2 ALGEBRE LIBERE - APLICAŢII

Virgil Emil Căzănescu

March 11, 2008

1 Expresii

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. În continuare 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 poloneză inversă $xyz**$ şi $xy*z*$. Ne interesează *o definiţie a conceptului de expresie care să fie independentă de forma de scriere a acesteia.*

Definiţia 1.1 Σ-algebra $\mathcal{A} = (A_s, A_\sigma)$ se numeşte liber generată de $V \subseteq A$ dacă pentru orice Σ-algebră D şi pentru orice funcţie sortată $f : V \to D$, există un unic morfism de Σ-algebre $f^\# : A \to D$ care extinde f.

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

Definiţia 1.2 Se numeşte *expresie* un element dintr-o algebră liberă.

Bineînţeles că acest concept este încă dependent de signatura cu care lucrăm. 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 generatorilor.

1.2 Evaluarea expresiilor

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

Menţionăm că a da valori variabilelor din mulţimea X în algebra $\mathcal{D}$ este echivalent cu a da o funcţie $v : X \longrightarrow \mathcal{D}$. 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 $v : X \longrightarrow \mathcal{D}$ 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 algebre de la $T_\Sigma(X)$ la $\mathcal{D}$ şi $Set_S(X, \mathcal{D})$ mulţimea funcţiilor S-sortate de la X la $\mathcal{D}$. Fie

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

funcţia restricţie, adică $r(h) : X \longrightarrow \mathcal{D}$ este restricţia morfismului h 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 ne mai făcând 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 spunem 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 1.3 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 $h : X \longrightarrow \mathcal{D}$. □

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

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

1.3 Semantica instrucţiunii de atribuire

Fie X mulţimea variabilelor utilizate în 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 $S(x := e)$ 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

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

2 Unicitatea abstracţie de un izomorfism a algebrelor libere

Teorema 2.1 *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 şi 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 $i_B; f = i_A$.

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 $f; g = 1_\mathcal{A}$.

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 $g; f = 1_\mathcal{B}$.

Deci f şi g sunt izomorfisme inverse unul altuia.

Propoziţie 2.2 *Orice algebră izomorfă cu o algebră liberă este algebră liberă.*

2.1 Algebre iniţiale

Definiţia 2.3 O Σ-algebră $\mathcal{I}$ se numeţe 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.
Din teoremele de mai sus rezultă că:

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

Acest fapt are aplicaţii importante în informatică.
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 dare.
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.

3 Tipuri abstracte de date - introducere

Un tip de date se numeşte **abstract** dacă este unic determinat abstracţie făcând de un izomorfism. 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 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 este definit abstract ca fiind semiinelul iniţial.
Tipul numerelor întregi este definit abstract ca fiind inelul iniţial.

Definiţiile de mai sus, deşi corecte sunt ineficiente, deoarece nu face posibilă execuţia de calcule. Prin urmare dorim alte definiţii echivalente dar prin care calculatorul să fie învăţat să facă calcule. Vom exemplifica pentru numere naturale fără intra în prea multe detalii.

3.1 Tipul abstract al numerelor naturale

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 3.1 *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

$h(0_N) = O_A$
$h(n + 1) = s_A(h(n))$ 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
$g(n+1) = g(s_N(n)) = s_A(g(n)) = s_A(h(n)) = h(s_N(n)) = h(n+1)$. □

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 adauge la mulţimea de mai sus a numerelor natural 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 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, deoarece 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:

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

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

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 făcută mai târziu.

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 .