale între elementele algebrei $\mathcal{A}$ denumite **R**eflexivitate, **T**ranzitivitate şi **C**ompatibilitate cu operaţiile din Σ:

R $a \overset{\circ}{=}_s a$

T $a \overset{\circ}{=}_s d$ şi $d \overset{\circ}{=}_s c$ implică $a \overset{\circ}{=}_s c$

CΣ $a_i \overset{\circ}{=}_{s_i} c_i$ pentru orice $i \in [n]$ implică
$A_\sigma(a_1, a_2, \ldots, a_n) \overset{\circ}{=}_s A_\sigma(c_1, c_2, \ldots, c_n)$ pentru orice $\sigma \in \Sigma_{s_1 s_2 \ldots s_n, s}$

Remarcăm că $\mathbf{C}\Sigma$ şi $\mathbf{R}$ implică $\mathbf{CA}\Sigma$.
Reciproca $\mathbf{CA}\Sigma$, $\mathbf{R}$ şi $\mathbf{T}$ implică $\mathbf{C}\Sigma$ este demonstrată de lema următoare.

Lemă 11.4 *Fie $\rho \subset A \times A$ o relaţie tranzitivă şi reflexivă în algebra $\mathcal{A}$. Dacă ρ este compatibilă pe argumente cu operaţiile algebrei $\mathcal{A}$, atunci ρ e compatibilă cu operaţiile, adică închisă la $\mathbf{C}\Sigma$.*

Demonstraţie: Fie $\sigma \in \Sigma_{s_1 \dots s_n, s}$ şi $a_i, b_i \in A_{s_i}$ astfel încât $a_i \rho_{s_i} b_i$ pentru orice $i \in [n]$.
Arătăm că $A_\sigma(a_1, \dots, a_n)\, \rho_s\, A_\sigma(b_1, \dots, b_n)$.
Dacă $n = 0$ din reflexivitate deducem $A_\sigma\, \rho_s\, A_\sigma$.
Dacă $n = 1$ din ipoteza $\mathbf{CA}\Sigma$ rezultă $A_\sigma(a_1)\, \rho_s\, A_\sigma(b_1)$.
Dacă $n \geq 2$ aplicând succesiv ipoteza $\mathbf{CA}\Sigma$ obţinem

$$A_\sigma(a_1, a_2, \dots, a_n)\, \rho_s\, A_\sigma(b_1, a_2, \dots, a_n)\, \rho_s\, A_\sigma(b_1, b_2, a_3, \dots, a_n)\, \rho_s \dots \rho_s\, A_\sigma(b_1, b_2, \dots, b_n)$$

Din tranzitivitatea relaţiei ρ deducem $A_\sigma(a_1, a_2, \dots, a_n)\, \rho_s\, A_\sigma(b_1, b_2, \dots, b_n)$. $\odot$

11.4 Γ–rescriere

Un prim pas pentru suplinirea simetriei este înlocuirea regulii substituţiei cu regula **rescrierii**. Vom mai folosi şi regula **rescrierii în subtermeni**.

$\mathbf{Rew}_\Gamma$ Pentru orice $(\forall X)\, l \overset{\circ}{=}_s r$ **if** $H \in \Gamma$ şi orice morfism $h : T_\Sigma(X) \longrightarrow \mathcal{A}$
$(\forall u \overset{\circ}{=}_t v \in H)(\exists d \in A_t) h_t(u) \overset{\circ}{=}_t d$ şi $h_t(v) \overset{\circ}{=}_t d$ implică $h_s(l) \overset{\circ}{=}_s h_s(r)$.

$\mathbf{SRew}_\Gamma$ Pentru orice $(\forall X)\, l \overset{\circ}{=}_s r$ **if** $H \in \Gamma$ şi orice morfism $h : T_\Sigma(X) \longrightarrow \mathcal{A}$
$(\forall u \overset{\circ}{=}_t v \in H)(\exists d \in A_t) h_t(u) \overset{\circ}{=}_t d$ şi $h_t(v) \overset{\circ}{=}_t d$ implică
$c[h_s(l)] \overset{\circ}{=}_{s'} c[h_s(r)]$ pentru orice context $c \in T_\Sigma(A \cup \{\bullet\})_{s'}$.

Remarcaţi că $\mathbf{SRew}_\Gamma$, *rescrierea într-un subtermen*, este o regulă de deducţie mai puternică decât $\mathbf{Rew}_\Gamma$, *rescrierea*, care poate fi obţinută din $\mathbf{SRew}_\Gamma$ pentru $c = \bullet$.
Deasemenea $\mathbf{Rew}_\Gamma$ şi regula contextului implică $\mathbf{SRew}_\Gamma$.

Comparând substituţia şi rescrierea observăm următoarele.
1) $\mathbf{Rew}_\Gamma$ şi $\mathbf{R}$ implică $\mathbf{Sub}_\Gamma$.
2) $\mathbf{Sub}_\Gamma$, $\mathbf{S}$ şi $\mathbf{T}$ implică $\mathbf{Rew}_\Gamma$.

Probăm ultima afirmaţie. Fie $(\forall X)\, l \overset{\circ}{=}_s r$ **if** $H \in \Gamma$ şi un morfism $h : T_\Sigma(X) \longrightarrow \mathcal{A}$ cu proprietatea că pentru orice $u \overset{\circ}{=}_{s'} v \in H$ există $d \in A_{s'}$ cu $h_{s'}(u) \overset{\circ}{=}_{s'} d$ şi $h_{s'}(v) \overset{\circ}{=}_{s'} d$. Pentru orice $u \overset{\circ}{=}_{s'} v \in H$ cu $\mathbf{S}$ deducem $d \overset{\circ}{=}_{s,} h_{s'}(v)$ şi apoi cu $\mathbf{T}$ obţinem $h_{s'}(u) \overset{\circ}{=}_{s'} h_{s'}(v)$. In final aplicăm $\mathbf{Sub}_\Gamma$ şi deducem $h_s(l) \overset{\circ}{=}_s h_s(r)$.

Pentru a rezuma desenăm schema

$$\mathbf{Sub}_\Gamma \quad \underset{\mathbf{R}}{\overset{\mathbf{S,T}}{\rightleftarrows}} \quad \mathbf{Rew}_\Gamma \quad \underset{\mathbf{CA}\Sigma}{\overset{}{\rightleftarrows}} \quad \mathbf{SRew}_\Gamma$$

Definiţia 11.5 Cea mai mică relaţie închisă la $\mathbf{R}$, $\mathbf{T}$, $\mathbf{C}\Sigma$ şi $\mathbf{Rew}_\Gamma$ este denumită Γ–**rescriere** sau mai scurt **rescriere** şi este notată cu $\overset{*}{\Longrightarrow}_\Gamma$.

12 RELAŢIA DE ÎNTÂLNIRE, FORME NORMALE

12.1 Confluenţă

Definiţia 12.1 O relaţie $\succ$ pe o mulţime A se numeşte **confluentă** dacă

$$(\forall a, b, c \in A)\{[a \succ b \text{ şi } a \succ c] \Rightarrow (\exists d \in A)[b \succ d \text{ şi } c \succ d]\}.\ \odot$$

Observăm că orice relaţie simetrică este confluentă.

Propoziţie 12.2 *Dacă $\succ$ este o preordine confluentă pe mulţimea A, atunci relaţia $\downarrow$ definită prin*

$$a \downarrow b \Longleftrightarrow (\exists c \in A) a \succ c \text{ şi } b \succ c$$

şi denumită "întâlnire prin $\succ$" este cea mai mică echivalenţă pe A care include $\succ$.

Demonstraţie:

- Deoarece din $a \succ a$ şi $a \succ a$ deducem $a \downarrow a$. Deci relaţia $\downarrow$ ese reflexivă.
- Simetria este evidentă.
- Probăm tranzitivitatea. Fie $a \downarrow b$ şi $b \downarrow c$. Observăm că există $d, e \in A$ astfel încât $a \succ d$, $b \succ d$, $b \succ e$ şi $c \succ e$. Din confluenţă rezultă existenţa lui $f \in A$ astfel încât $d \succ f$ şi $e \succ f$. Rezultă prin tranzitivitate că $a \succ f$ şi $c \succ f$, deci $a \downarrow c$.
- Pentru a dovedi că $\succ \subseteq \downarrow$, presupunând că $a \succ b$ şi observând că $b \succ b$ deducem $a \downarrow b$
- Fie $\equiv$ o relaţie de echivalenţă pe A care include $\succ$. Probăm că $\equiv$ include $\downarrow$. Presupunând că $a \downarrow b$ rezultă existenţa lui $c \in A$ astfel încât $a \succ c$ şi $b \succ c$. Deducem $a \equiv c$ şi $b \equiv c$, deci $a \equiv b$. $\odot$

Observaţia 12.3 *Dacă $\succ$ este o preordine confluentă şi compatibilă pe o Σ-algebră multisortată, atunci $\downarrow$ este o congruenţă.*

Demonstraţie: Fie $\sigma \in \Sigma_{s_1 s_2 \ldots s_n, s}$ şi $a_i \downarrow b_i$ pentru orice $i \in [n]$. Pentru orice $i \in [n]$, există c_i astfel încât $a_i \succ c_i$ şi $b_i \succ c_i$, prin urmare cu $\mathbf{C}\Sigma$ deducem

$$A_\sigma(a_1, a_2, \ldots, a_n) \succ A_\sigma(c_1, c_2, \ldots, c_n) \text{ şi } A_\sigma(b_1, b_2, \ldots, b_n) \succ A_\sigma(c_1, c_2, \ldots, c_n).$$

Deci $A_\sigma(a_1, a_2, \ldots, a_n) \downarrow A_\sigma(b_1, b_2, \ldots, b_n)$. $\odot$

12.2 Corectitudinea rescrierii pentru congruenţa semantică

Teorema 12.4 *Toate regulile de deducţie de mai sus sunt corecte pentru congruenţa semantică $\equiv_\Gamma$.*

Demonstraţie: Din lecţia privind logica ecuaţională ştim că regulile de deducţie $\mathbf{R}$, $\mathbf{S}$, $\mathbf{T}$, $\mathbf{C}\Sigma$ şi $\mathbf{Sub}_\Gamma$ sunt corecte.
Deoarece $\mathbf{C}\Sigma$ şi $\mathbf{R}$ implică $\mathbf{CA}\Sigma$ deducem că $\mathbf{CA}\Sigma$ este corectă.
Deoarece $\mathbf{Sub}_\Gamma$, $\mathbf{S}$ şi $\mathbf{T}$ implică $\mathbf{Rew}_\Gamma$ rezultă că $\mathbf{Rew}_\Gamma$ este corectă.
Deoarece $\mathbf{Rew}_\Gamma$ şi $\mathbf{CA}\Sigma$ implică $\mathbf{SRew}_\Gamma$ deducem că $\mathbf{SRew}_\Gamma$ este corectă. $\odot$

Corolar 12.5 $\overset{*}{\Longrightarrow} \subseteq \equiv_\Gamma$. *(Corectitudinea rescrierii pentru congruenţa semantică)*

Rescrierea nu este completă pentru congruenţa semantică deoarece $\mathbf{SRew}_\Gamma$ nu captează întreaga forţa a simetriei.

12.3 Completitudinea întâlnirii prin recriere

Prin definiţie, pentru orice $s \in S$ şi orice $a, d \in A_s$:

$$a \downarrow_\Gamma d \text{ dacă şi numai dacă există } c \in A_s \text{ astfel încât } a \overset{*}{\Rightarrow}_\Gamma c \text{ şi } d \overset{*}{\Rightarrow}_\Gamma c.$$

Relaţia $\downarrow_\Gamma$ este denumită întâlnire prin rescriere.

Propoziţie 12.6 $\downarrow_\Gamma \subseteq \equiv_\Gamma$.

Demonstraţie: Din $a \downarrow_\Gamma d$ deducem existenţa lui c astfel încât: $a \overset{*}{\Rightarrow}_\Gamma c$ şi $d \overset{*}{\Rightarrow}_\Gamma c$. Din aceasta utilizând corectitudinea rescrierii faţă de congruenţa semantică deducem că $a \equiv_\Gamma c$ şi $d \equiv_\Gamma c$ deci $a \equiv_\Gamma d$. $\odot$

Aceasta dovedeşte corectitudinea lui $\downarrow_\Gamma$ în raport cu congruenţa semantică. Pentru a demonstra şi completitudinea sa, adică incluziunea inversă, avem nevoie să presupunem că $\overset{*}{\Rightarrow}_\Gamma$ este confluentă. Folosind observaţia 12.3 deducem că $\downarrow_\Gamma$ este o congruenţă.

Lemă 12.7 *Dacă $\overset{*}{\Rightarrow}_\Gamma$ este confluentă, atunci congruenţa $\downarrow_\Gamma$ este închisă la substituţie.*

Demonstraţie: Fie $(\forall X)\, l \overset{\circ}{=}_s r$ **if** $H \in \Gamma$ şi fie $h:\ T_\Sigma(X) \longrightarrow \mathcal{A}$ un morfism pentru care $h_t(u) \downarrow_\Gamma h_t(v)$ pentru orice $u \overset{\circ}{=}_t v \in H$. Pentru orice $u \overset{\circ}{=}_t v \in H$ există $a_{uv} \in A_t$ cu proprietăţile $h_t(u) \overset{*}{\Rightarrow}_\Gamma a_{uv}$ şi $h_t(v) \overset{*}{\Rightarrow}_\Gamma a_{uv}$. Deoarece $\overset{*}{\Rightarrow}_\Gamma$ este închisă la $\mathbf{Rew_\Gamma}$ deducem că $h_s(l) \overset{*}{\Rightarrow}_\Gamma h_s(r)$. Prin urmare $h_s(l) \downarrow_\Gamma h_s(r)$. $\odot$

Doarece congruenţa $\downarrow_\Gamma$ este închisă la substituţie deducem din propoziţia 7.4 că factorizarea lui $\mathcal{A}$ prin $\downarrow_\Gamma$ este o Γ–algebră.

Teorema 12.8 *Dacă* $\overset{*}{\Rightarrow}_\Gamma$ *este confluentă, atunci* $\downarrow_\Gamma = \equiv_\Gamma$.

Demonstraţie: Folosind propoziţia 12.6 mai trebuie să demonstrăm doar $\equiv_\Gamma \subseteq \downarrow_\Gamma$.

Incluziunea de mai sus rezultă aplicând propoziţia 7.5 congruenţei $\downarrow_\Gamma$ care conform propoziţiei precedente este închisă la substituţie.

În concluzie $\downarrow_\Gamma = \equiv_\Gamma$ ceea ce demonstrează completitudinea lui $\downarrow_\Gamma$. $\odot$

Din punct de vedere practic această teoremă arată că demonstrarea propoziţiei $a \equiv_\Gamma c$ *poate fi redusă la a demonstra* $a \downarrow_\Gamma c$ *dacă rescrierea este confluentă. Prin urmare principala problemă devine*

"poate o maşină să demonstreze că $a \downarrow_\Gamma c$*?"*

12.4 Forme normale

12.4.1 La mulţimi

Presupunem în continuare că $\succ$ este o preordine pe mulţimea A.

Definiţia 12.9 Elementul $n \in A$ se numeşte **o formă normală** pentru relaţia $\succ$ pe mulţimea A dacă

$$(\forall b \in A)(n \succ b \Rightarrow n = b).$$

In limbajul teoriei mulţimilor partial ordonate, o formă normală este numită element minimal. Unele texte folosesc "ireductibil" sau formă "canonică" în loc de formă "normală".

Fie N mulţimea elementelor din A care sunt forme normale pentru $\succ$. Presupunem **axioma Formei Normale unice**

$$\mathbf{FN!}\ \ (\forall a \in A)(\exists!\, fn(a) \in N) a \succ fn(a).$$

Observaţia 12.10 *Dacă* $a \succ d$, *atunci* $fn(a) = fn(d)$.

Demonstraţie: Din ipoteză şi $d \succ fn(d)$ deducem $a \succ fn(d)$, deci din unicitatea formei normale a lui a deducem $fn(a) = fn(d)$.

Observaţia 12.11 *Axioma* **FN!** *implică* $\succ$ *este confluentă.*

Demonstraţie: Presupunem $a \succ d$ şi $a \succ c$. Deducem $fn(a) = fn(d) = fn(c)$, deci $d \succ fn(d)$ şi $c \succ fn(d)$.

Observaţia 12.12 *Funcţia* $fn:\ A \longrightarrow N$ *este surjectivă şi*

$$a \downarrow d \Leftrightarrow fn(a) = fn(d).$$

Demonstraţie: Deoarece pentru orice element n în formă normală $n = fn(n)$ rezultă surjectivitatea funcţiei fn.

Presupunem $fn(a) = fn(d)$. Deoarece $a \succ fn(a)$ şi $d \succ fn(a)$ deducem $a \downarrow d$.

Presupunem $a \downarrow d$. Fie $c \in A$ astfel încât $a \succ c$ şi $d \succ c$. Deducem $fn(a) = fn(c)$ şi $fn(d) = fn(c)$, deci $fn(a) = fn(d)$. $\odot$

Din punct de vedere practic această observaţie arată că dacă rescrierea are proprietatea **FN!** *demonstrarea lui* $a \downarrow_\Gamma d$ *este echivalentă cu egalitatea formelor normale pentru rescriere ale lui* a *şi* d.

12.5 Relaţii canonice

In practică pentru a verifica că $\stackrel{*}{\Rightarrow}_\Gamma$ satisface axioma **FN!**, preferăm uneori să verificăm că satisface proprietăţile de confluenţă şi terminare.

Definiţia 12.13 Spunem că $\succ$ are proprietatea de **terminare** dacă nu există şiruri $\{a_n\}$ de elemente din A astfel încât pentru orice număr natural n să avem $a_n \succ a_{n+1}$ şi $a_n \neq a_{n+1}$. $\odot$

Remarcăm că terminarea, concept venit din informatică, coincide cu buna fondare, concept provenit din lumea relaţiilor sau din teoria axiomatică a mulţimilor.

Observaţia 12.14 *Dacă $\succ$ are proprietatea de terminare, atunci pentru orice element a din A există un element în formă normală $n \in N$ cu proprietatea $a \succ n$.*

Demonstraţie: Raţionând prin absurd, a nu este în forma normală; prin urmare există $a_1 \in A$ cu proprietăţile $a \succ a_1$ şi $a \neq a_1$. Raţionând în continuare prin absurd, a_1 nu este în forma normală; prin urmare există $a_2 \in A$ cu proprietăţile $a_1 \succ a_2$ şi $a_1 \neq a_2$. Continuând acest raţionament prin inducţie putem construi un şir de elemente care contrazice proprietatea de terminare. $\odot$

Definiţia 12.15 Dacă preordinea $\succ$ este confluentă şi are proprietatea de terminare, atunci $< A, \succ >$ se numeşte **canonică**. $\odot$

Propoziţie 12.16 *Dacă $< A, \succ >$ este canonică atunci axioma* **FN!** *este verificată.*

Demonstraţie: Unicitatea va rezulta din confluenţă. Presupunând că n' şi n'' sunt două elemente în formă normală cu proprietatea $a \succ n'$ şi $a \succ n$" din confluenţă există $d \in A$ astfel încât $n' \succ d$ şi n" $\succ d$. Deoarece n' şi n" în sunt formă normală deducem că $n' = d$ şi n" $= d$, deci $n' = n$". $\odot$

Din punct de vedere practic terminarea este necesară pentru a opri execuţia calculatorului iar confluenţa este necesară pentru completitudine. Aceste proprietăţi implică **FN!**. *Pentru a demonstra că $a \equiv_\Gamma c$, adică $a \downarrow_\Gamma c$, este suficient conform observaţiei 12.12 să rescriem a si c in forma lor normală $fn(a)$ şi $fn(c)$ pentru ca apoi să verificăm egalitatea $fn(a) = fn(c)$.*

13 RESCRIERE ÎN SUBTERMENI

Am văzut că rescrierea canonică dă o metodă de demonstrare automată a propoziţiilor valide. Următoarea problemă constă în a arăta cum regulile de deducţie pentru rescriere pot fi utilizate în mod practic.

Teoremele din această lecţie explică modul în care se fac rescrierile în limbajele de programare declarativă.

13.1 Completitudinea rescrierii în subtermeni

Propoziţiile **??** şi 11.5 ne asigură că rescrierea este cea mai mică preordine închisă la $\mathbf{SRew}_\Gamma$ compatibilă cu toate operaţiile din $\mathcal{A}$. Teorema următoare ne spune ceva mai mult.

Teorema 13.1 *Rescrierea este cea mai mică preordine din A închisă la $\mathbf{SRew}_\Gamma$.*

Demonstraţie: Fie R_Γ cea mai mică relaţie pe A care este închisă la $\mathbf{R}$, $\mathbf{T}$ şi $\mathbf{SRew}_\Gamma$. Din definiţia lui R_Γ deducem că $R_\Gamma \subseteq \stackrel{*}{\Longrightarrow}$.

Pentru a demonstra incluziunea contrară este suficient să arătăm că R_Γ este închisă în $\mathbf{C}\Sigma$.

Dar R_Γ este tranzitivă şi reflexivă, deci este suficient să arătăm că R_Γ este închisă la $\mathbf{CA}\Sigma$, adică pentru fiecare $\sigma \in \Sigma_{s_1 \ldots s_n, s'}$ şi $a_i \in A_{s_i}$ că $a \, R_\Gamma \, d$ implică

$$A_\sigma(a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n) \, R_\Gamma \, A_\sigma(a_1, \ldots, a_{i-1}, d, a_{i+1}, \ldots, a_n).$$

Demonstrăm că mulţimea

$$D = \{a \, R_\Gamma \, d \mid \sigma \in \Sigma_{s_1 s_2 \ldots s_n, s}; (\forall a_j \in A_{s_j}) A_\sigma(a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n) \, R_\Gamma \, A_\sigma(a_1, \ldots, a_{i-1}, d, a_{i+1}, \ldots, a_n)\}$$

este închisă la **R**, **T** şi $\mathbf{SRew}_\Gamma$. Observăm că $D \subseteq R_\Gamma$.

Reflexivitatea lui D rezultă din reflexivitatea lui R_Γ.

Presupunem că (a,u) şi (u,d) sunt în D. Folosind tranzitivitatea lui R_Γ din $a\,R_\Gamma\,u$ şi $u\,R_\Gamma\,d$ deducem $a\,R_\Gamma\,d$ şi din $A_\sigma(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)\ R_\Gamma\ A_\sigma(a_1,\ldots,a_{i-1},u,a_{i+1},\ldots,a_n)$ şi
$A_\sigma(a_1,\ldots,a_{i-1},u,a_{i+1},\ldots,a_n)\ R_\Gamma\ A_\sigma(a_1,\ldots,a_{i-1},d,a_{i+1},\ldots,a_n)$ deducem
$A_\sigma(a_1,\ldots,a_{i_1},a,a_{i+1},\ldots,a_n)\,R_\Gamma\,A_\sigma(a_1,\ldots,a_{i-1},d,a_{i+1},\ldots,a_n)$. Prin urmare $(a,d)\in D$. Deci D este tranzitivă.

Probăm că D este închisă la $\mathbf{SRew}_\Gamma$. Fie $(\forall X) l \overset{\circ}{=}_s r$ **if** $H \in \Gamma$, un morfism $h : T_\Sigma(X) \to \mathcal{A}$ cu proprietatea că pentru orice $u \overset{\circ}{=}_t v \in H$. există $a_{uv} \in A_t$ astfel încât $(h_t(u), a_{uv}) \in D$ şi $(h_t(v), a_{uv}) \in D$ Pentru orice context $c \in T_\Sigma(A \cup \{\bullet\})_{s'}$ trebuie arătat că $(c[h_s(l)], c[h_s(r)]) \in D$.

Deoarece R_Γ este închisă la $\mathbf{SRew}_\Gamma$ din $h_t(u)\,R_\Gamma\,a_{uv}$ şi $h_t(v)\,R_\Gamma\,a_{uv}$ pentru orice $u \overset{\circ}{=}_t v \in H$ deducem

$$c[h_s(l)]\,R_\Gamma\,c[h_s(r)] \qquad \text{şi} \qquad c'[h_s(l)]\,R_\Gamma\,c'[h_s(r)]$$

unde $c' = \sigma(a_1,\ldots,a_{i-1},c,a_{i+1},\ldots,a_n)$ este un context.
Din $c'[h_s(l)] = A_\sigma(a_1,\ldots,a_{i-1},c[h_s(l)],a_{i+1}\ldots,a_n)$ şi $c'[h_s(r)] = A_\sigma(a_1,\ldots,a_{i-1},c[h_s(r)],a_{i+1},\ldots,a_n)$ obţinem

$$A_\sigma(a_1,\ldots,a_{i-1},c[h_s(l)],a_{i+1},\ldots,a_n)\,R_\Gamma\,A_\sigma(a_1,\ldots,a_{i-1},c[h_s(r)],a_{i+1},\ldots,a_n).$$

Deci $(c[h_s(l)], c[h_s(r)]) \in D$.

Deoarece R_Γ este cea mai mică relaţie care este închisă la **R**, **T** şi $\mathbf{SRew}_\Gamma$ şi D este închisă la **R**, **T** şi $\mathbf{SRew}_\Gamma$ rezultă că $R_\Gamma \subseteq D$. Prin urmare $R_\Gamma = D$.

Probăm că R_Γ este închisă la $\mathbf{CA}\Sigma$. Fie $a \overset{\circ}{=} d$ în R_Γ. rezultă că $a \overset{\circ}{=} d$ este în D. Prin urmare $(\forall\sigma \in \Sigma_{s_1s_2\ldots s_n,s})(\forall a_j \in A_{s_j})A_\sigma(a_1,\ldots,a_{i-1},a,a_{i+1},\ldots,a_n)\,R_\Gamma\,A_\sigma(a_1,\ldots,a_{i-1},d,a_{i+1},\ldots,a_n)$. Deci R_Γ este închisă la $\mathbf{CA}\Sigma$.

Deoarece R_Γ este reflexivă, tranzitivă şi închisă la $\mathbf{CA}\Sigma$ deducem că R_Γ este închisă la $\mathbf{C}\Sigma$.

Deoarece R_Γ este închisă la **R**, **T**, $\mathbf{C}\Sigma$ şi $\mathbf{Rew}_\Gamma$, iar $\overset{*}{\Longrightarrow}$ este cea mai mică relaţie închisă la acestea deducem că $\overset{*}{\Longrightarrow} \subseteq R_\Gamma$.

În concluzie $R_\Gamma = \overset{*}{\Longrightarrow}$. $\odot$

13.2 Rescriere într-un pas

Fie c în $T_\Sigma(A \cup \{\bullet\})_{s'}$ un context.

Fie $(\forall X) l \overset{\circ}{=}_s r$ **if** $H \in \Gamma$ şi fie $h : T_\Sigma(X) \to \mathcal{A}$ un morfism astfel încât pentru orice $u \overset{\circ}{=}_t v \in H$ există $a_{uv} \in A_t$ cu proprietăţile $h_t(u) \overset{*}{\Rightarrow}_\Gamma a_{uv}$ şi $h_t(v) \overset{*}{\Rightarrow}_\Gamma a_{uv}$. Dacă $c[h_s(l)] \neq c[h_s(r)]$ spunem că $c[h_s(l)]$ se rescrie într-un pas în $c[h_s(r)]$ şi scriem

$$c[h_s(l)] \Rightarrow c[h_s(r)].$$

Propoziţie 13.2 *Rescrierea este închiderea reflexivă şi tranzitivă a rescrierii într-un pas.*

Demonstraţie: Fie $\overset{*}{\Rightarrow}$ închiderea reflexiv-tranzitivă a rescrierii într-un pas.

Deoarece $\Rightarrow \subseteq \overset{*}{\Rightarrow}_\Gamma$ şi $\overset{*}{\Rightarrow}_\Gamma$ este preordine deducem incluziunea $\overset{*}{\Rightarrow} \subseteq \overset{*}{\Rightarrow}_\Gamma$.

Reciproc, deoarece $\overset{*}{\Rightarrow}$ este reflexivă şi tranzitivă este suficient să arătăm că $\overset{*}{\Rightarrow}$ este închisă la $\mathbf{SRew}_\Gamma$.
Fie $(\forall X)\, l \overset{\circ}{=}_s r$ **if** $H \in \Gamma$, $h : T_\Sigma(X) \to \mathcal{A}$ un morfism astfel încât pentru orice $u \overset{\circ}{=}_t v \in H$ există $a_{uv} \in A_t$ cu proprietăţile $h_t(u) \overset{*}{\Rightarrow} a_{uv}$ şi $h_t(v) \overset{*}{\Rightarrow} a_{uv}$ şi c un context din $\mathcal{A}[z]$. Din incluziunea deja demonstrată rezultă că pentru orice $u \overset{\circ}{=}_t v \in H$ există $a_{uv} \in A_t$ cu proprietăţile $h_t(u) \overset{*}{\Rightarrow}_\Gamma a_{uv}$ şi $h_t(v) \overset{*}{\Rightarrow}_\Gamma a_{uv}$. Avem cazurile:

1. Dacă $c[h(l)] = c[h(r)]$ deducem din reflexivitatea lui $\overset{*}{\Rightarrow}$ că $c[h(l)] \overset{*}{\Rightarrow} c[h(r)]$.

2. Altfel, dacă $c[h(l)] \neq c[h(r)]$, atunci $c[h(l)] \Rightarrow c[h(r)]$, deci $c[h(l)] \overset{*}{\Rightarrow} c[h(r)]$ deoarece rescrierea într-un pas este inclusă în închiderea reflexivă şi tranzitivă a rescrierii într-un pas. $\odot$

În continuare rescrierea într-un pas va fi notată cu $\Rightarrow_\Gamma$ sau $\Rightarrow$.

13.3 Consideraţii metodologice

Pentru a demonstra că $a \equiv_\Gamma b$ este suficient, în ipotezele de mai sus, să arătăm că $fn_s(a) = fn_s(b)$ şi aceasta se face prin rescrierea lui a, respectiv a lui b în formele lor normale, pentru a verifica că sunt egale.

Fie $a \in A_s$ elementul de rescris. Se caută

$(\forall X) l \stackrel{\circ}{=}_s r$ **if** H în Γ, $h : T_\Sigma(X) \longrightarrow \mathcal{A}$ şi un context c in $\mathcal{A}[z]$ astfel încât $c[h_s(l)] = a$.

Pentru a putea face rescrierea lui $a = c[h_s(l)]$ în $c[h_s(r)]$ este suficient să arătăm că $h_t(u) \downarrow_\Gamma h_t(v)$ pentru orice $u \stackrel{\circ}{=}_t v \in H$. Se întrerupe rescrierea principală pentru a face acestă verificare şi în caz de reuşită se rescrie a în $c[h_s(r)]$. In caz de nereuşită se continuă căutarile pentru a reajunge în situaţia de mai sus. Intrucât în timpul verificării uneia dintre condiţiile $h_t(u) \downarrow_\Gamma h_t(v)$ se rescriu atât $h_t(u)$ cât şi $h_t(v)$ în forma lor normală, fenomenul de mai sus se poate repeta se apelează la mecanismul numit backtracking.

Dacă Γ conţine doar ecuaţii necondiţionate, rescrierea într-un singur pas nu depinde de alte rescrieri, ca în cazul general. Acest fapt explică de ce, în acest caz metoda backtracking nu este folosită în implementarea rescrierii.

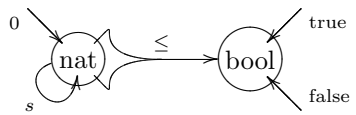
Remarcăm diferenţa esenţială dintre o mulţime Γ de axiome şi un program. Un program este o mulţime de axiome care verifică nişte condiţii şi care sunt scrise respectând sintaxa unui limbaj de programare declarativă.

Am pus deja în evidenţă nişte condiţii suficiente pe care trebuie să le îndeplinească o mulţime Γ de axiome pentru a deveni program: rescrierea generată trebuie să fie confluentă şi să aibă proprietatea de terminare.

Practic, de exemplu, într-o ecuaţie din Γ este interzis ca în membrul drept să apară o variabilă care nu apare în membrul stâng, deoarece în caz contrar rescrierea nu are proprietatea de terminare. De asemenea este interzis ca membrul stâng al unei ecuaţii să fie o variabilă. Dacă ecuaţia $x \stackrel{\circ}{=}_s r$ cu x variabilă ar fi în Γ, atunci orice expresie t de sort s ar putea fi rescrisă $t \Longrightarrow (x \leftarrow t)r$.

Exemplu Signatura are două sorturi *nat* şi *bool*; trei constante *true*, $false$ de sort *bool* şi 0 de sort *nat*; două simboluri de operaţii $s : nat \longrightarrow nat$ şi $\leq$: *nat nat* $\longrightarrow$ *bool*. Axiomele acceptate cu x şi y variabile de sort *nat* sunt

1. $0 \leq x = true$ 2. $s(x) \leq s(y) = true$ **if** $x \leq y = true$ 3. $s(x) \leq 0 = false$



Vrem să rescriem $s(s(0)) \leq s(s(z))$. Se poate aplica numai axioma 2 pentru $x = s(0)$ şi $y = s(z)$.

Înainte de aceasta trebuie să rescriem $s(0) \leq s(z)$. Se poate aplica numai axioma 2 pentru $x = 0$ şi $y = z$.

Înainte de aceasta rescriem $0 \leq z \Rightarrow_\Gamma true$ cu axioma 1.

Deoarece am obţinut *true* putem rescrie $s(0) \leq s(z) \Rightarrow_\Gamma true$.

Deoarece prin rescrierea lui $s(0) \leq s(z)$ am obţinut *true* putem rescrie $s(s(0)) \leq s(s(z)) \Rightarrow_\Gamma true$.

14 UNIFICARE

In semantica operaţională a limbajelor de specificaţie rescrierea joacă un rol primordial. Pentru a rescrie un subtermen $a \in T_\Sigma(Y)$ al termenului de rescris cu ajutorul unei reguli se pune problema identificării lui a cu imaginea membrului stâng $l \in T_\Sigma(X)$ al unei reguli din Γ printr-o substituţie(morfism) de la $T_\Sigma(X)$ la $T_\Sigma(Y)$. Această problemă de potrivire(matching în engleză) se poate rezolva cu ajutorul algoritmului de unificare. Mai precis se caută a se unifica l cu a în care toate variabilele din a sunt considerate constante. Dacă în l şi a există variabile comune, aceste variabile comune pot fi înlocuite în axioma utilizată din Γ prin variabile noi. Acest fapt nu restrânge generalitatea deoarece variabilele din axiome sunt cuantificate universal fapt ce permite redenumirea lor.

Toate algebrele sunt presupuse libere. Prin **substituţie** vom înţelege un morfism între două algebre libere.

Problema unificării. Se dau un număr finit de egalităţii formale $l_i \stackrel{\circ}{=}_{s_i} r_i$ şi se cere găsirea unui unificator, adică a unei substituţii u cu proprietatea $u(l_i) = u(r_i)$ pentru orice i.

De fapt mulţimea finită de egalităţii formale este privită ca un sistem de ecuaţii. Şi de aici poate proveni denumirea de ecuaţie folosită ca un sinonim pentru egalităţile formale. Acest subcapitol descrie modul de rezolvare a unui astfel de sistem chiar în algebra liberă din care provin termenii ecuaţiilor. Cuvântul unificator înseamnă de fapt soluţie în algebra liberă din care provin termenii ecuaţiilor.

Algoritmul prezentat aici este ales pentru simplcitatea sa. Menţionăm existenţa unor algoritmi mai eficienţi.

14.1 Algoritmul de unificare

Vom lucra cu două liste: soluţie şi de rezolvat. Initial lista soluţie este vidă şi lista de rezolvat conţine mulţimea ecuaţiilor de unificat.

Algoritmul de unificare constă în execuţia nedeterministă a urmatorilor trei paşi.

1. **Scoate.** Se scoate din lista de rezolvat orice ecuaţie de forma $t \stackrel{\circ}{=} t$.
2. **Descompune.** Orice ecuaţie din lista de rezolvat de forma

$$f(a_1, a_2, \ldots, a_n) \stackrel{\circ}{=} f(b_1, b_2, \ldots, b_n)$$

   se înlocuieşte cu ecuaţiile $a_1 \stackrel{\circ}{=} b_1$, $a_2 \stackrel{\circ}{=} b_2$, ..., $a_n \stackrel{\circ}{=} b_n$,
3. **Elimină.** Orice ecuaţie din lista de rezolvat, de forma $x \stackrel{\circ}{=} t$ sau $t \stackrel{\circ}{=} x$ în care x este o variabilă care nu apare în t este mutată sub forma $x \stackrel{\circ}{=} t$ în lista soluţie. In toate celelalte ecuaţii x se substitue cu t.

Algoritmul este oprit cu concluzia inexistenţei unui unificator în urmatoarele două cazuri.
1) Existenţa în lista de rezolvat a unei ecuaţii de forma $f(a_1, a_2, \ldots, a_n) \stackrel{\circ}{=} g(b_1, b_2, \ldots, b_m)$ cu $f \neq g$,
2) Existenţa în lista de rezolvat a unei ecuaţii de forma $x \stackrel{\circ}{=} t$ sau $t \stackrel{\circ}{=} x$ în care x este o variabilă care apare în t şi este diferită de t.

Oprirea normală a algoritmului se face prin epuizarea listei de rezolvat, caz în care, aşa cum vom proba mai jos, lista soluţie ne furnizează cel mai general unificator.

14.2 Terminare

Terminarea algoritmului este probată folosind în ordine lexicografică două criterii exprimate prin numere naturale:

1. numărul variabilelor care apar în lista de rezolvat, care in funcţie de pasul algoritmului utilizat are următoarea comportare
   (a) **Scoate**: rămâne egal sau se micşorează,
   (b) **Descompune**: rămâne egal,
   (c) **Elimină**: se micşorează
2. numărul apariţiilor simbolurilor(semnelor) care apar în lista de rezolvat, care se micşorează în cele două cazuri care ne mai interesează **Scoate** şi **Descompune**.

14.3 Corectitudine

Corectitudinea algoritmului se bazează pe demonstrarea faptului că mulţimea unificatorilor pentru ecuaţiile din reuniunea celor două liste este un invariant, adică nu se modifică prin aplicarea celor trei paşi ai algoritmului.

Deoarece pentru pasul **Scoate** afirmaţia este evidentă ne referim doar la ceilalţi paşi.

Descompune: Observăm că pentru orice substituţie s egaliatea

$$s(f(a_1, a_2, \ldots, a_n)) \stackrel{\circ}{=} s(f(b_1, b_2, \ldots, b_n))$$

este echivalentă cu

$$f(s(a_1), s(a_2), \ldots, s(a_n)) \stackrel{\circ}{=} f(s(b_1), s(b_2), \ldots, s(b_n))$$

adică cu $s(a_i) \stackrel{\circ}{=} s(b_i)$ pentru orice $i \in [n]$.

Elimină: Observăm că orice unificator u pentru ecuaţiile din reuniunea celor două liste atât înainte de aplicarea pasului cât şi după aceasta trebuie să satisfacă egalitatea $u(x) = u(t)$. Pentru o substituţie s cu proprietatea $s(x) = s(t)$ observăm că

$$x \leftarrow t; s = s$$

deoarece $(x \leftarrow t; s)(x) = s(t) = s(x)$ şi $(x \leftarrow t; s)(y) = s(y)$ pentru orice alta variabilă y. Prin urmare pentru o astfel de substituţie

$$s(l) = s(r) \text{ dacă şi numai dacă } s((x \leftarrow t)(l)) = s((x \leftarrow t)(r))$$

ceea ce arată că un unificator pentru ecuaţiile din reuniunea celor două liste dinainte de aplicarea pasului este unificator şi pentru ecuaţiile din reuniunea celor două liste de după aplicarea pasului şi reciproc.

Algoritmul este oprit cu concluzia inexistenţei unui unificator deoarece în cele două situaţii menţionate se constată că mulţimea unificatorilor este vidă.

Să observăm că variabilele care apar în membrul stâng al ecuaţiilor din lista soluţie sunt diferite două câte două şi nu mai apar în nici una dintre celelalte ecuaţii din cele două liste.

Faptul poate fi dovedit prin inducţie.

Menţionăm că aplicarea primilor doi paşi ai algoritmului nu modifică lista soluţie şi nu produc apariţii noi de variabile în cele două liste.

Fie $x \stackrel{\circ}{=} t$ ecuaţia introdusă în lista soluţie prin aplicarea pasului **Elimină**. Deoarece variabilele din membrul stâng ai listei soluţie precedentă nu apar în celelalte ecuaţii rezultă că variabila x este diferită de celelalte variabile care apar în membrul stâng al ecuaţiilor din lista soluţie. Prin urmare, variabilele din membrul stâng al noii liste soluţie sunt diferite între ele.

In plus prin substituirea lui x cu t în celelalte ecuaţii variabila x dispare din ele deoarece x nu apare în t. Deoarece nici x şi nici variabilele din membrul stâng ai listei soluţie precedentă nu apar în t rezultă că după efectuarea substituţiei lui x cu t variabilele din membrul stâng ai listei soluţie nu apar în restul ecuaţiilor.

Să presupunem că algoritmul s-a terminat prin epuizarea listei de rezolvat. Să notăm cu k numărul ecuaţiilor din lista soluţie şi cu $x_i \stackrel{\circ}{=} t_i$ pentru $i \in [k]$ ecuaţiile din ea.

Fie U substituţia definită prin

$$U(x_i) = t_i \text{ pentru orice } i \in [k].$$

Definiţia este corectă deoarece variabilele x_i sunt distincte. Deoarece variabilele x_i nu apar în termenii t_i deducem că $U(t_i) = t_i$, prin urmare $U(t_i) = U(x_i)$, deci U este un unificator. Vom dovedi că este cel mai general.

Observăm că pentru orice substituţie s compunerea $U; s$ este tot un unificator. Vom arăta că orice alt unificator este de această formă. Fie u un alt unificator, adica $u(x_i) = u(t_i)$ pentru orice $i \in [k]$. Observăm că $U; u = u$, căci $u(U(x_i)) = u(t_i) = u(x_i)$ pentru orice $i \in [k]$ şi $u(U(y)) = u(y)$ pentru orice altă variabilă y.

Deci U este cel mai general unificator, deoarece orice alt unificator poate fi exprimat ca o compunere a lui U cu o substituţie. In plus observăm că el este idempotent deoarece $U; U = U$.

Notând $V = X - \{x_1, x_2, \ldots, x_k\}$ observăm că $V \subseteq X$ şi că

$$U : T_\Sigma(X) \longrightarrow T_\Sigma(V)$$

este un Σ–morfism cu proprietatea $U(v) = v$ pentru orice $v \in V$.

În cele ce urmează vom scrie CGU ca o prescurtare pentru Cel mai General Unificator.

15 LOCAL CONFLUENŢĂ

Dacă terminarea unui program poate fi chiar şi o problemă deschisă, problema confluenţei are soluţie, adică se pot scrie programe care pentru o mulţime dată de reguli Γ să constate, în ipoteza terminării, dacă Γ-rescrierea este confluentă. De fapt se probează local confluenţa care împreună cu terminarea implică confluenţa.

Reamintim că relaţia de rescriere este confluentă dacă

$$(\forall a, b, c)\ [(a \stackrel{*}{\Longrightarrow} b \text{ şi } a \stackrel{*}{\Longrightarrow} c) \text{ implică } (\exists d)\ (b \stackrel{*}{\Longrightarrow} d \text{ şi } c \stackrel{*}{\Longrightarrow} d)].$$

Definiţia 15.1 Spunem că rescrierea este **local confluentă** dacă

$$(\forall a, b, c)\ [(a \Longrightarrow b \text{ şi } a \Longrightarrow c) \text{ implică } (\exists d)\ (b \stackrel{*}{\Longrightarrow} d \text{ şi } c \stackrel{*}{\Longrightarrow} d)].$$

Reamintim că rescrierea are proprietatea de terminare, dacă nu există şiruri $\{a_n\}_n$ cu proprietatea $a_n \Longrightarrow a_{n+1}$ pentru orice n număr natural.

Propoziţie 15.2 *Dacă rescrierea este local confluentă şi are proprietatea de terminare atunci rescrierea este confluentă.*

Demonstraţie: Reamintim că proprietatea de terminare ne aisgură pentru orice a existenţa unei forme normale n cu proprietatea $a \stackrel{*}{\Longrightarrow} n$. Vom demonstra unicitatea formei normale fapt ce implică confluenţa. Notăm cu N mulţimea formelor normale. Fie M mulţimea elementelor care se pot rescrie în cel puţin două forme normale.

$$M = \{a : (\exists n_1, n_2 \in N) a \stackrel{*}{\Longrightarrow} n_1, a \stackrel{*}{\Longrightarrow} n_2, n_1 \neq n_2\}.$$

Probăm că $a \in M$ implică $(\exists b \in M) a \Longrightarrow b$.

Deoarece $a \in M$ există $n_1, n_2 \in N$ cu proprietăţile $a \stackrel{*}{\Longrightarrow} n_1$, $a \stackrel{*}{\Longrightarrow} n_2$ şi $n_1 \neq n_2$. Dacă a ar fi o formă normală, atunci $a = n_1$ şi $a = n_2$ ceea ce contrazice $n_1 \neq n_2$.

Deoarece a nu este formă normală există elemntul b cu proprietăţile $a \Longrightarrow b$ şi $b \stackrel{*}{\Longrightarrow} n_1$. La fel există c cu proprietăţile $a \Longrightarrow c$ şi $c \stackrel{*}{\Longrightarrow} n_2$. Deoarece rescrierea este local confluenţa există elementul d cu proprietăţile $b \stackrel{*}{\Longrightarrow} d$ şi $c \stackrel{*}{\Longrightarrow} d$. Deoarece terminarea implică existenţa formei normale deducem $\exists n \in N$ cu proprietatea $d \stackrel{*}{\Longrightarrow} n$.

Deoarece $n_1 \neq n_2$ deducem $n \neq n_1$ sau $n \neq n_2$.

Dacă $n \neq n_1$ deoarece $b \stackrel{*}{\Longrightarrow} n_1$ şi $b \stackrel{*}{\Longrightarrow} n$ decucem $b \in M$. Dacă $n \neq n_2$ deoarece $c \stackrel{*}{\Longrightarrow} n$ şi $c \stackrel{*}{\Longrightarrow} n_2$ deducem $c \in M$.

Probăm unicitatea formei normale. Raţionând prin absurd presupunem că M este nevidă. Plecând de la $a_1 \in M$ aplicând observaţia de mai sus deducem existenţa lui $a_2 \in M$ cu proprietatea $a_1 \Longrightarrow a_2$. Repetând raţionamentul printr-o inducţie deducem existenţa unui şir $\{a_n\}_n$ cu proprietatea $a_n \Longrightarrow a_{n+1}$ pentru orice n natural. Deci rescrierea nu are proprietatea de terminare, în contradicţie cu ipoteza. $\odot$