Calculul fluxnomial
-de la teorie la practică-

student Alin Ştefănescu
îndrumat de conf. dr. Gheorghe Ştefănescu

Universitatea Bucurşti
iunie 2000

Rezumat

Calculul fluxnomial este un calcul abstract pentru reţele văzute ca hipergrafuri direcţionate şi etichetate. Se lucrează cu expresii fluxnomiale care sunt obţinute prin aplicarea repetată a operaţiilor de juxtapunere, compunere şi feedback, pornind de la atomi de tip variabilă sau conexiuni. Cu ajutorul acestui formalism putem modela scheme de programe, grafuri, relaţii, reţele de flux de date, circuite etc.

Lucrarea conţine trei părţi, trecând gradat, după cum ne spune şi subtitlul ales, *de la teorie la practică*.

Prima parte este o prezentare succintă a calculului fluxnomial. Se porneşte de la definiţiile de bază, setul de axiome BNA; se trece apoi la prezentarea algebrei reţelelor, a formei normale asociate unei expresii şi a simulării via bijecţii; pentru ca în final să fie expuse modelele aditiv, multiplicativ şi mixt.

În a doua parte, se aplică teoria fluxnoamelor în modelarea câtorva concepte din informatică: grafurile şi apoi clasele şi obiectele din domeniul programării orientate pe obiecte.
Pentru grafuri, sunt oferiţi algoritmi de obţinere a unei expresii fluxnomiale asociate unui graf şi invers, dându-se o expresie fluxnomială, se poate genera graful asociat ei.
Pentru modelarea unei clase, concentrăm într-o singură expresie fluxnomială mixtă informaţia asociată ei, juxtapunând aditiv metodele şi atributele, iar apoi aplicând un feedback multiplicativ. Un obiect aparţinând unei clase va fi o valoare memorată în conexiunea feedback a clasei sale.

A treia parte descrie o implementare în Java a expresiilor fluxnomiale. Aplicatia permite introducerea unei expresii fluxnomiale, normalizarea ei, cu vizualizarea formei sale normale şi posibilitatea testării dacă două expresii sunt echivalente. Acest proiect a fost realizat împreună cu Cornelia Oancea, studentă la aceeaşi secţie.

Cuprins

1 Introducere în teoria fluxnoamelor

Termenul de **reţea** este foarte răspândit în domeniul informaticii teoretice. În sensul general al cuvântului, o reţea este o structură ce conţine o mulţime de entităţi aflate într-o anumită relaţie. Reţelele sunt cele mai potrivite pentru a căpăta o intuiţie despre diferitele aspecte ale sistemelor informatice sau reale. De aceea, o bună formalizare matematică a noţiunii de reţea este foarte necesară. Algebra reţelelor (*Network Algebra*) este un instrument matematic care oferă un cadru teoretic cu o bună axiomatiare pentru modelarea diferitelor tipuri de reţele.

1.1 Noţiuni de bază

Calculul expresiilor regulate ale lui Kleene şi algebra regulată asociată descriu într-un mod elegant procese ciclice cu stări globale. Acest calcul are o bază teoretică solidă şi multe aplicaţii la programe secvenţiale, automate, limbaje formale, circuite etc., dar puţine în calculul paralel sau distribuit. Acest calcul are o sintaxă şi o semantică simple. Expresiile regulate peste o mulţime de simboluri atomice X, notate $RegExp(X)$, sunt date de următoarea gramatică:

$$E ::= E + E \mid E \cdot E \mid E^* \mid 0 \mid 1 \mid x (\in X)$$

În semantica standard, expresiile regulate sunt interpretate ca mulţimi de şiruri. Operaţiile sunt interpretate ca 'reuniune', 'concatenare' şi 'concatenare repetată', iar constantele ca 'mulţimea vidă' şi 'mulţimea cu unic element şirul de lungime 0'.

În această parte vom prezenta **'calculul fluxnoamelor'** (în engleză: *flownomial calculus*) care poate fi văzut ca o extensie a calculului de mai sus, capabil să descrie procese cu intrări şi ieşiri multiple.

Calculul fluxnomialelor este un calcul abstract pentru reţele (modelate ca hipergrafuri direcţionate etichetate) şi comportamentele lor. Acest calcul conţine două familii de elemente dublu indexate:

- o familie de variabile $X = \{X(a, b)\}_{a,b \in M}$ (aceste variabile reprezintă 'cutii negre', adică elemente atomice cu două tipuri de interfeţe de conectare, interfaţa de intrare descrisă de a şi interfaţa de ieşire descrisă de b

- o structură abstractă $T = \{T(a,b)\}_{a,b\in M}$. Elementele acestei teorii abstracte doresc a modela mesaje ce se transmit via canale de comunicare între interfaţa de intrare a şi interfaţa de ieşire b. Aceasta este partea 'cunoscută' (sau 'interpretată') a modelului, adică operaţiile sunt deja definite pe această parte.

În cele de mai sus, $M = (M, \star, \varepsilon)$ este un monoid ce modelează interfeţele celulelor reţelei văzute ca şiruri de porturi[1].

Sintaxa calculului cu fluxnoame este dată de următoarele constante şi operaţii:

$$E ::= E \star E \mid E \cdot E \mid E \uparrow^c \mid \mathsf{I}_a \mid {}^a\mathsf{X}^b \mid \wedge_m^a \mid \vee_a^m \mid x(\in X) \mid f(\in T)$$

unde $a, b, c \in M$ şi $m, n \in \mathbf{N}$. Rolul acestora va fi explicat mai jos.

Observaţia 1.1.1 *Calculul cu expresii regulate este un caz particular al calculului cu fluxnoame: alegem* $M = (\mathbf{N}, +, 0)$ *şi* $0 := \wedge_0^1 \cdot \vee_1^0$, $1 := \mathsf{I}_1$, $f + g := \wedge_2^1 \cdot (f \star g) \cdot \vee_1^2$, $f \cdot g := f \cdot g$, $f^* := [\vee_1^2 \cdot (1 + f) \cdot \wedge_2^1] \uparrow^1$.

Obţinem astfel **mulţimea expresiilor fluxnomiale peste** X **şi** T (în engleză: *flownomial expressions over* X *and* T), notată cu $FlowExp(X,T)$, construită pornind de la elementele din X şi T şi aplicând cele trei operaţii: *compunere monoidală* '$\star$', *compunere* '$\cdot$' şi *feedback* '$\uparrow^c$', pentru $c \in M$.

Avem funcţiile $i : FlowExp(X,T) \longrightarrow M$ şi $o : FlowExp(X,T) \longrightarrow M$ pentru a extrage şirurile intrărilor şi al ieşirilor unei reţea specificată de o expresie fluxnomială.

Compunerea monoidală $f \star g$ este întotdeauna definită şi $i(f \star g) = i(f) \star i(g)$ şi $o(f \star g) = o(f) \star o(g)$. Compunerea $f \cdot g$ este definită doar în cazul $o(f) = i(g)$ şi în acest caz, $i(f \cdot g) = i(f)$ şi $o(f \cdot g) = o(g)$. Feedbackul $f \uparrow^c$ este definit doar în cazul $i(f) = a \star c$ şi $o(f) = b \star c$, pentru anumiţi $a, b \in M$ şi M satisface regula: $a \star c = a' \star c \Rightarrow a = a'$.

[1] în informatică, termenul de *port* este folosit cu sensul de 'dispozitiv care asigură comunicarea între un calculator şi un periferic (terminal, imprimantă, etc.)'

Mulţimea T va conţine anumite clase de relaţii binare finite. Câteva exemple de astfel de clase sunt funcţiile finite, funcţiile finite parţial definite şi relaţiile finite arbitrare. La nivel abstract, aceasta presupune că va trebui să avem anumite constante pentru modelarea/axiomatizarea acestor relaţii. Mulţimea constantelor folosite în acest scop vor fi: *ramificarea* $\wedge_m^a$, *transpoziţia* ${}^a\mathsf{X}^b$ şi *identificarea* $\vee_a^n$, unde $m, n \in \mathbf{N}$ şi $a, b \in M$. Fiecare dintre clasele date ca exemplu mai sus (şi alte 13) pot fi generate folosind doar anumite subfamilii ale acestor constante. Aceste subfamilii se obţin punând restricţii asupra constantelor de ramificare $\wedge_m$ şi $\vee^n$. Restricţia $x = a$ selectează doar constantele de tip $\wedge_m^a$ cu $m = 1$; restricţia $x = b$ forţează $m \leq 1$; $x = c$ alege $m \geq 1$; iar $x = d$ nu impune nici o restricţie (m este arbitrar). Restricţia y este definită într-un mod asemănător, având de-a face de data aceasta cu n de la constantele de tip $\vee_a^n$ şi literele greceşti. De exemplu, $y = \beta$ impune $n \leq 1$.

Notaţia xy, unde $x \in \{a, b, c, d\}$ şi $y \in \{\alpha, \beta, \gamma, \delta\}$ se referă la clasa relaţiilor generate folosind transpoziţii şi constante de ramificare satisfăcând restricţia xy. De exemplu, aplicând restricţia $a\delta$ obţinem funcţiile (finite), aplicând restricţia $b\delta$, funcţiile parţiale, iar $d\delta$ generează relaţiile.

1.1.1 Algebra relaţiilor finite

Algebra polinoamelor se bazează pe o structură algebrică de *inel* peste numere $((R, +, \cdot, 0, 1)$ cu $R = \mathbf{Z}, \mathbf{R} \ldots)$. Într-un mod asemănător, *algebra fluxnoamelor* se bazează pe *algebra reţelelor* peste relaţii. Să 'algebrizăm' relaţiile finite şi sortate.

Fie S o mulţime de sorturi. Elementele din monoidul liber $(S^\star, \star, \varepsilon)$ le vom nota cu $a = a_1 \star \ldots \star a_{|a|}$, unde $|a|$ este lungimea lui a şi $a_i \in S$, pentru orice $i \in [|a|]$.

Definiţia 1.1.2 (**Relaţie finită** S**-sortată**) Pentru S o mulţime de sorturi şi $a, b \in S^\star$, notăm:

$$Rel_S(a, b) = \{f \subseteq [|a|] \times [|b|] \mid (i, j) \in f \Rightarrow a_i = b_j\}.$$

Un element din $Rel_S(a, b)$ se numeşte **relaţie finită** S**-sortată** şi se notează cu $f : a \to b$ sau $a \xrightarrow{f} b$.

Rolul sorturilor devine clar atunci când privim o relaţie sortată ca un mod de a conecta două interfeţe *in* şi *out*. O interfaţă $a \in S^\star$ este o mulţime ordonată de canale $\{a_1, \ldots, a_{|a|}\}$ prin care pot intra/ieşi date de un anumit tip (specificat de sortul corespunzător a_i). Condiţia din definiţia relaţiei sortate impune restricţia ca relaţia să conecteze doar canale de acelaşi tip (sort), aceasta fiind un lucru natural.

Să înzestrăm relaţiile finite sortate (pentru S fixat) cu o structură de algebră a reţelelor. **Operaţiile** de *juxtapunere* $\star$, *compunere* $\cdot$ şi *feedback* $\uparrow$ sunt definite astfel:

- pentru $a \xrightarrow{f} b$ şi $c \xrightarrow{g} d$, alegem $a \star c \xrightarrow{f \star g} b \star d$

$$f \star g = f \cup \{\ (|a| + i, |b| + j)\ |\ (i, j) \in g\ \}$$

- pentru $a \xrightarrow{f} b \xrightarrow{g} c$, alegem $a \xrightarrow{f \cdot g} c$

$$f \cdot g = \{\ (i, k)\ |\ \exists j \in [|b|] \text{ a.î. } (i, j) \in f \text{ şi } (j, k) \in g\ \}$$

- pentru $s \in S$ şi $a \star s \xrightarrow{f} b \star s$, alegem $a \xrightarrow{f \uparrow^s} b$

  $f \uparrow^s = \{\ (i, j)\ |\ i \in [|a|],\ j \in [|b|]$ şi

$$[sau\ (i, j) \in f\ sau\ ((i, |b| + 1) \in f \text{ şi } (|a| + 1, j) \in f)]\ \}$$

  Această operaţie este extinsă la $\uparrow^a$ pentru $a \in S^\star$ prin

$$f \uparrow^\varepsilon = f, \quad f \uparrow^{a \star b} = (f \uparrow^b) \uparrow^a .$$

Constantele unei algebre a reţelelor sunt: I (*identitatea*), X (*transpoziţia*), $\vee$ şi $\top$ (*constante de identificare* de gradul 2, respectiv 0), $\wedge$ şi $\perp$ (*constante de ramificare* de gradul 2, respectiv 0). Ele acţionează asupra blocurilor şi au în *Rel* următoarea semnificaţie:

- $a \xrightarrow{\mathsf{I}_a} a$ cu $\mathsf{I}_a = \{\ (i, i)\ |\ i \in [|a|]\ \}$
- $a \star b \xrightarrow{{}^a\mathsf{X}^b} b \star a$ cu ${}^a\mathsf{X}^b = \{\ (i, |b| + i)\ |\ i \in [|a|]\} \cup \{(|a| + i, i)\ |\ i \in [|b|]\ \}$

- $a \star a \xrightarrow{\vee_a} a$ cu $\vee_a = \{\ (i,i) \mid i \in [|a|]\ \} \cup \{\ (|a|+i, i) \mid i \in [|a|]\ \}$
- $\varepsilon \xrightarrow{\top_a} a$ cu $\top_a = \emptyset$
- $a \xrightarrow{\wedge^a} a \star a$ cu $\wedge^a = \{\ (i,i) \mid i \in [|a|]\ \} \cup \{\ (i, |a|+i) \mid i \in [|a|]\ \}$
- $a \xrightarrow{\perp^a} \varepsilon$ cu $\perp^a = \emptyset$

În anexa A este listarea clasei Java din proiectul dizertaţiei care implementează pe calculator algebra relaţiilor finite sortate.

1.1.2 Nucleul BNA

Lăsând deocamdată la o parte constantele de ramificare, să prezentăm nucleul axiomelor algebrei reţelelor.

Algebra de bază a reţelelor (*în engleză*: Basic Network Algebra), prescurtat BNA, este formată dintr-o signatură BNA: $\star, \cdot, \uparrow \mathsf{I}, \mathsf{X}$ şi următoarele *axiome* BNA:

Tabelul 1: Axiomele BNA

I	Axiome pentru symocat-uri	II	Axiome pentru feedback
B1	$f \star (g \star h) = (f \star g) \star h$	R1	$f \cdot (g \uparrow^c) \cdot h = ((f \star \mathsf{I}_c) \cdot g \cdot (h \star \mathsf{I}_c)) \uparrow^c$
B2	$\mathsf{I}_\varepsilon \star f = f = f \star \mathsf{I}_\varepsilon$	R2	$f \star g \uparrow^c = (f \star g) \uparrow^c$
B3	$f \cdot (g \cdot h) = (f \cdot g) \cdot h$	R3	$(f \cdot (\mathsf{I}_b \star g)) \uparrow^c = ((\mathsf{I}_a \star g) \cdot f) \uparrow^d$
B4	$\mathsf{I}_a \cdot f = f = f \cdot \mathsf{I}_b$		pentru $f : a \star c \to b \star d,\ g : d \to c$
B5	$(f \star f') \cdot (g \star g') = f \cdot g \star f' \cdot g'$	R4	$(f \uparrow^b) \uparrow^a = f \uparrow^{a \star b}$
B6	$\mathsf{I}_a \star \mathsf{I}_b = \mathsf{I}_{a \star b}$	R5	$\mathsf{I}_a \uparrow^a = \mathsf{I}_\varepsilon$
B7	${}^a\mathsf{X}^b \cdot {}^b\mathsf{X}^a = \mathsf{I}_{a \star b}$	R6	${}^a\mathsf{X}^a \uparrow^a = \mathsf{I}_a$
B8	${}^a\mathsf{X}^{b \star c} = ({}^a\mathsf{X}^b \star \mathsf{I}_c) \cdot (\mathsf{I}_b \star {}^a\mathsf{X}^c)$		
B9	$(f \star g) \cdot {}^c\mathsf{X}^d = {}^a\mathsf{X}^b \cdot (g \star f)$		
	pentru $f : a \to c,\ g : b \to d$		

În continuare, un monoid arbitrar $(M, \star, \varepsilon)$ va fi folosit pentru a specifica interfeţele de intrare/ieşire. Să dăm definiţia unui $a\alpha$-flux:

Definiţia 1.1.3 (**BNA/$a\alpha$-flux**) Un **BNA** sau un **$a\alpha$-flux peste un monoid** M este o structură abstractă $(T, \star, \cdot, \uparrow, \mathsf{I}_a, {}^a\mathsf{X}^b)$ dată de:

- o familie de săgeţi $T = \{T(a,b)\}_{a,b \in M}$;
- constante: *identităţi* $\mathsf{I}_a \in T(a,a)$ şi *transpoziţii* ${}^a\mathsf{X}^b \in T(a \star b, b \star a)$;
- operaţii: *juxtapunere* $\star : T(a,b) \times T(c,d) \to T(a \star c, b \star d)$, *compunere* $\cdot : T(a,b) \times T(b,c) \to T(a,c)$ şi *feedback* $\uparrow^c$: $T(a \star c, b \star c) \to T(a,b)$.

care verifică axiomele din Tabelul 1, $(a,b,c \in M)$.

Definiţia 1.1.4 (**Morfism de $a\alpha$-fluxuri**) Dându-se un $M - a\alpha$-flux T şi un $M' - a\alpha$-flux T', un **morfism de $a\alpha$-fluxuri** $H : T \to T'$ este dat de un morfism de monoizi $h : M \to M'$ şi o familie de aplicaţii dublu-indexate după $m, n \in M$, $H_{m,n} : T(m,n) \to T'(h(m), h(n))$ care respectă operaţiile:

$$H(f \star g) = H(f) \star H(g), \quad H(f \cdot g) = H(f) \cdot H(g), \quad H(f \uparrow^p) = H(f) \uparrow^{h(p)},$$
$$H(\mathsf{I}_a) = \mathsf{I}_{h(a)}, \quad H({}^m\mathsf{X}^n) = {}^{h(m)}\mathsf{X}^{h(n)}.$$

Luând BNA-urile ca obiecte şi morfismele mai sus definite, obţinem o categorie.

1.1.3 Expresii fluxnomiale

Până acum am definit operaţii şi constante pentru relaţiile din Rel_S. Pentru a termina construcţia calculului fluxnomial, mai avem nevoie şi de o familie X de mulţimi dublu-indexate, $X = \{X(a,b)\}_{a,b \in S^\star}$. Un element $x \in X(a,b)$ reprezintă o celulă *variabilă* din reţea, având $|a|$ canale de intrare de sorturi $a_1, \ldots, a_{|a|}$ şi $|b|$ canale de ieşire de sorturi $b_1, \ldots, b_{|b|}$.

Definiţia 1.1.5 (**Expresii fluxnomiale peste** T) Fie T un BNA peste un monoid $(S^\star, \star, \varepsilon)$, de exemplu Rel_S. Pentru $a, b \in S^\star$, mulţimile **expresiilor fluxnomiale de tip** $a \to b$ **peste** X **şi** T, notate $\mathbf{FlowExp}[X,T](a,b)$ sunt definite astfel:

1. expresii atomice: o variabilă x din $X(a,b)$ sau un simbol care reprezintă o conexiune f din $T(a,b)$ este o expresie fluxnomială de tip $a \to b$;

2. expresii compuse: dacă $E_1 : a \to b$, $E_2 : c \to d$, $E_3 : b \to c$ şi $E : a \star s \to b \star s$ (cu $a, b, c \in S^\star$, $s \in S$) sunt expresii fluxnomiale de tipul indicat, atunci

$$\begin{array}{l} (E_1 \star E_2) : a \star c \to b \star d \\ (E_1 \cdot E_3) : a \to c \\ (E \uparrow^s) : a \to b \end{array}$$

sunt expresii fluxnomiale de tipul specificat;

3. toate expresiile fluxnomiale se obţin aplicând de un număr finit de ori regulile 1 şi 2.

Uneori, pentru uşurinţa exprimării, expresiile fluxnomiale le vom numi într-un cuvânt: **fluxnoame**[2].

Pentru simplitate, vom stabili priorităţi între operatori şi vom putea folosi doar parantezele strict necesare în scrierea expresiei. Astfel, feedback-ul 'leagă' cel mai puternic, el fiind urmat de compunere şi în ultimul rând de juxtapunere. Juxtapunerea şi compunerea sunt presupuse asociative. De exemplu:

$$((x \star x) \star ((x \star y) \cdot ((f \uparrow^s) \uparrow^t))) \text{ se scrie } x \star x \star (x \star y) f \uparrow^{t \star s} .$$

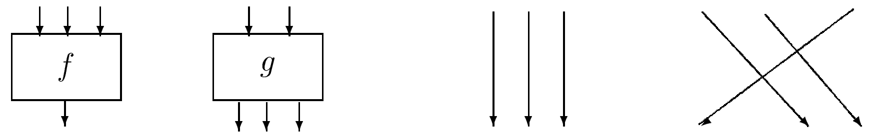
Acest exemplu arată de asemenea că $\uparrow^a$ înseamnă $\uparrow^{a_{|a|}} \ldots \uparrow^{a_1}$ şi că $\cdot$ poate fi omis.

Fiecărui fluxnom din $\mathbf{FlowExp}[X, T]$ cu $T \subseteq \mathbf{Rel}$ i se poate asocia o reprezentare grafică. Atomii din X vor fi reprezentaţi prin celule cu interfeţe date de aplicaţiile i şi o, iar relaţiile ca simple conexiuni.

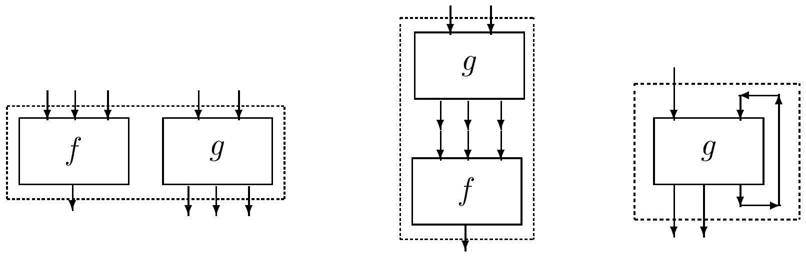
Operaţiile capătă şi ele o reprezentare grafică naturală. Se observă faptul că juxtapunerea 'pune' reprezentările una lângă alta, compunerea uneşte ieşirile reprezentării primei expresii cu intrările celei de a doua, iar feedback-ul uneşte ieşirile cu intrările reprezentării fluxnomului asupra căruia este aplicat feedback-ul.

[2] termenul *fluxnom* poate să însemne şi reprezentantul unei clase de expresii fluxnomiale echivalente

Iată câteva exemple de reprezentări (într-un cadru monosortat):



$f:3\longrightarrow 1$ $\quad g:2\longrightarrow 3$ $\quad \mathsf{I}_3:3\longrightarrow 3$ $\quad {}^2\mathsf{X}^1:3\longrightarrow 3$



$f\star g:5\longrightarrow 4$ $\quad g\cdot f:2\longrightarrow 1$ $\quad g\uparrow^1:1\longrightarrow 2$

În secţiunea 2.1 sunt prezentaţi algoritmi formalizaţi care fac trecerea de la fluxnoame la reprezentarea lor grafică şi invers.

Numele cadrului calculului fluxnomial, acela de *algebră a reţelelor* este perfect justificat. Setul de axiome BNA reprezintă un calcul pentru reţele. Cu alte cuvinte, reţelele obişnuite pot fi identificate cu fluxnoame modulo axiomele BNA. Aceasta afirmaţie se bazează pe următoarele două observaţii:

- Reţelele concrete nedeterministe construite cu atomi din X coincid cu reprezentarea grafică a fluxnoamelor din $\mathbf{FlowExp}[X,\mathbf{Rel}]$. Alte clase de reţele pot fi reprezentate prin fluxnoame alegând ca T să fie egal cu diferite submulţimi ale lui **Rel**, de exemplu **Pfn** pentru reţele deterministe, parţial definite, **Fn** pentru reţele deterministe, **Bi** pentru reţele cu conexiuni bijective (fără constante de ramificare), etc.

- Fie $T\subseteq\mathbf{Rel}$ o substructură a lui **Rel** închisă la operaţiile din BNA, de exemplu $T=\mathbf{Pfn}$ sau $T=\mathbf{Rel}$. Următoarele afirmaţii sunt adevărate: (i) (*corectitudine*) Două fluxnoame peste T care sunt echivalente via axiomele BNA au reprezentări grafice izomorfe; şi (ii) (*completitudine*) Două fluxnoame peste T care au reprezentări grafice izomorfe sunt egale via regulile BNA.

Pentru demonstraţii şi mai multe detalii, a se vedea [Ste00].

1.2 Forma normală

Folosind axiomele BNA, orice expresie fluxnomială poate fi adusă la o formă normală care va grupa variabilele şi conexiunile dintre ele.

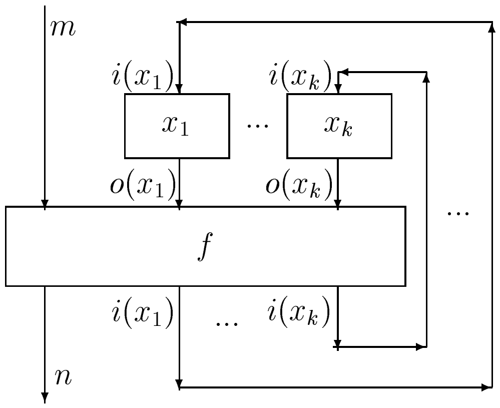
Fie T un BNA. O expresie fluxnomială de forma:

$$((\mathsf{I}_m \star x_1 \star \ldots \star x_k) \cdot f) \uparrow^{i(x_1)\star\ldots\star i(x_k)}$$

cu $x : i(x) \longrightarrow o(x)$ şi $f \in T(m \star o(x_1) \star \ldots \star o(x_k), n \star i(x_1) \star \ldots \star i(x_k))$ se numeşte **formă normală** şi se mai scrie şi sub formă de pereche

$$(x_1 \star \ldots \star x_k, f) \in [X, T](m, n) \text{ sau } (x, f) \text{ cu } x \in X^\star.$$

Grafic, această formă normală arată astfel:



O expresie fluxnomială se construieşte recursiv aplicând paşii prezentaţi în definiţia 1.1.5. Pentru a construi forma ei normală, vom urma prin inducţie structurală aceşti paşi. Să descriem o aplicaţie $nf(\_)$ care duce o expresie fluxnomială în forma ei normală.

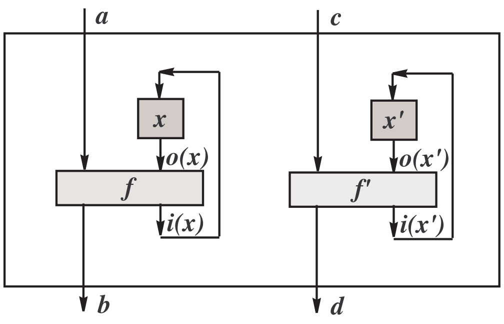
În primul rând, pentru elementele de bază avem:

- Pentru $x \in X(m, n)$, $nf(x) \stackrel{def}{=} (x, {}^m\mathsf{X}^n)$.
- Pentru $f \in T(m, n)$, $nf(f) \stackrel{def}{=} (\varepsilon, f)$.

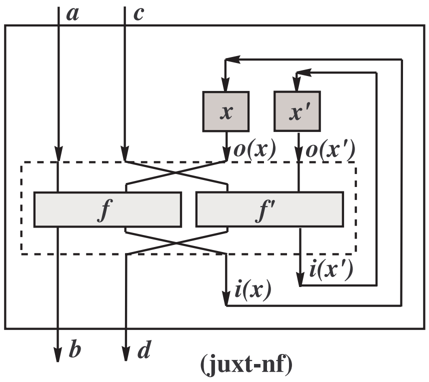
Să vedem acum care este forma normală a aplicării operaţiilor asupra unor expresii aduse la forma normală (prin inducţia structurală).

- pentru $(x, f) \in [X, T](a, b)$ şi $(x', f') \in [X, T](c, d)$, avem operaţia de juxtapunere:

$$nf((x, f)\star(x', f')) \overset{def}{=} (x\star x', (\mathsf{I}_a\star{}^{c}\mathsf{X}^{o(x)}\star\mathsf{I}_{o(x')})\cdot(f\star f')\cdot(\mathsf{I}_b\star{}^{i(x)}\mathsf{X}^{d}\star\mathsf{I}_{i(x')}))$$



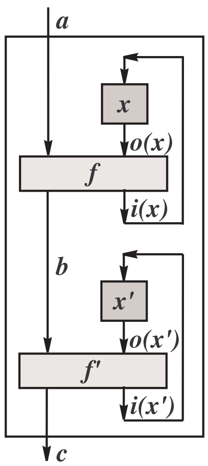
(juxt)



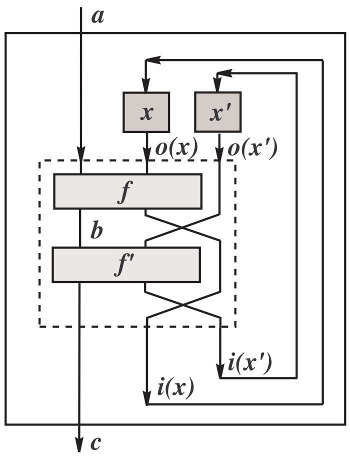
(juxt-nf)

- pentru $(x, f) \in [X, T](a, b)$ şi $(x', f') \in [X, T](b, c)$, avem operaţia de compunere:

$$nf((x, f)\cdot(x', f')) \overset{def}{=} (x\star x', (f\star\mathsf{I}_{o(x')})\cdot(\mathsf{I}_b\star{}^{i(x)}\mathsf{X}^{o(x')})\cdot(f'\star\mathsf{I}_{i(x)})\cdot(\mathsf{I}_c\star{}^{i(x')}\mathsf{X}^{i(x)}))$$



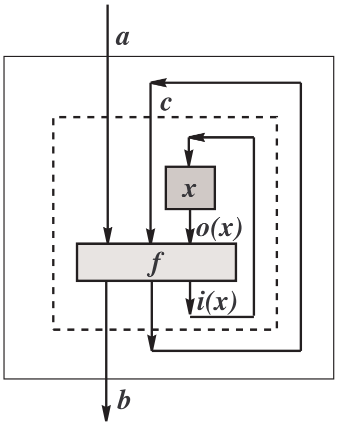
(comp)



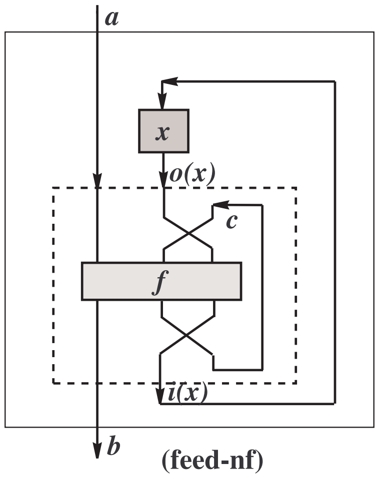
(comp-nf)

- pentru $(x,f) \in [X,T](a\star c, b\star c)$ avem operaţia de feedback:

$$nf((x,f)\uparrow^p) \stackrel{def}{=} (\, x, (\mathsf{I}_a \star {}^{o(x)}\mathsf{X}^c)\cdot f \cdot (\mathsf{I}_b \star {}^c\mathsf{X}^{i(x)})\,)$$



(feed)



(feed-nf)

În implementarea prezentată în secţiunea 3, pentru normalizarea unui fluxnom a fost folosită construcţia de mai sus.

1.3 Simulare via bijecţii

Presupunem că T este un BNA.

Definiţia 1.3.1 (**Simulare**) Două forme normale $F = (x_1 \star \ldots \star x_m, f)$ şi $G = (y_1 \star \ldots \star y_n, g)$ sunt **similare** via o relaţie $u : [m] \longrightarrow [n]$ (şi vom nota aceasta: $F \longrightarrow_u G$) dacă şi numai dacă

1. $(i,j) \in u \;\; \Rightarrow \;\; x_i = y_j$
2. $f \cdot (\mathsf{I}_b \star i(u)) = (\mathsf{I}_a \star o(u)) \cdot g$,

unde $i(u)$ reprezintă extensia lui u la intrări şi $o(u)$ reprezintă extensia lui u la ieşiri.

Să explicăm cum se obţine (de exemplu) extensia lui u la intrări $i(u)$. Dacă u pune în relaţie două variabile x_i şi y_j, atunci $i(u)$ va pune în relaţie prima intrare a lui x_i cu prima intrare a lui y_j, a doua intrare a lui x_i cu a doua intrare a lui y_j şi aşa mai departe. Dacă u este bijecţie, atât $i(u)$, cât şi $o(u)$ vor fi la rândul lor bijecţii, însă cu domeniile şi codomeniile date de intrări, respectiv ieşiri.

Definiţia 1.3.2 (**Simulare via bijecţii**) Două forme normale $F = (x_1 \star \ldots \star x_m, f)$ şi $G = (y_1 \star \ldots \star y_n, g)$ sunt **similare via o bijecţie** dacă şi numai dacă $m = n$ şi există o bijecţie $u : [m] \longrightarrow [m]$ a.î. $F \longrightarrow_u G$

Prima condiţie de similaritate ne spune că u pune în relaţie celule egale, în timp ce cea de-a doua condiţie de similaritate ne arată că, *abstracţie făcând de permutarea* u, conexiunile celor două forme normale sunt aceleaşi.

Teorema 1.3.3 *Două forme normale reprezintă aceeaşi reţea dacă şi numai dacă ele sunt similare via o bijecţie.*

1.4 Modele ale calculului fluxnomial

Există trei modele importante ce satisfac setul de axiome BNA şi diferite fragmente din setul NA: aditiv, multiplicativ şi temporal. Există diferite variante de mixare a acestora. Noi vom prezenta în subsecţiunile următoare doar modelul aditiv AddNA, cel multiplicativ MultNA şi modelul mixt MixNA, care este o combinaţie aditiv-multiplicativă.

1.4.1 Algebra reţelelor aditivă: AddNA

Pentru a evita confuzia, vom folosi următorul set de simboluri [3]:

$$\oplus \quad \cdot_{\oplus} \quad \uparrow_{\oplus} \quad - \quad \between \quad \bullet\!\!<_k \quad {}_k\!\!>\!\!\bullet$$

pentru operaţiile şi constantele fluxnomiale, în cazul în care se doreşte o interpretare aditivă. În cazurile elementare $k = 0$ şi $k = 2$ a constantelor de ramificare $\bullet\!\!<_k$ (resp. ${}_k\!\!>\!\!\bullet$) vom folosi următoarea notaţie alternativă:

$$\perp \quad \wedge \quad (\text{respectiv} \quad \top \quad \vee)$$

Interpretarea aditivă a operaţiilor şi a constantelor este dată în modelele semantice *AddRel*(D). Acest model este prezentat detaliat în [Ste00][4]. Un rezultat fundamental de corectitudine:

> *Toate identităţile din Tabelul 2 sunt valide în modelele semantice AddRel*(D). [5]

Să amintim pe scurt semnificaţia operaţiilor algebrei NA (*Network Algebra*) în *AddRel*(D). Presupunem că D este o mulţime de vectori-valoare ce reprezintă stările memoriei unui sistem de calcul. Dacă avem dată o interpretare a variablelor din X, atunci obţinem o interpretare a fluxnomului

[3] simbolurile pentru constante au fost alese astfel încât să se potrivească cu desenarea orizontală şi de la stânga la dreapta a reţelelor aditive (de control) şi cu desenarea verticală şi de sus în jos a reţelelor multiplicative (reactive)

[4] notaţia folosită în [Ste00] pentru *AddRel*(D) este $\mathbb{R}$el

[5] corespondenţa dintre notaţia AddNA şi cea din Tabelul 2 este următoarea: $\star = \oplus$, $\cdot = \cdot_{\oplus}$, $\uparrow = \uparrow_{\oplus}$, $\perp = \bullet\!\!<_0$, $\wedge = \bullet\!\!<_2$, $\top = {}_0\!\!>\!\!\bullet$, $\vee = {}_2\!\!>\!\!\bullet$

$E : m \to n$ peste X ca fiind o relaţie $E_I \subseteq ([m] \times D) \times ([n] \times D)$, cu semnificaţia: $((j, d), (j', d')) \in E_I$ dacă şi numai dacă "dacă este rulat programul obţinut din expresia E şi interpretarea I, programul începând pe intrarea j şi având starea iniţială a memoriei d, atunci programul *eventual* se termină pe ieşirea j' având starea memoriei d' ".

Această observaţie ne conduce la construcţia unui model $AddRel(D)$ pentru algebra AddNA în cazul multisortat, după cum urmează.

Fie S o mulţime de sorturi atomice şi fie $D = \{D_s\}_{s\in S}$ o mulţime S-sortată de date. Tipurile interfeţelor programelor în cazul aditiv sunt modelate prin elemente ale monoidului aditiv $S^{\oplus} = (S^{\oplus}, \oplus, 0)$ liber generat de S. Deci un sort general pentru o interfaţă aditivă este reprezentat printr-un cuvânt $a = a_1 \oplus \ldots \oplus a_{|a|}$, unde a_i este a i-a literă a lui a.

Oricărui sort a i se poate asocia într-un mod natural un domeniu semantic pentru o dată de tip a după cum urmează. Dacă $a \in S^{\oplus}$ este ca mai sus, vom nota cu D_a suma **disjunctă** $(D_{a_1} + \ldots + D_{a_{|a|}})$. Aici, '+' este reuniunea disjunctă, ordonată pe mulţimi. De exemplu, acest '+' poate fi definit prin $A + B = (\{1\} \times A) \cup (\{2\} \times B)$. Astfel, avem:

$$D_a = (\{1\} \times D_{a_1}) \cup \ldots \cup (\{| a |\} \times D_{a_{|a|}}).$$

Un element generic în această ultimă mulţime va fi notat prin

$$\langle x_1, \ldots, x_n \rangle,$$

cu semnificaţia că: x este 'unul dintre' elementele tuplului; mai precis, x este un x_i şi ştim numărul componentei corespondente. Pentru aceasta folosim notaţia separatorului punct: x este $i.x_i$

Suma disjunctă este *asociativă* modulo un izomorfism, dar operaţia $\langle \ldots \rangle$ o considerăm *necomutativă*.

În sfârşit, notăm prin $AddRel(D)$ structura relaţională 'aditivă' definită prin

$$AddRel(D)(a, b) = \{f \mid f \subseteq D_a \times D_b\}$$

pentru $a, b \in S^{\oplus}$.

Operaţiile care ne interesează au următoarele semnificaţii în $AddRel(D)$:

- **Sumare:** pentru $f : a \to b$ şi $g : c \to d$, suma $f \oplus g : a \oplus c \to b \oplus d$ este definită prin
$$\begin{array}{lll} f \oplus g = \{ \ (x, y) \ | & ori\,(x = 1.x_1,\ y = 1.y_1 \text{ şi } (x_1, y_1) \in f) \\ & ori\,(x = 2.x_2,\ y = 2.y_2 \text{ şi } (x_2, y_2) \in g) \ \} \end{array}$$

- **Compunere:** pentru $f : a \to b$ şi $g : b \to c$, compunerea $f \cdot_\oplus g : a \to c$ este cea uzuală (numită şi *angelică*) definită prin
$$f \cdot_\oplus g = \{ \ (x, z) \mid \exists y \in D_b \,\text{astfel încât}\, (x, y) \in f \,\text{şi}\, (y, z) \in g \ \}$$

- **Feedback:** Pentru a defini feedback-ul, observăm că o relaţie $f : a \to b$ este dată de o familie de relaţii $\{f_{i,j}\}_{i \in [|a|],\ j \in [|b|]}$, unde
$$f_{i,j} = \{ \ (d, d') \mid ((i.d, j.d'\,)) \in f \ \} \subseteq D_{a_i} \times D_{b_j}.$$
Ca de obicei, notăm prin r^* închiderea reflexiv-tranzitivă a relaţie r.
Pentru $f : a \oplus s \to b \oplus s$, feedback-ul $f \uparrow^s_\oplus : a \to b$ este definit prin
$$(f \uparrow^s_\oplus)_{i,j} = f_{i,j} \ \cup \ f_{i,|b \oplus s|} \cdot f^*_{|a \oplus s|,|b \oplus s|} \cdot f_{|a \oplus s|,j}$$
pentru $i \in [|\ a\ |],\ j \in [|\ b\ |]$. Definiţia se extinde în mod natural la cuvinte arbitrare prin $f \uparrow^0_\oplus = f, \quad f \uparrow^{a \oplus s}_\oplus = (f \uparrow^s_\oplus) \uparrow^a_\oplus$.

- **Identitate:** $-_a : a \to a$ se defineşte prin:
$$-_a = \{ \ (x, x) \mid x \in D_a \ \}.$$

- **Transpoziţie aditivă:** ${}^b_a\!\!\times : a \oplus b \to b \oplus a$ se defineşte prin:
$$\begin{array}{lll} {}^b_a\!\!\times = \{ \ (x, y) \ | & ori\,(x = 1.x_1,\ y = 2.y_2 \text{ şi } x_1 = y_2 \\ & ori\,(x = 2.x_2,\ y = 1.y_1 \text{ şi } x_2 = y_1) \ \} \end{array}$$

- **Ramificare aditivă (Split):** ${}_a\!\!\bullet\!\!<_k : a \to ka$ se defineşte prin
$${}_a\!\!\bullet\!\!<_k = \{ \ (x, \langle x, \ldots, x \rangle) \mid x \in D_a \ \}.$$
Ramificările binare ($k = 2$) şi nule ($k = 0$) sunt cazuri particulare folositoare.

- **Identificare aditivă (Merge):** ${}_k{>}\!\!\bullet_a : ka \to a$ se defineşte prin:

  $${}_k{>}\!\!\bullet_a = \{\ (x,y) \mid \text{dacă } x \text{ este } i.x_i, \text{ atunci } y = x_i\ \}.$$

  Cazurile binare ($k=2$) şi nule ($k=0$) sunt cazuri particulare folositoare.

În continuare vom prezenta un exemplu care să demonstreze cum poate fi folosit acest model pentru a da semantica programelor secvenţiale descrise de reţele de flux (*flowchart*).

Exemplul 1.4.1 Să descriem un program dat de o reţea de flux pentru calculul coeficienţilor binomiali C_n^k (notaţi şi $\binom{k}{n}$):

Fie patru variabile n,k,y,z interpretate ca numere naturale. Fie mulţimea sorturilor $S=\{s_1,s_2,s_3,s_4\}$ şi mulţimile de date asociate sorturilor:

$$D_{s_1} = D_n \times D_k;\ D_{s_2} = D_n \times D_k \times D_y \times D_z;\ D_{s_3} = D_k \times D_y \times D_z;\ D_{s_4} = D_z$$

unde toate mulţimile D_n, D_k, D_y, D_z sunt egale cu $\mathbf{N}$. În final, alegem cinci celule

$$i_1 : s_1 \to s_2;\ t_1 : s_2 \to s_2 \oplus s_2;\ i_2 : s_2 \to s_3;\ t_2 : s_3 \to s_3 \oplus s_3;\ e : s_3 \to s_4$$

Semantica relaţională pentru aceste celule este:

$(n,k) \xrightarrow{i_1} (n,k,n,1)$;
$(n,k,y,z) \xrightarrow{t_1}$ dacă $y \le n-k$ atunci $1.(n,k,y,z)$ altfel $2.(n,k,y-1,z*y)$;
$(n,k,y,z) \xrightarrow{i_2} (k,1,z)$;
$(k,y,z) \xrightarrow{t_2}$ dacă $y \ge k$ atunci $1.(k,y,z)$ altfel $2.(k,y+1,z/y)$;
$(k,y,z) \xrightarrow{e} (z)$

Fie următoarea expresie fluxnomială în formă normală:

$$E\ =\ [(\mathsf{I}_{s_1} \oplus i_1 \oplus t_1 \oplus i_2 \oplus t_2 \oplus e)(2_{s_1}3_{s_2}4_{s_2}3_{s_2}5_{s_3}6_{s_3}1_{s_4})] \uparrow_{\oplus}^{s_1\oplus s_2\oplus s_2\oplus s_3\oplus s_3}$$

Avem că $|E|(n,k) = C_n^k$, pentru $k \le n$, unde $|E|$ este funcţia din $AddRel(D)$ asociată lui E, obţinută din interpretările de mai sus ale celulelor.

Aşa cum putem vedea din acest exemplu, celulele reţelei pot schimba spaţiul stărilor, deci spaţiul stărilor poate fi diferit în diferite puncte ale programului. Deşi acest spaţiu are o structură multiplicativă, acest lucru nu este semnificativ, semantica celulelor fiind *pur aditivă.* Aceasta se datorează faptului că acţiunile sunt *atomice*, adică nu conţin subacţiuni ce prelucrează în paralel fragmente din spaţiul stărilor.

1.4.2 Algebra reţelelor multiplicativă: MultNA

Vom folosi următorul set de simboluri:

$$\otimes \quad \cdot_{\otimes} \quad \uparrow_{\otimes} \quad \mathsf{I} \quad \mathsf{X} \quad \curlywedge_k \quad \curlyvee^k$$

pentru operaţiile şi constantele fluxnomiale, în cazul în care se doreşte o interpretare multiplicativă. În cazurile elementare $k = 0$ şi $k = 2$ a constantelor de ramificare $\curlywedge_k$ (resp. $\curlyvee^k$) vom folosi următoarea notaţie alternativă:

$$\curlywedge_0 \quad \curlywedge \quad \text{(respectiv} \quad \curlyvee^0 \quad \curlyvee)$$

În continuare vom descrie un model $MultRel(D)$ pentru transformatoare de date paralele. Un astfel de transformator $f : m \to n$ acţionează pe un m-tuplu de date de intrare şi produce un n-tuplu de date de ieşire. În acest model definim operatori de compunere paralelă şi secvenţială şi anumite constante: identitatea, transpoziţia, constante de ramificare (*copy* şi *sink*) şi constante de identificare (*source* şi *equality test*) În cele ce urmează vom defini feedback-ul folosind mulţimea tuturor invarianţilor posibili pentru canalele feedback. Aceste modele sunt prezentate în detaliu în [BeS94, BrS96, BeS95]. Reamintim că:

> *Pentru modelele $MultRel(D)$,* **nu** *toate axiomele din Tabelul 2 sunt valide, totuşi axiomele din nucleul BNA sunt valide.* [6]

În continuare vom folosi o extensie la cazul multisortat. Fie S o mulţime de sorturi atomice şi fie $D = \{D_s\}_{s\in S}$ o mulţime de date S-sortată. Tipurile

[6] corespondenţa dintre notaţia MultNA şi cea din Tabelul 2 este următoarea: $\star = \otimes$, $\cdot = \cdot_{\otimes}$, $\uparrow = \uparrow_{\otimes}$, $\perp = \curlywedge_0$, $\wedge = \curlywedge_2$, $\top = \curlyvee^0$, $\vee = \curlyvee^2$

interfeţelor programelor în cazul multiplicativ sunt modelate prin elemente ale monoidului multiplicativ $S^{\otimes} = (S^{\otimes}, \otimes, 1)$ liber generat de S. Deci un sort general pentru o interfaţă multiplicativă este reprezentat printr-un cuvânt $a = a_1 \otimes \ldots \otimes a_{|a|}$, unde a_i este a i-a literă a lui a.

Oricărui sort a i se poate asocia într-un mod natural un domeniu semantic pentru o dată de tip a după cum urmează. Dacă $a \in S^{\otimes}$ este ca mai sus, vom nota cu D_a produsul cartezian $(D_{a_1} \times \ldots \times D_{a_{|a|}})$. Un element generic în această mulţime va fi notat (ca de obicei) prin

$$(x_1, \ldots, x_n)$$

În sfârşit, notăm prin $MultRel(D)$ structura relaţională 'multiplicativă' definită prin

$$MultRel(D)(a, b) = \{f \mid f \subseteq D_a \times D_b\}$$

pentru $a, b \in S^{\otimes}$.

În $MultRel(D)$ constantele şi operaţiile sunt interpretate astfel:

- **Produs:** pentru $f : a \to b$ şi $g : c \to d$ produsul $f \otimes g : a \otimes c \to b \otimes d$ este definit prin

$$f \otimes g = \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in f \text{ şi } (x_2, y_2) \in g\}$$

- **Compunere:** pentru $f : a \to b$ şi $g : b \to c$ compunerea $f \cdot_{\otimes} g : a \to c$ este definită prin

$$f \cdot_{\otimes} g = \{\ (x, z) \mid \exists y \in D_b \text{ astfel încât } (x, y) \in f \text{ şi } (y, z) \in g\ \}$$

- **Identitatea:** $\mathsf{I}_a : a \to a$ se defineşte prin:

$$\mathsf{I}_a = \{\ (x, x) \mid x \in D_a\ \}.$$

- **Transpoziţia multiplicativă:** ${}^a\mathsf{X}^b : a \otimes b \to b \otimes a$ este definită prin

$${}^a\mathsf{X}^b = \{\ ((x, y), (y, x)) \mid x \in D_a \text{ şi } y \in D_b\ \}.$$

- **Ramificare multiplicativă (Copy):** $\curlywedge_k^a : a \to a^k$ se defineşte prin

  $$\curlywedge_k^a = \{\ (x,(x,\ldots,x)) \mid x \in D_a\ \}.$$

  Copierea binară $\curlywedge^a : a \to a \otimes a$ $(k=2)$ şi scufundarea (sink) $\perp^a : a \to 1$ $(k=0)$ sunt cazuri particulare folositoare.

- **Identificare multiplicativă (Equality test):** $\curlyvee_a^k : a^k \to a$ se defineşte prin:

  $$\curlyvee_a^k = \{\ ((x_1,\ldots,x_k),x) \mid \begin{array}{l} \text{dacă } x_1 = \ldots = x_n, \\ \text{atunci } y = x_1,\ \text{altfel } x = err_a\ \}. \end{array}$$

  (în această definiţie err_a reprezintă un element-eroare pe care îl presupunem existent în fiecare D_a)

  În cazul $k=0$, prin convenţie, rezultatul este err_a şi obţinem o 'sursă săracă' $\blacktriangle_a : 1 \to a$ definită prin $\{\ ((),err_a)\ \}$. De asemenea, putem folosi şi 'surse bogate', $\vartriangle_a : 1 \to a$, definite prin $\{\ ((),x) \mid x \in D_a\ \}$. [7] Un caz particular folositor este testul de egalitate binar $\curlyvee_a : a \otimes a \to a$ $(k=2)$.

O complicaţie în acest model este definirea operatorului de feedback. Mai jos dăm definiţia feedback-ului ca punct fix maximal. [8]

- **Feedback:** pentru $f : a \otimes s \to b \otimes s$ definim $f \uparrow_{\otimes}^{s} : a \to b$ ca punct fix maximal, prin

  $$f \uparrow_{\otimes}^{s} = \{\ (x,y) \mid \exists z \in D_s \text{ cu } ((x,z),(y,z)) \in f\ \}.$$

Cadrul multiplicativ se pretează la descrierea comportametelor circuitelor şi a reţelelor de flux de date (*dataflow networks*).

[7] tipul sărac sau bogat al sursei depinde de interpretarea care este dată operatorului de feedback. La noi, $\curlywedge^a \uparrow_{\otimes}^{a} = \vartriangle_a$, în timp ce pentru interpretarea feedback-ului cu cel mai mic punct fix avem $\curlywedge^a \uparrow_{\otimes}^{a} = \blacktriangle_a$

[8] pentru feedback-ul ca cel mai mic punct fix trebuie folosite relaţii extinse în timp, adică relaţii peste şiruri de date

1.4.3 Algebra reţelelor mixtă: MixNA

Acum putem să descriem un formalism mixt care să combine algebrele reţelelor bazate pe control şi respectiv flux de date. Matematic, aceasta înseamnă să combinăm algebrele de reţea aditive şi multiplicative (AddNA şi MultNA).

Dacă combinăm signaturile algebrelor AddNA şi MultNA, obţinem următoarea mulţime de operatori şi constante: $\oplus, \cdot_\oplus, \uparrow_\oplus, -, \times, \bullet\!\!<_k, {}_k\!\!>\!\!\bullet \otimes, \cdot_\otimes, \uparrow_\otimes, \mathsf{I}, \mathsf{X}, \curlywedge_k, \curlyvee^k$. Prima ipoteză este că ambele algebre de reţea împart aceeaşi structură categorială. Cum $\cdot_\oplus = \cdot_\otimes$ şi $- = \mathsf{I}$, vom nota compunerea şi identitatea prin $\cdot$ şi respectiv I. În plus, algebra mixtă MixNA va avea câteva constante noi $\delta_;$ şi $\rho_;$, care leagă cele două algebre de reţea. În concluzie, signatura algebrei mixte MixNA este dată de următoarele operaţii şi constante:

$$\oplus \quad \otimes \quad \cdot \quad \uparrow_\oplus \quad \uparrow_\otimes \quad \mathsf{I} \quad \delta_; \quad \rho_; \quad \times \quad \mathsf{X} \quad \bullet\!\!<_k \quad {}_k\!\!>\!\!\bullet \quad \curlywedge_k \quad \curlyvee^k.$$

Fie S o mulţime de sorturi atomice. Mulţimea $s^{\oplus,\otimes}$ de sorturi arbitare este obţinută prin următoarele reguli:

$$a = a \oplus a \mid a \otimes a \mid 0 \mid 1 \mid s(\in S).$$

Astfel, un sort poate fi gândit ca o mulţime finită de cuvinte peste S.

Un astfel de sort poate fi folosit să descrie înterfaţa de comunicare a reţelei. Pentru semantică, presupunem că se dă o mulţime D_s pentru fiecare sort atomic s, care modelează datele ce se transmit pe canalele de tip s. Extindem notaţia D_a la sorturi arbitrare prin regulile:

$$\begin{array}{rcl} D_0 & = & \emptyset \\ D_1 & = & \{(\,)\} \\ D_{a\oplus b} & = & D_a + D_b \\ D_{a\otimes b} & = & D_a \times D_b \end{array}$$

Produsul pe mulţimi este *asociativ* modulo un izomorfism. De asemenea, suma disjunctă este *asociativă* modulo un izomorfism. Pe de altă parte, pe nici una dintre cele două operaţii de mai sus nu o considerăm *comutativă*. Notaţiile folosite pentru tupluri sunt $\langle \ldots \rangle$, respectiv $(\ldots)$.

Modelul semantic de bază va fi $MixRel_S(D)$:

$$MixRel_S(D)(a,b) = \{\ f \mid f \subseteq D_a \times D_b\ \},$$

unde $a, b \in S^{\oplus,\otimes}$.

Noile constante $\delta_;$ şi $\rho_;$ vor reflecta izomorfismele corespunzătoare distributivităţii produsului asupra sumei. În $MixRel$, ele au următoarea semnificaţie:

- **Distributivitate:** $\delta_{a_1,\ldots,a_m;b_1,\ldots,b_n} : (a_1 \oplus \ldots \oplus a_m) \otimes (b_1 \oplus \ldots \oplus b_n) \to (a_1 \otimes b_1) \oplus (a_1 \oplus b_2) \oplus \ldots \oplus (a_m \otimes b_1) \oplus \ldots \oplus (a_m \otimes b_n)$ este definită prin:

$$\delta_{a_1,\ldots,a_m;b_1,\ldots,b_n} = \{\ ((x,y),z) \mid \quad \begin{array}{l} \text{dacă } x = i.x_i \text{ şi } y = j.y_j, \\ \text{atunci } z = (i-1)n + j.(x_i, y_j), \text{ unde} \\ i \in [m],\ j \in [n],\ x_i \in D_{a_i},\ y_j \in D_{b_j}\ \}. \end{array}$$

- **Reducere:** $\rho_{a_1,\ldots,a_m;b_1,\ldots,b_n} : (a_1 \otimes b_1) \oplus (a_1 \oplus b_2) \oplus \ldots \oplus (a_m \otimes b_1) \oplus \ldots \oplus (a_m \otimes b_n) \to (a_1 \oplus \ldots \oplus a_m) \otimes (b_1 \oplus \ldots \oplus b_n)$ este definită ca fiind relaţia inversă asociată lui $\delta_{a_1,\ldots,a_m;b_1,\ldots,b_n}$.

Structura de algebră a reţelelor mixtă este folositoare în studiul programelor cu procese paralele (v. [GLS00]) şi în încercarea de a modela fluxnomial conceptele de clasă şi obiect (v. secţiunea 2.2). Nu ne vom ocupa prea mult de fundamentul axiomatic. Mai precis, nu se va prezenta un set de axiome pentru interacţia dintre structurile AddNA şi MultNA din algebra MixNA relativ la operatorii de feedback.

Fragmentul fără feedback dintr-o algebră MixNA este specificat de regulile unei categorii simetrice semiinelare îmbogăţită cu constante de ramificare aditive şi multiplicative (vezi [GLS00]). Structura de bază este aceea a unei *categorii simetrice semiinelare* $(M, \oplus, \otimes, \cdot, \mathsf{I}, \delta_;, \rho_;, \between, \mathsf{X})$ constând din două categorii strict monoidale $(M, \oplus, \cdot, \mathsf{I}, \between)$ şi $(M, \otimes, \cdot, \mathsf{I}, \mathsf{X})$ care respectă regulile D0–D11 de mai jos:

D0 $\delta_{a;1,0} = \mathsf{I}_a$

D1 $\delta_{a;b,c} \cdot \rho_{a;b,c} = \mathsf{I}_{a\otimes(b\oplus c)}$

D2 $\rho_{a;b,c} \cdot \delta_{a;b,c} = \mathsf{I}_{(a\otimes b)\oplus(a\otimes c)}$

D3 $\delta_{a;b\oplus c,d} \cdot (\delta_{a;b,c} \oplus \mathsf{I}_{a\otimes d}) = \delta_{a;b,c\oplus d} \cdot (\mathsf{I}_{a\otimes b} \oplus \delta_{a;c,d})$

D4 $\delta_{a;b,c} \cdot {}^{a\otimes c}_{a\otimes b}\between = (\mathsf{I}_a \otimes {}^{c}_{b}\between) \cdot \delta_{a;c,b}$

D5 $\delta_{0;b,c} = \mathsf{I}_0$

D6 $\delta_{1;b,c} = \mathsf{I}_{b\oplus c}$

D7 $\delta_{a'\oplus a'';b,c} = \delta_{a',a'';b\oplus c}\cdot(\delta_{a';b,c}\oplus\delta_{a'';b,c})\cdot(\mathsf{I}_{a'\otimes b}\oplus {}^{a''\otimes b}_{a'\otimes c}\mathsf{X}\oplus\mathsf{I}_{a''\otimes c})\cdot(\rho_{a',a'';b}\oplus\rho_{a',a'';c})$

D8 $\delta_{a'\otimes a'';b,c} = (\mathsf{I}_{a'}\otimes\delta_{a'';b,c})\cdot\delta_{a';a'\otimes b,a''\otimes c}$

D9 $\delta_{a;} = \mathsf{I}_0$

D10 $(f\otimes(g\oplus h))\cdot\delta_{a';b'\oplus c'} = \delta_{a;b,c}\cdot((f\otimes g)\oplus(f\otimes h))$

D11 $(f\otimes\mathsf{I}_0)\cdot\delta_{a';} = \delta_{a;}$

Privitor la operatorii de feedback, vom folosi următoarele identităţi valide:

Mix1 Feedback-ul aditiv comută cu produsul cu o identate, mai precis, pentru $f : b\oplus d\to c\oplus d$ avem:

$$\mathsf{I}_a\otimes(f\uparrow^{d}_{\oplus}) = (\rho_{a;b,d}\cdot(\mathsf{I}_a\otimes f)\cdot\delta_{a;c,d})\uparrow^{a\otimes d}_{\oplus}\ .$$

Mix2 Pentru $f : a\otimes b\to c\oplus d$ şi $g : d\to b$, notăm cu

$$E := ({}_2{>}\!\!\bullet_{a\otimes b}\cdot(\overset{\circ}{\wedge}{}^{a}\otimes\mathsf{I}_b)\cdot(\mathsf{I}_a\otimes f)\cdot\delta_{a;c,d}\cdot(\mathsf{I}_{a\otimes c}\oplus\mathsf{I}_a\otimes g))\uparrow^{a\otimes b}_{\oplus}\ .$$

Identitatea Mix2 este

$$E = (\overset{\circ}{\wedge}{}^{a}\otimes\mathsf{I}_b)\cdot(\mathsf{I}_a\otimes E\cdot(\underset{\circ}{\vert}{}^{a}\otimes\mathsf{I}_c)).$$

Această egalitate ne spune faptul că variabilele ‘read-only’ folosite într-un ciclu îşi păstrează valoarea iniţială la ieşirea din ciclu.

2 Concepte informatice modelate fluxnomial

În această parte a lucrării de dizertaţie dorim să prezentăm câteva abordări folosind teoria fluxnoamelor a unor concepte clasice în informatică, de genul grafuri, obiecte şi clase. Vor fi exprimate formal exprimarea unui graf (în particular, a unui arbore) printr-un fluxnom şi invers, obţinerea grafului/reţelei asociate unei expresii. De asemenea, vom specifica conceptele de clasă şi obiect din domeniul obiect-orientării cu ajutorul expresiilor fluxnomiale.

2.1 Grafuri

Modul cel mai simplu de a reprezenta grafic o reţea este sub forma unui *graf*, aceasta datorându-se faptului că el se apropie cel mai mult de definiţia generală a reţelei.

Un *graf* (direcţionat) este format dintr-o mulţime de *noduri* şi o mulţime de perechi (ordonate) de noduri numite *muchii*, arătând relaţia dintre noduri. În continuare, vom lucra cu *hipergrafuri*, forme mai generale de grafuri. Ele vor accepta mai multe muchii între două noduri şi nişte noduri speciale de intrare, respectiv ieşire. De asemenea, putem considera cazul în care muchiile sunt etichetate.

2.1.1 Definiţii

Să dăm pentru început definiţiile necesare studiului din subsecţiunile următoare:

Definiţia 2.1.1 (Graf direcţionat) Un **graf** G este o pereche (N, M) unde:

- N este o mulţime nevidă ale cărei elemente se numesc **noduri**;
- M este o mulţime (eventual vidă) de perechi ordonate de noduri, numite **muchii**.

Definiţia 2.1.2 (Hipergraf direcţionat) Un **hipergraf** H este un cvadruplu $(N, I, O; M)$ unde:

- N este o mulţime ale cărei elemente se numesc **noduri (interne)**;
- I este o mulţime ale cărei elemente se numesc **noduri de intrare**;
- O este o mulţime ale cărei elemente se numesc **noduri de ieşire**;
- M este un multiset (mulţime cu repetiţii) de perechi ordonate de noduri (interne, de intrare sau de ieşire) numite **muchii**, astfel încât într-o pereche, nodurile de intrare nu pot apărea decât pe prima poziţie, iar cele de ieşire pe a doua poziţie.

N, I şi O sunt mulţimi disjuncte şi N, I, O, M pot fi eventual vide, cu restricţia $N \cup I \cup O \neq \emptyset$.

Pentru simplitate, vom spune graf, respectiv hipergraf în loc de graf *direcţionat*, respectiv hipergraf *direcţionat*.

Definiţia 2.1.3 (Izomorfism de hipergrafuri) Două hipergrafuri se numesc **izomorfe** dacă există o bijecţie între nodurile interne, de intrare şi de ieşire, astfel încât folosind această bijecţie, multimulţimea muchiilor primului hipergraf este transformată în multimulţimea muchiilor celui de-al doilea hipergraf.

Observaţia 2.1.4 *Grafurile sunt cazuri particulare de hipergrafuri, alegând $I = O = \emptyset$ şi M mulţime.*

Se poate considera şi cazul în care muchiile sunt *etichetate*. În această situaţie, muchiile nu vor mai fi reprezentate prin perechi ordonate, ci prin triplete de forma (i, j, e) cu i şi j noduri, iar e etichetă aparţinând unei mulţimi de etichete E. Presupunem că toate muchiile care pleacă dintr-un nod de intrare (respectiv toate muchiile care ajung într-un acelaşi nod de ieşire) sunt etichetate la fel. Punem această restricţie pentru a fi în concordanţă cu schemele de programe: printr-o intrare intră doar date de un anumit tip şi canalele care le distribuie trebuie etichetate la fel (de obicei cu tipul datei). Oricum, în cazul în care am avea nevoie de etichete diferite pentru a modela o structură, se pot face 'artificii' tehnice folosind noduri interne, acestea neavând impusă această condiţie.

Exemplul 2.1.5 (Reţea de calculatoare) Un exemplu de reţea este dat de o reţea locală de calculatoare (*în engleză*: LAN [9]). Modelând acest exemplu cu *grafuri*, putem privi calculatoarele ca fiind noduri într-un graf, iar legăturile dintre ele ca fiind muchii. Dacă $\{C_1, C_2, \ldots, C_n\}$ sunt calculatoarele, o muchie (C_i, C_j) ne spune faptul că C_i poate avea acces la resursele lui C_j. Dacă dorim să specificăm şi rata de transfer a informaţiei între C_i şi C_j, putem să etichetăm muchiile cu elemente din mulţimea numerelor reale, reprezentând câţi Mb/secundă se pot transfera.

Complicând puţin exemplul, putem presupune că această reţea locală de calculatoare este legată la Internet, deci există nişte dispozitive ce permit operaţiuni de intrare/ieşire de date în/din reţeaua locală de calculatoare şi mai putem să presupunem că între două calculatoare pot exista mai multe căi de comunicaţie (cabluri). Această situaţie o modelăm cu *hipergrafuri*, acestea având noduri de intrare I, noduri de ieşire O pentru dispozitivele de intrare/ieşire şi permiţând mai multe muchii între două noduri. Condiţia etichetării la fel pentru muchiile adiacente cu noduri de intrare/ieşire nu este prea restrictivă: se poate presupune că dispozitivele de intrare/ieşire pot transmite date doar la o anumită rată de transfer.

2.1.2 De la (hiper)grafuri la fluxnoame

În continuare vom oferi un algoritm care transformă un hipergraf într-o expresie fluxnomială. Cum grafurile sunt cazuri particulare de hipergrafuri, algoritmul este funcţionabil şi pentru ele. Mai precis, construim o aplicaţie de la mulţimea hipergrafurilor la mulţimea fluxnoamelor (peste X şi T alese corespunzător):

$$\mathcal{F} : HiperGraf \longrightarrow Flux.$$

Considerăm cazul grafurilor cu muchiile neetichetate pentru a simplifica lucrurile.

[9]Local Area Network

Algoritmul 2.1.6 Fie $HG \in HiperGraph$ cu $HG = (N, I, O; M)$. Aplicăm următorii paşi:

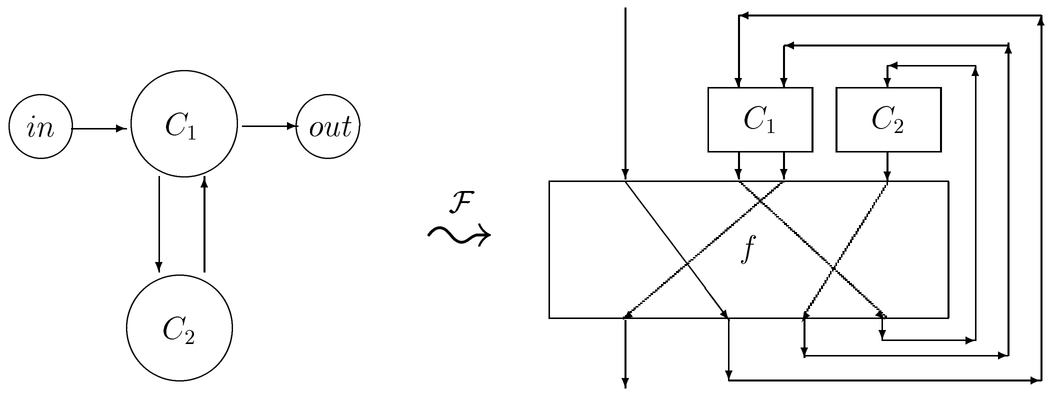
1. Ordonăm elementele mulţimilor N, I, O şi M (deci va exista o ordine strictă $<$ pe fiecare din aceste mulţimi).
2. Considerăm funcţiile
$$in, out : N \longrightarrow \mathbf{N}$$
ce specifică câte muchii 'intră', respectiv 'ies' dintr-un anumit nod.
3. Iniţializăm doi vectori indexaţi după elemente $c \in N$: $k_c := 1$ şi $l_c := 1$.
4. Luăm $X := N$ şi f o relaţie, $f \subseteq [m] \times [n]$ cu
$$m := card(I) + \sum_{x \in N} out(x), \quad n := card(O) + \sum_{x \in N} in(x).$$
5. Pentru fiecare muchie $(a, b) \in M$, adăugăm câte o pereche (i, j) la relaţia f, astfel:
$$i := \begin{cases} a & \text{dacă} \quad a \in I \\ card(I) + \sum_{t<a} out(t) + k_a & \text{dacă} \quad a \in N \end{cases}$$
$$j := \begin{cases} b & \text{dacă} \quad b \in O \\ card(O) + \sum_{t<b} in(t) + l_b & \text{dacă} \quad b \in N \end{cases}$$
şi incrementăm pe k_a, dacă $a \in N$ şi pe l_b, dacă $b \in N$.
6. Alegem $\mathcal{F}(HG) = \Big((\mathsf{I}_{card(I)} \star c_1 \star \ldots \star c_{card(N)}) \cdot f \Big) \uparrow^{in(c_1) + \ldots + in(c_{card(N)})}$

Aplicând algoritmul de mai sus, obţinem o expresie fluxnomială în formă normală care are $card(I)$ intrări şi $card(O)$ ieşiri, nodurile de intrare transformându-se în intrări, cele de ieşire în ieşiri, iar nodurile interne c se transformă în variabile din $X(in(c), out(c))$. Observăm că efortul trecerii de la hipergrafuri la fluxnoame a fost liniar, fiind de ordinul $\mathcal{O}(card(N))$. Forma normală obţinută oferă o imagine structurată asupra hipergrafului, grupând nodurile şi respectiv muchiile.

Pentru cazul hipergrafurilor *etichetate* cu etichete din E, algoritmul este similar. Apar modificări în alegerea monoidului interfeţelor, care nu mai este $(\mathbf{N}, +, 0)$, ci este ales $(E^*, \star, \lambda)$, monoidul liber generat al cuvintelor peste alfabetul E (considerând elementele lui E atomi). Operaţia de concatenare va fi notată cu $\star$ (pentru intuiţia juxtapunerii a două interfeţe), iar λ este cuvântul vid. Etichetele muchiilor se transferă la interfeţele nodurilor adiacente, devenind sorturi. Condiţia aceloraşi etichete pentru nodurile de intrare/ieşire îşi arată utilitatea, nodurile de intrare/ieşire fiind transformate în intrări/ieşiri de sort egal cu eticheta comună.

Astfel, în algoritmul 2.1.6 înlocuim $+$-ul cu $\star$ acolo unde apar funcţiile *in* sau *out*, cardinalul cu juxtapunerea sorturilor intrărilor/ieşirilor, iar f-ul va deveni o relaţie sortată corespunzător etichetelor muchiilor.

Exemplul 2.1.7 (O reţea concretă de calculatoare) Iată o reţea formată din două calculatoare, din care primul este legat la Internet:



unde $HG = (\{in\}, \{out\}, \{C_1, C_2\}; \{(in, C_1), (C_1, C_2), (C_1, out), (C_2, C_1)\})$ şi $\mathcal{F}(HG) = ((\mathsf{I}_1 \star C_1 \star C_2) \cdot f) \uparrow^3$, cu $f = \{(1,2), (2,4), (3,1), (4,3)\}$ şi $C_1 : 2 \longrightarrow 2$, $C_2 : 1 \longrightarrow 1$.

Astfel, dacă avem o reţea de calculatoare şi ştim cine are acces la cine, folosind algoritmul 2.1.6, am putea organiza reţeaua într-un mod mai eficient. În loc să legăm fiecare calculator cu cele cu care este în legătură, folosim un dispozitiv care se ocupă cu primirea şi distribuţia datelor şi întreaga comunicaţie se face prin intermediul lui (un exemplu de un astfel de dispozitiv este 'hub'-ul, foarte folosit în LAN-uri). În acest exemplu, f-ul modelează dispozitivul de legătură, arătându-ne conexiunile interne, C_1 şi C_2 fiind conectate corespunzător la el.

Transformarea $\mathcal{F}$ este folositoare de exemplu în cazul în care dorim să scriem compact într-o singură expresie o anumită reţea: stabilim legăturile dintre diferitele elemente şi obţinem un graf, iar apoi aplicăm algoritmul dat mai sus.

2.1.3 De la fluxnoame la (hiper)grafuri

Să vedem şi construcţia în sens invers. Presupunem că avem o expresie fluxnomială. Putem să oferim algoritmi pentru a o transforma într-un hipergraf. Mai precis, construim o aplicaţie de la mulţimea fluxnoamelor peste X şi T la mulţimea hipergrafurilor:

$$\mathcal{HG} : Flux \longrightarrow HiperGraf.$$

Considerăm cazul interfeţelor netipizate sau monosortate pentru a simplifica lucrurile. Avem cel puţin două posibilităţi de a face acest lucru, astfel încât să obţinem un hipergraf care să reflecte legăturile dintre elementele din X. Prima posibilitate este aceea de a aduce expresia la o formă normală (x, f), iar apoi să folosim f-ul pentru a obţine muchiile. O a doua posibilitatea este de a 'construi' hipergraful pornind de la elementele atomice şi apoi aplicând iterativ operaţiile care apar în expresie în cazul hipergrafurilor. De fapt, metodele sunt similare deoarece aducerea expresiei la o formă normală presupune o construcţie inductivă asemănătoare celei de a doua posibilităţi.

Algoritmul 2.1.8 Fie E o expresie fluxnomială peste X şi T. Aplicăm următorii paşi:

1. Aducem expresia E la o formă normală (x, f) cu $x = x_1 \star \ldots \star x_p$, (v. secţiunea 1.2). Notăm $m := in(f)$, $n := out(f)$, $i := m - out(x)$, $o := n - in(x)$.
2. Fie $I := \{in_1, \ldots, in_i\}$, $O := \{out_1, \ldots, out_o\}$, $N = \{x_1, \ldots, x_p\}$ şi $M := \emptyset$. M presupunem că este multiset, deci pot apărea şi repetiţii atunci când vom adăuga elemente în el.

3. Alegem pe rând fiecare pereche (k, l) din f. Adăugăm câte o pereche (a, b) la M, unde

$$a := \begin{cases} in_k, & \text{dacă} \quad k \leq i \\ x_{a'}, & \text{dacă} \quad k > i \end{cases} \qquad b := \begin{cases} out_l, & \text{dacă} \quad l \leq o \\ x_{b'}, & \text{dacă} \quad l > o \end{cases}$$

$$a' := min\{t \mid 1 \leq t \leq p \text{ şi } i + \sum_{s:=1}^{t-1} out(x_s) < k \leq i + \sum_{s:=1}^{t} out(x_s)\},$$

$$b' := min\{t \mid 1 \leq t \leq p \text{ şi } o + \sum_{s:=1}^{t-1} in(x_s) < l \leq o + \sum_{s:=1}^{t} in(x_s)\}.$$

4. Alegem $\mathcal{HG}(E) := (I, O, N; M)$.

Algoritmul 2.1.9 Fie E o expresie fluxnomială peste X şi T. Vom construi hipergraful $\mathcal{HG}(E)$ inductiv după structura lui E:

1. Dacă $f \subseteq [m] \times [n]$ este o relaţie în T, folosim m noduri de intrare, n noduri de ieşire şi nici un nod intern:

$$\mathcal{HG}(f) = (\{in_1, \ldots, in_m\}, \{out_1, \ldots, out_n\}, \emptyset; f).$$

De exemplu, constantele (care apar în T) pot fi reprezentate astfel:

$$\begin{array}{rcl} \mathcal{HG}(\mathsf{I}_1) & := & (\{in_1\}, \{out_1\}, \emptyset; \{(in_1, out_1)\}) \\ \mathcal{HG}({}^1\mathsf{X}^1) & := & (\{in_1, in_2\}, \{out_1, out_2\}, \emptyset; \{(in_1, out_2), (in_2, out_1)\}) \\ \mathcal{HG}(\wedge_k^1) & := & (\{in_1\}, \{out_1, \ldots, out_k\}, \emptyset; \{(in_1, out_1), \ldots, (in_1, out_k)\}) \\ \mathcal{HG}(\vee_1^k) & := & (\{in_1, \ldots, in_k\}, \{out_1\}, \emptyset; \{(in_1, out_1), \ldots, (in_k, out_1)\}) \end{array}$$

2. Dacă x este o variabilă în $X(m, n)$, folosim m noduri de intrare, n noduri de ieşire şi un nod intern:

$$\mathcal{HG}(x) = (\{in_1, \ldots, in_m\}, \{out_1, \ldots, out_n\}, \{x\};$$
$$\{(in_1, x), \ldots, (in_m, x), (x, out_1), \ldots, (x, out_n)\}).$$

3. Fie E_1 şi E_2 două expresii fluxnomiale. Dacă pentru $i \in \{1, 2\}$ avem $\mathcal{HG}(E_i) = (\{in_1, \ldots, in_{m_i}\}, \{out_1, \ldots, out_{n_i}\}, N_i; M_i)$, alegem

$$\mathcal{HG}(E_1 \star E_2) = (\{in_1, \ldots, in_m\}, \{out_1, \ldots, out_n\}, N; M),$$

unde $m := m_1 + m_2$, $n := n_1 + n_2$, N este reuninunea disjunctă a mulţimilor N_1 şi N_2, iar M este reuninunea disjunctă a multimulţimilor M_1 şi M_2, ţinând cont de faptul că intrările $\{in_1, \ldots, in_{m_2}\}$ din $\mathcal{HG}(E_2)$ au fost redenumite cu $\{in_{m_1+1}, \ldots, in_{m_1+m_2}\}$ (analog ieşirile) şi de asemenea N_1 şi N_2 au fost redenumite.

4. Fie E_1 şi E_2 două expresii fluxnomiale, ca mai sus. Dacă $n_2 = m_1$, alegem

$$\mathcal{HG}(E_1 \cdot E_2) = (\{in_1, \ldots, in_m\}, \{out_1, \ldots, out_n\}, N; M),$$

unde $m := m_1$, $n := n_2$, N este reuniunea disjunctă a mulţimilor N_1 şi N_2, iar M este obţinut din M_1 şi M_2 prin identificarea intrărilor lui $\mathcal{HG}(E_1)$ cu ieşirile lui $\mathcal{HG}(E_2)$ şi 'absorbirea' lor. Mai precis,

$$M := M_1 \cup M_2 \backslash (\{(a, out_i) \mid (a, out_i) \in M_1\} \cup \{(in_i, b) \mid (in_i, b) \in M_2\}) \cup$$

$$\cup \{(a, b) \mid \exists\, 1 \leq i \leq n_1 \text{ a.î. } (a, out_i) \in M_1 \text{ şi } (in_i, b) \in M_2\}.$$

5. Fie E o expresie fluxnomială astfel încât

$$\mathcal{HG}(E) = (\{in_1, \ldots, in_{m+1}\}, \{out_1, \ldots, out_{n+1}\}, N; M).$$

Alegem $\mathcal{HG}(E \uparrow^1) = (\{in_1, \ldots, in_m\}, \{out_1, \ldots, out_n\}, N; M')$, unde M' este obţinut din M, prin 'legarea' ieşirii out_{n+1} de intrarea in_{m+1} şi apoi 'absorbirea' lor. Mai precis,

$$M' := M \backslash \{(a, b) \mid a = in_{m+1} \text{ sau } b = out_{n+1}\} \cup$$

$$\cup \{(a, b) \mid \exists\, (a, out_{n+1}) \in M \text{ şi } (in_{m+1}, b) \in M\}.$$

Propoziţia 2.1.10 *Dacă se dă o expresie fluxnomială, prin aplicarea algoritmilor 2.1.8 şi 2.1.9 obţinem două hipergrafuri izomorfe.*

Demonstraţie: Uşor, prin inducţie structurală asupra expresiei. ■

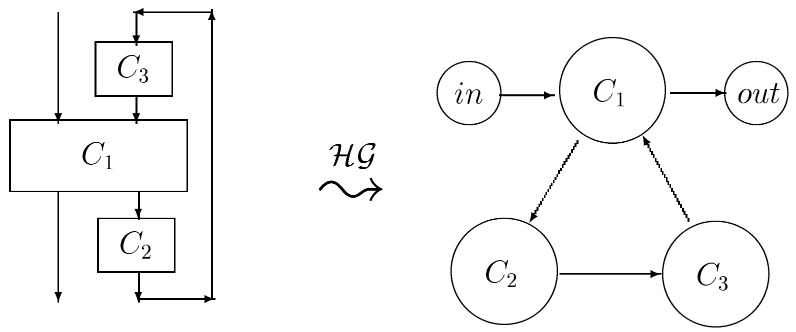
Propoziţia 2.1.11 *Avem $\mathcal{HG} \circ \mathcal{F} \simeq Id$, adică dându-se un hipergraf HG, el va fi izomorf cu $\mathcal{HG}(\mathcal{F}(HG))$.*

Demonstraţie: Fie HG un hipergraf. Aplicându-i algoritmul 2.1.6, obţinem $\mathcal{F}(HG)$, care este o expresie în formă normală. Acesteia îi aplicăm algoritmul 2.1.8. Pasul 1 al acestuia este nu face nici o operaţie datorită faptului că expresia noastră este deja în formă normală. Aplicând următorii doi paşi, vom obţine un hipergraf izomorf cu HG, datorită construcţiei care este de fapt inversă celei din algoritmul 2.1.6 (muchiile hipergrafului $\leftrightarrow$ perechile relaţiei f).

Dacă expresiei $\mathcal{F}(HG)$ i se aplică cel de-al doilea algoritm 2.1.9, datorită propoziţiei anterioare şi faptului că pentru primul algoritm obţinem un izomorfism, deducem că propoziţia este adevărată şi dacă în construcţia lui $\mathcal{HG}$ folosim cel de-al doilea algoritm. ■

Observaţia 2.1.12 *Nu este adevărat că $\mathcal{F} \circ \mathcal{HG} = Id$ (modulo axiome), adică dându-se o expresie E, ea nu va fi egală cu $\mathcal{F}(\mathcal{HG}(E))$. De exemplu, pentru $E = {}^1\mathsf{X}^1$, avem $\mathcal{F}(\mathcal{HG}({}^1\mathsf{X}^1)) = \mathsf{I}_2 \neq {}^1\mathsf{X}^1$. Aceasta se datorează faptului că în cazul hipergrafurilor nu există o ordine a muchiilor adiacente unui nod, în timp ce la fluxnoame, ordinea intrărilor şi a ieşirilor este foarte importantă.*

Exemplul 2.1.13 Să vedem cum arată hipergraful asociat unei expresii particulare:



Avem $E = ((\mathsf{I}_1 \star C_3) \cdot C_1 \cdot (\mathsf{I}_1 \star C_2)) \uparrow^1$, cu $C_1 : 2 \longrightarrow 2$, $C_2, C_3 : 1 \longrightarrow 1$ şi aplicând algoritmul obţinem $\mathcal{HG}(E) = (\{in\}, \{out\}, \{C_1, C_2, C_3\}; M)$, cu $M = \{(in, C_1), (C_1, C_2), (C_2, C_3), (C_3, C_1), (C_1, out)\}$. Având expresia E, este destul de greu a intui ce reţea specifică, dar transformând-o într-un hipergraf, vedem că acea reţea ar putea modela o reţea de calculatoare locală *inelară* legată la Internet.

Atunci când dorim să modelăm un sistem format din mai multe module aflate într-o anumită relaţie de interdependenţă, putem face aceasta pe mai multe nivele de abstractizare. Grafurile oferă cel mai simplu mod, ele conţinând doar informaţii despre ce module sunt în sistem şi ce relaţii există între ele. Fluxnoamele conţin mai multă informaţie, oferă o rafinare a abstractizării, presupun o ordonare a legăturilor, o stratificare, o organizare spaţială a modulelor, legăturile dintre ele putând chiar să se 'întretaie' (v. ${}^i\mathsf{X}^j$). Algoritmii prezentaţi fac trecerea de la hipergrafuri la fluxnoame şi invers, dându-ne libertatea ca, având o structură simplă, să o formalizăm algebric şi având o prezentare algebrică a unui sistem, să căpătăm intuiţia dată de o reprezentare simplificată.

2.1.4 Arbori

Arborii fiind un caz particular de grafuri, acceptă o reprezentare fluxnomială, însă tot datorită faptului că sunt un caz particular, lor li se poate găsi o exprimare mai elegantă şi mai apropiată de elementele caracteristice structurii de arbore.

Definiţia 2.1.14 (Arbore) Un **arbore** este un graf conex fără cicluri având un nod distins numit **rădăcină**.

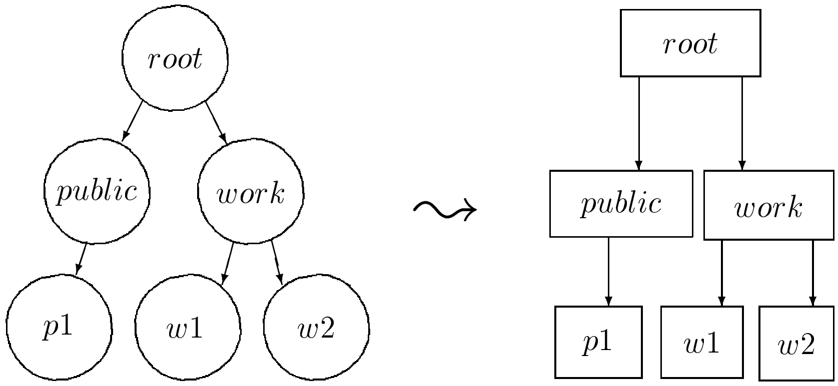
Trecerea de la arbori la fluxnoame:

Algoritmul 2.1.15 Ideea este următoarea:

1. se parcurge arborele pe nivele;
2. toate nodurile de pe fiecare nivel se juxtapun, obţinând câte o expresie pentru fiecare nivel;

3. toate expresiile corespunzătoare nivelelor se compun, obţinând expresia arborelui.

Exemplul 2.1.16 (O structură arborescentă de directoare) Iată un exemplu:



Expresia fluxnomială asociată arborelui de mai sus este

$$root \cdot (public \star work) \cdot (p1 \star w1 \star w2),$$

unde $root : 0 \longrightarrow 2$, $public : 1 \longrightarrow 1$, $work : 1 \longrightarrow 2$, $p1, w1, w2 : 1 \longrightarrow 0$.

2.2 Un exerciţiu: obiecte şi clase

În istoria oamenilor şi de multe ori în cercetarea ştiinţifică este valabilă afirmaţia: *Orice om cu un ciocan în mână va fi înclinat să vadă toate lucrurile din jurul lui în formă de cuie.* Având la îndemână teoria fluxnoamelor, putem încerca să o folosim în lumea largă a informaticii, acesta fiind un exerciţiu (uneori încununat de succes) şi nu un scop în sine. În continuare vom căuta să utilizăm teoria fluxnoamelor pentru a modela conceptele de *obiect* şi *clasă* din vastul şi importantul domeniu al *programării orientate pe obiecte*. În literatură există o mulţime de abordări ale acestei paradigme, dintre cele mai importante amintind: tipuri abstracte de date (*Abstract Data Type*) (v. [GTW78], [Mey88]), coalgebre (*Coalgebras*) (v. [Jac96]) şi algebre cu sorturi ascunse (*Hidden Algebra*) (v. [GD94]). Abordarea pe care o vom prezenta se apropie cel mai mult de cea coalgebrică a lui Bart Jacobs ([Jac96]).

În ultimii zece ani ai mileniului doi, tehnologia produselor software s-a aflat sub zodia paradigmelor orientării pe obiecte şi probabil va rămâne pe această linie şi în continuare. Aceste paradigme oferă o viziune mult mai limpede asupra lucrurilor şi a relaţiilor dintre ele. Tehnologia obiect-orientării are la baza ideea simplă că cea mai bună organizare a unui sistem este obţinută prin descompunerea lui în **entităţi** aflate într-o anumită relaţie. Aceste entităţi au anumite **caracteristici** ce se împart in două clase: **atribute** şi **operaţii**. Astfel, o entitate informatică are anumiţi parametri (sau *atribute*) şi poate executa anumite acţiuni (cu ajutorul *operaţiilor*). Observăm că în acest fel sunt surprinse atât dimensiunea *spaţială* dată de parametrii ce sunt memoraţi static sau spaţial, cât şi cea *temporală* dată de acţiunile ce se desfăşoară dinamic, în timp. O **clasă** este prin definiţie o astfel de entitate care deţine un set de informaţii şi uneltele necesare de a folosi informaţiile respective. Să prezentăm elementele esenţiale ale tehnologiei orientării pe obiecte:

- **specificaţiile de clase** sunt o descriere abstractă a unei clase, conţinând tipul atributelor şi operaţiilor împreună cu constrângerile comportamentale asupra acestora;
- **clasele** reprezintă o implementare a specificaţiilor de clase, asociind fiecărui atribut o mulţime în care poate lua valori (tip de date) şi fiecărei operaţii o funcţie clar precizată;

- **obiectele** sunt pur şi simplu instanţieri ale atributelor, asupra cărora se pot aplica operaţiile clasei din care face parte.

Exemple concrete de clase sunt oferite în anexele A şi B, clasele prezentate fiind implemetări ale relaţiilor finite sortate (v. subsecţiunea 1.1.1) şi ale formelor normale ale expresiilor fluxnomiale (v. secţiunea 1.2).

2.2.1 Un exemplu financiar

Pentru a înţelege mai uşor structura unei *clase*, care este elementul principal în lumea obiect-orientării (deşi iniţial suntem tentaţi să spunem că *obiectul* este entitatea de bază), să începem printr-un exemplu.

Exemplul 2.2.1 Fără a fi catalogaţi prozaici sau materialişti, să specificăm printr-o clasă un cont bancar rudimentar:

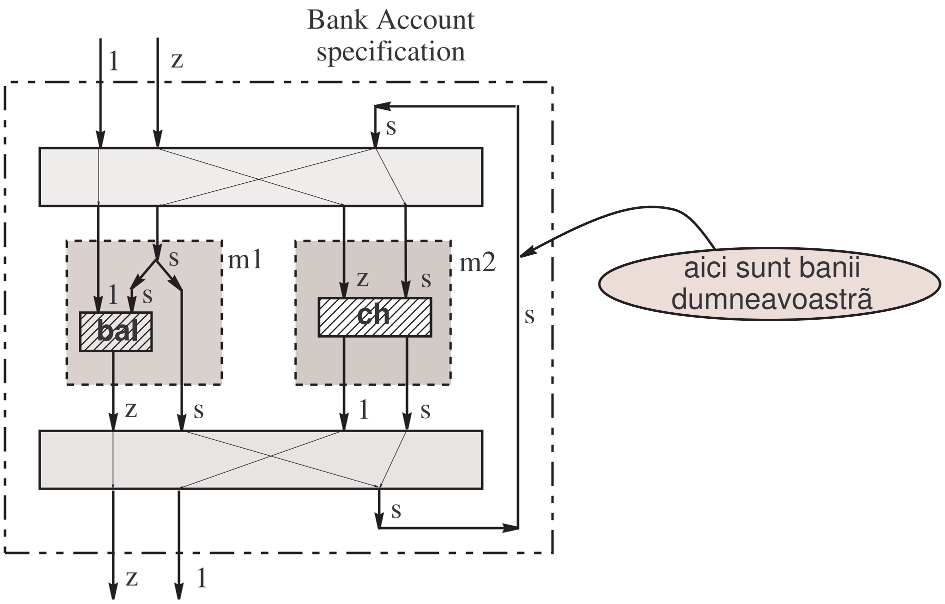
class spec: `BA` `// numele specificaţiei clasei (Bank Account)`
  public methods:
    `bal`:$X \longrightarrow \mathbf{Z}$ `// atributul pentru suma din cont (balance)`
    `ch`:$X \times \mathbf{Z} \longrightarrow X$ `// procedură ce schimbă starea contului`
      `// folosind un parametru întreg (change)`
  assertions:
    `s.ch(`a`).bal = s.bal+`a `// 's' este starea locală (state)`
  creation:
    `new.bal = 0`
end class spec

Această clasă are ca atribut pe `bal`, adică suma de bani din cont şi operează asupra ei cu procedura `ch`, care poate adăuga sau scoate bani din cont după cum parametrul întreg este pozitiv sau negativ. Expresia fluxnomială din cadrul mixt MixNA prin care vom descrie această specificaţie de clasă este:

$$E_{\mathtt{BA}} = [\delta_{1,z;s} \cdot (m_1 \oplus m_2) \cdot \rho_{z,1;s}] \uparrow^{s}_{\otimes}$$

cu $m_1 = (\mathsf{I}_1 \otimes \wedge^s) \cdot (\mathtt{bal} \otimes \mathsf{I}_s)$ şi $m_2 = \mathtt{ch}$, unde $\mathtt{bal} : 1 \otimes s \longrightarrow z$ şi $\mathtt{ch} : z \otimes s \longrightarrow 1 \otimes s$. Deşi avem $\mathtt{bal} : X \longrightarrow \mathbf{Z}$ în specificaţia clasei `BA` de mai sus, ne-am folosit de faptul că 1 este element neutru faţă de juxtapunerea multiplicativă $\otimes$: $1 \otimes s = s = s \otimes 1$. Observăm că $m_1 : 1 \otimes s \longrightarrow z \otimes s$ şi

$m_2 : z \otimes s \longrightarrow 1 \otimes s$. Se subînţelege că z este sortul asociat lui $\mathbf{Z}$, iar s este sortul asociat stării locale/interne S a obiectului. Iată reprezentarea grafică a clasei:



În această specificare, spunem ce metode dorim să avem pentru contul de bancă şi ce aserţiuni (ecuaţionale) trebuie ca acestea să satisfacă. Ecuaţia

$$\mathtt{s}.\mathtt{ch}(a).\mathtt{bal} = \mathtt{s}.\mathtt{bal} + a$$

trebuie citită: dacă stării `s` i se trimite mesajul de schimbare a sumei din cont `ch` cu parametrul a şi apoi este cerută suma din cont `bal`, atunci rezultatul este acelaşi cu a cere balanţa stării `s` şi a o aduna cu a. Ultima parte din specificaţie menţionează faptul că obiectele nou-create din clasa `BA` au ca balanţă pe $0 \in \mathbf{Z}$. Aceasta descrie comportamentul stării iniţiale. Ca observatori exteriori, nu ne interesează cum sunt implementate operaţiunile bancare, atâta timp cât ele îndeplinesc specificaţia. Noi nu avem acces la spaţiul stărilor interne S decât prin intermediul celor două metode de mai sus. De remarcat că o astfel de specificaţie de clasă seamănă mult cu *clasele neimplementate* (sau *virtuale*) *cu aserţiuni* din Eiffel din cunoscuta carte *Object-Oriented Software Construction* a lui Bertrand Meyer [Mey88], unde sunt date doar metodele cu tipurile intrărilor şi ieşirilor (fără vreo interpretare), iar comportamentul lor este determinat prin aserţiuni. Având o specificaţie de clasă, vom putea defini o clasă interpretând simbolurile de funcţii astfel încât să fie satisfăcute aserţiunile.

Semantica folosită va fi una mixtă. Starea locală va fi 'depozitată' în conexiunea ieşiri-intrări făcută de feedback-ul de sort s. Pentru aceasta, vom presupune că firele sunt buffere de capacitate 1.

Ne permitem să fim hâtri şi să afirmăm că un mare avantaj al modelării ca mai sus a unui cont bancar este siguranţa că putem să adăugăm şi să scoatem o mulţime de bani din cont fără ca Garda Financiară să ne ceară vreo chitanţă şi de asemenea putem să fim siguri că nu ne vom pierde săracele noastre economii prin falimentarea băncii de către miliardari de carton, aceştia neavând la dispoziţie 'metode' de împrumuturi fără acoperire...

2.2.2 Obiecte şi clase - definiţii

Folosind teoria fluxnoamelor, vom da în continuare definiţii formale noţiunilor de bază din teoria obiect-orientării.

Definiţia 2.2.2 (**Specificaţia unei clase**) O **specificaţie a unei clase** $\mathcal{C}$ este o structură care are un nume şi conţine trei componente:

1. o expresie fluxnomială de forma

$$E_{\mathcal{C}} = [\delta_{in_1,in_2,\ldots,in_n;s} \cdot (m_1 \oplus m_2 \oplus \ldots \oplus m_n) \cdot \rho_{out_1,out_2,\ldots,out_n;s}] \uparrow^{s}_{\otimes}$$

   unde m_i este o mulţime de '*metode*' [10] de forma

$$m_i : in_i \otimes s \longrightarrow out_i \otimes s.$$

   Aceste metode m_i sunt de două feluri:

   - **atribute**: $m_i \stackrel{def}{=} (\mathsf{I}_{in_i} \otimes \wedge^s) \cdot (\mathtt{atr_i} \otimes \mathsf{I}_s)$ cu $\mathtt{atr_i} : in_i \otimes s \longrightarrow out_i$;
   - **operaţii**: $m_i \stackrel{def}{=} \mathtt{op_i} : in_i \otimes s \longrightarrow out_i \otimes s$, care admit parametri şi pot schimba starea internă.

[10] sau '*caracteristici*' în limbajul Eiffel din [Mey88] sau '*membrii*' în C++

De notat că

$$E_{\mathcal{C}} : in_1 \oplus in_2 \oplus \ldots \oplus in_n \longrightarrow out_1 \oplus out_2 \oplus \ldots \oplus out_n.$$

2. *aserţiuni*, care pot fi condiţionale. Acestea modelează (restricţionează) comportamentul obiectelor aparţinând clasei.

3. proprietăţi observabile ale stării iniţiale a obiectelor create cu `new`[11]. Acestea pot avea sau nu parametri.

Definiţia 2.2.3 (**Clasă**) O **clasă** ce verifică *specificaţia unei clase* are un nume şi este alcătuită din:

1. o mulţime S, asociată lui s şi care reprezintă *spaţiul stărilor* pe care le poate avea un obiect al clasei şi mulţimi (notate A, B sau In_i) asociate sorturilor (notate a, b sau in_i);

2. funcţii ce interpretează/implementează atributele şi operaţiile:

$$Atr_i : In_i \times S \longrightarrow Out_i, \quad Op_i : In_i \times S \longrightarrow Out_i \times S$$

   şi care satisfac condiţiile impuse de aserţiuni [12]. Având aceste funcţii putem construi şi funcţiile

$$M_i : In_i \times S \longrightarrow Out_i \times S$$

   asociate metodelor din definiţia specificaţiilor de clasă;

3. o stare iniţială $\mathtt{s}_0 \in S$ care satisface condiţiile din **creation**.

Această distincţie modelată teoretic între o specificare de clasă şi o clasă ca implementare a ei corespunde distincţiei din ingineria software dintre clasele abstracte ale căror metode sunt toate doar declarate şi clasele concrete ale căror metode sunt toate implementate.

Definiţia 2.2.4 (**Obiect**) Un **obiect** aparţinând unei clase este pur şi simplu un element `s` din mulţimea S a spaţiului stărilor clasei.

[11] corespondentul în Eiffel este '*make*', iar în C++ este '*constructorul*'

[12] vom vedea mai târziu în ce mod verificăm acest lucru

Am putea defini un obiect ca fiind o pereche: un element din S împreună cu clasa din care face parte. Datorită faptului că în practică avem de-a face cu sisteme de obiecte (instanţe a diferitelor clase ale sistemului), la perechea (*valoare, clasă*) adăugăm şi un număr care să identifice în mod unic un anumit obiect, ţinând cont de faptul că acesta îşi poate modifica valoarea (starea internă memorată în conexiunea feedback-ului).

În lumea informaticii, un *obiect* trebuie să aibă două caracteristici: (1) să aibă o stare locală, accesibilă doar prin metodele obiectului şi (2) să combine structurile de date cu comporatmentul. Observăm că ambele aspecte sunt bine surprinse în modelarea teoretică de mai sus. Un obiect poate fi văzut şi ca o mică bază de date. Un lucru important este faptul că înr-un univers de obiecte *nu există o stare globală*. Din acest motiv, nu a fost suficient un doar un cadru multiplicativ, ci a fost nevoie de combinare lui cu unul aditiv.

O clasă este adeseori considerată ca o combinaţie a două aspecte: este în acelaşi timp privită ca un **tip** şi ca un **modul**. Aceasta se potriveşte cu modelul nostru:

- 'clasa ca un tip' este mulţimea stărilor interne S, ale cărei elemente sunt obiectele care aparţin clasei;
- 'clasa ca un modul' este expresia fluxnomială asociată ei, aceasta reuşind să *încapsuleze* întreaga structură a clasei.

Metodele pot fi şi ele specificate prin expresii fluxnomiale, acestea fiind potrivite pentru scheme de programe. În exemplul contului bancar, acest lucru nu poate fi exemplificat, funcţiile fiind foarte simple, însă în general metodele unei clase sunt foarte complexe. O astfel de specificare a metodelor în acelaşi cadrul al fluxnoamelor, prezintă un foarte mare avantaj pentru că ne permite să optăm între o prezentare mai generală a clasei şi pentru diferite grade de rafinare sau detaliere.

Observaţia 2.2.5 *Avantajele prezentării claselor şi obiectelor în teoria fluxnoamelor în varianta mixtă sunt:*

- *modularizarea;*
- *persistenţa unei stări interne prezente în conexiunea feedback-ului;*

- *posibilitatea exprimării atât a clasei, cât şi a metodelor în aceeaşi teorie a fluxnoamelor, optând pentru nivelul dorit de granularitate;*
- *protejarea stării interne prin accesul la ea doar prin intermediul metodelor.*

2.2.3 Implementări de metode

Pentru exemplul 'financiar' iniţial să dăm trei posibile implementări, cu diferite interpretări ale spaţiului stărilor S şi ale metodelor `bal` şi `ch`.

(1) O primă încercare este de a vedea contul din bancă ca o secvenţă de schimbări consecutive. Astfel luăm spaţiul local al stărilor $S_1 = \mathbf{Z}^*$, mulţimea secvenţelor finite de întregi. Pentru o stare arbitrară $\mathtt{s} = \langle a_0, \ldots, a_n \rangle \in S_1$, definim metodele:

$$\mathtt{s.bal} = a_0 + \ldots + a_n \quad \text{şi} \quad \mathtt{s.ch}(a) = \langle a_0, \ldots, a_n, a \rangle.$$

Este evident că este satisfăcută ecuaţia `s.ch(`a`).bal = s.bal+`a. Ca stare iniţială alegem secvenţa vidă $\langle\rangle$ din S_1 (presupunem că $\langle\rangle.\mathtt{bal} = 0$). Un exemplu de obiect aparţinând clasei este secvenţa: $\langle 2, -3 \rangle \in S_1$. 'Balanţa' contului este -1. Această implementare este destul de ineficientă pentru că implică adunarea de fiecare dată a tuturor operaţiunilor bancare făcute. Pentru client însă, care poate doar apela metodele `bal` şi `ch`, acest lucru este invizibil. În acest prim caz este memorată întreaga istorie a operaţiunilor bancare.

(2) A doua noastră implementare ţine cont de schimbări, de data acesta adunările fiind făcute pe loc, astfel că atunci când este cerută suma din cont, răspunsul este mai direct. Pentru aceasta, vom considera spaţiul local al stărilor $S_2 = \mathbf{Z}^+$. mulţimea secvenţelor nevide de întregi. Pentru un element $\mathtt{s} = \langle a_1, \ldots, a_n \rangle \in S_2$ definim:

$$\mathtt{s.bal} = a_n \quad \text{şi} \quad \mathtt{s.ch}(a) = \langle a_1, \ldots, a_n, a_n + a \rangle.$$

Această alegere verifică aserţiunea, iar un exemplu de obiect este o secvenţă oarecare nevidă de întregi, ultimul dintre întregi fiind chiar suma din cont. Ca stare iniţială putem alege secvenţa formată doar din 0, $\langle 0 \rangle \in S_2$, dar şi secvenţa $\langle 1, 0\rangle \in S_2$, ambele cazuri respectând condiţiile impuse de **creation**. În acest al doilea caz este memorată întreaga istorie a sumelor din cont.

(3) Oferim o a treia implementare care pur şi simplu are ca spaţiu al stărilor mulţimea $S_3 = \mathbf{Z}$ a numerelor întregi. Pentru o stare $\mathtt{s} \in S_3$ definim

$$\mathtt{s.bal} = \mathtt{s} \quad \text{şi} \quad \mathtt{s.ch}(a) = \mathtt{s} + a.$$

Aceasta implementare este cea mai eficientă, în starea internă ţinându-se minte suma actuală de bani. Starea iniţială este evident 0.

2.2.4 Apeluri de metode

Obiecte comunică şi se coordonează reciproc trimiţându-şi mesaje. Acest lucru înseamnă că îşi apelează reciproc metodele publice ale claselor din care fac parte. Să prezentăm în continuare o semantică *mixtă* pentru expresii fluxnomiale mixte şi mecanismul apelării unei metode a unei clase modelate ca mai înainte.

Definiţia 2.2.6 (**O semantică mixtă**) Modelul este oarecum asemănător cu cel al reţelelor Petri (v. [Rei85]). Semantica unei expresii fluxnomiale mixte cu care vom lucra presupune că toate firele reţelei asociate expresiei sunt *buffere de capacitate 1*. Celulele şi constantele au interfeţe mixte, de exemplu $in = (s \oplus s') \otimes t \otimes (u \oplus u')$. Asemănător unei reţele Petri, celulele consumă valori din bufferele firelor de intrare şi trimit valori în bufferele firelor de ieşire. Celulele nu consumă valori din toate intrările. Ele ţin cont de structura interfeţei, astfel că dacă avem un produs de sorturi, trebuie să avem valori în toate bufferele asociate, iar dacă avem o sumă, trebuie să avem valori măcar într-unul din 'termeni'. Un lucru asemănător se întâmplă şi la ieşiri. De exemplu, pentru ca o celulă cu interfaţa de intrare in de mai sus să intre în funcţiune, trebuie ca să avem valori în firele s sau s', în t şi în t sau t'.

Constanta de distribuţie din exemplul prezentat,

$$\delta_{1,z;s} : (1 \oplus z) \otimes s \longrightarrow (1 \otimes s) \oplus (z \otimes s),$$

are nevoie de o dată pe canalul z, să zicem, şi de o dată pe canalul s. Vor fi consumate întregul de la intrarea a doua a clasei şi data din feedback. Vor fi produse două copii ale acestora care vor fi trimise în bufferele z şi s din 'termenul' $(z \otimes s)$ de după $\oplus$-ul din interfaţa de ieşire. Aceste date vor fi preluate de metode şi se continuă în acest fel până când nici o celulă nu mai poate executa nici o transformare.

De remarcat că 1 este interpretat prin $D_1 = \{()\}$, deci pe canalul de sort 1 nu se trimite nici o dată efectivă, ci doar o incitare pentru a activa o celulă sau alta.

În cazul reţelelor date de expresiile asociate claselor, la iniţializare toate bufferele sunt goale, în afară de cel al feedback-ului, unde punem o valoare care să satisfacă condiţia din **creation**. După fiecare apel de metodă, fiecare buffer va fi gol, cu excepţia feedback-ului, aceasta fiind un invariant al sistemului între două 'folosiri' a clasei.

Diferenţele faţă de reţelele Petri [13] sunt: utilizarea unor date de un anumit tip în loc de puncte şi îmbinarea mediului aditiv cu cel multiplicativ, faţă de reţelele Petri care au doar un mediu de tip multiplicativ.

Definiţia 2.2.7 (**Apelul unei metode**) Pentru a apela metoda

$$m_i : in_i \otimes s \longrightarrow out_i \otimes s$$

a obiectului s cu parametrul $a \in In_i$, aplicăm perechii (a, s) funcţia asociată metodei:

$$s.m_i(a) \stackrel{def}{=} M_i(a, s) \in Out_i \times S.$$

Acest apel, datorită faptului că acum lucrăm cu clase care sunt exprimate prin expresii fluxnomiale, se face prin următorul mecanism:

> Parametrul a este trimis pe canalul de intrare in_i. Prin semantica constantei de distributivitate $\delta_{in_1,in_2,\ldots,in_n;s}$, este extrasă şi valoarea s, care era memorată în conexiunea feedback-ului, iar ambele valori (a, s) ajung pe intrările metodei m_i. Este aplicată funcţia $M_i(a, s) = (b, s')$ şi apoi (b, s') este preluată de constanta de reducere $\rho_{out_1,out_2,\ldots,out_n;s}$ care va trimite valoarea b la ieşirea out_i şi va trimite valoarea obţinută s' în bufferul de capacitate 1 al feedback-ului. În cazul în care metoda m_i 'conţine' un atribut

[13] Corespondenţa elementelor constituente este: fire - locuri, celule - tranziţii.

atr_i, datorită constantei de ramificare $\Re^s$, o copie a lui s intră împreună cu a în funcţia Atr_i, iar cealaltă merge spre ieşirea metodei, obţinând perechea $(Atr_i(a,s), s) \in Out_i \times S$. În acest caz, valoarea internă s rămâne neschimbată, aceasta fiind ceva natural, căci prin extragerea valorii unui atribut nu modificăm starea obiectului.

Definiţia 2.2.8 (Pas de tranziţie) Fie două obiecte s şi s' aparţinând aceleiaşi clase $\mathcal{C}$. Un **pas de tranziţie**

$$s \overset{x}{\underset{y}{\Longrightarrow}} s'$$

unde $x \in In = In_1 + In_2 + \ldots + In_n$ este o intrare şi $y \in Out = Out_1 + Out_2 + \ldots + Out_n$ este o ieşire, este definit după cum urmează. Pentru $x = \langle i, a\rangle \in In$ cu $a \in In_i$, avem:

$$y = \langle i, b\rangle, \text{ unde } M_i(a,s) = (b, s') \in Out_i \times S.$$

Mai precis, intrând în structura metodei M_i, putem avea următoarele două posibilităţi, după tipul metodei (atribut sau operaţie):

$$s \overset{\langle i,a\rangle}{\underset{\langle i,b\rangle}{\Longrightarrow}} s \qquad\qquad s \overset{\langle i,a\rangle}{\underset{\langle i,b\rangle}{\Longrightarrow}} s'$$

dacă $Atr_i(a,s) = b \in Out_i$ dacă $Op_i(a,s) = (b, s') \in Out_i \times S$.

După cum vedem, în cazul atributelor starea internă rămâne neschimbată. Faptul că atributele admit la rândul lor parametri nu este ceva anormal. De exemplu, pentru o clasă ce specifică datele personale ale unui om, dacă starea internă memorează data naşterii, atributul `vârstă` are nevoie de un parametru care este data curentă pentru a calcula etatea persoanei.

De asemenea, observăm că dacă trimitem o dată pe intrarea i, după apelul metodei i, vom obţine o dată pe ieşirea i, deci avem o relaţie clară între intrări, ieşiri şi metode.

Definiţia 2.2.9 (**Semantica operaţională** $\mathcal{O}(p)$) Pentru un obiect p cu starea locală $s \in S$,

$$\mathcal{O}(p)(\langle x_1, x_2, \ldots, x_n \rangle) = y \ \Leftrightarrow \begin{cases} \text{există obiectele } s_1, \ldots, s_n \in S \text{ şi} \\ \text{ieşirile } y_1, \ldots, y_{n-1} \in Out \text{ a. î.} \\ s \overset{x_1}{\underset{y_1}{\Longrightarrow}} s_1 \overset{x_2}{\underset{y_2}{\Longrightarrow}} \ldots \overset{x_{n-1}}{\underset{y_{n-1}}{\Longrightarrow}} s_{n-1} \overset{x_n}{\underset{y}{\Longrightarrow}} s_n \end{cases}$$

Astfel, scrierea în stilul programării orientate pe obiecte îşi va afla o exprimare matematică dată de semantica definită mai sus:

$$p.m_{i_1}(a_1).m_{i_2}(a_2)\ldots m_{i_n}(a_n) \overset{def}{=} \mathcal{O}(p)(\langle\langle i_1, a_1\rangle, \langle i_2, a_2\rangle, \ldots \langle i_n, a_n\rangle\rangle)$$

Observăm că în acest fel putem să scriem o expresie de tipul:

`s.ch(`a`).bal.ch(`b`).bal`, sau, şi mai riguros, `s.ch(`a`).bal().ch(`b`).bal()`,

ceea ce constituie un mod mai bun şi mai flexibil de apelare a metodelor faţă de cel folosit în practică (d.ex. C++).

Revenim să explicăm ce înseamnă că metodele unei clase satisfac condiţiile impuse de aserţiuni. Pentru aserţiunea din exemplul 2.2.1, `s.ch(`a`).bal = s.bal+`a, luăm două obiecte cu starea internă `s` şi aplicăm metodele, având ieşiri pe care le combinăm după formula dată. Pentru ca aserţiunea sa fie validă, trebuie ca orice `s` din spaţiul stărilor să fie soluţie a ecuaţiei aserţiunii (ţinând cont de eventuale precondiţii care pot restricţiona spaţiul stărilor).

Un aspect bun este faptul că este respectat un principiu foarte sănătos al obiect-orientării (v. [Mey88]), anume acela care recomandă ca accesul la atributele unei clase să fie făcut prin funcţii (de tip `set(...)` şi `get(...)`), iar nu direct (d.ex. `s.bal = 3;` cum este posibil în C++). Din motive de securitate, e bine ca starea internă să fie privată, accesul la ea fiind făcut prin metode publice. Şi dacă am ajuns să vorbim despre metode publice, trebuie să vorbim şi despre metode private. Deşi noi nu vom face această distincţie, totuşi este destul de simplu a modela acest lucru, prin marcarea intrărilor şi a ieşirilor corespunzătoare lor cu `public` sau `private` şi restricţionarea accesului metodelor externe clasei doar la cele marcate cu `public`. Metodele private vor putea fi apelate doar de metode din aceeaşi clasă cu ele. Trebuie să specificăm că lumea obiect-orientării este foarte complexă şi implică o mulţime de fenomene care sunt destul de greu de prins toate într-un acelaşi cadru teoretic. Simplificând lucrurile, căpătăm intuiţie, pierzând însă din vedere anumite fenomene, fapt care nu ne deranjează datorită caracterului de exerciţiu al prezentului demers teoretic.

Un alt aspect demn de menţionat este faptul că putem să trimitem ca parametru şi un obiect al unei clase. În practică, un obiect al unei clase specifică o zonă din memorie unde se găseşte starea sa internă şi un pointer către o zonă unde sunt memorate operaţiile specifice clasei din care face parte. Pe scurt, un obiect al unei clase poate fi trimis ca parametru unei metode, sortul canalului său de intrare fiind un anumit produs de sorturi.

Observaţia 2.2.10 *Având două expresii pentru specificaţii de clase* E_1 *şi* E_2*, nici una din expresiile*

$$E_1 \otimes E_2,\ E_1 \oplus E_2,\ E_1 \cdot E_2,\ E_1 \uparrow_{\otimes},\ E_1 \uparrow_{\oplus}$$

nu are o interpretare naturală în lumea obiect-orientării [14]*. Eventual fenomenul moştenirii s-ar putea exprima cu ajutorul expresiilor, pentru că, privind într-un mod simplist, o clasă care moşteneşte altă clasă, adaugă propriile metode la metodele clasei pe care o moşteneşte. Această adăugare ar putea fi simulată cu un* $\oplus$*.*

O altă utilizare a cadrului mixt MixNA a fost făcută în [Ste96b], unde este descris un limbaj numit **FEST** (Flownomial Expressions and System Tasks). Acesta integrează module de control (v. modelul aditiv din secţiunea 1.4.1) şi module reactive (v. modelul multiplicativ din secţiunea 1.4.2). Deşi atât abordarea noastră, cât şi **FEST**-ul au o semantică mixtă, ele nu sunt asemănătoare.

[14] În cadrul mixt, forma normală $nf(E_1)$ nu prea are sens, de aceea nu o studiem

2.3 2-cuvinte

În această secţiune vor fi prezentate noţiunile de 2-cuvânt, 2-limbaj, 2-automat etc. Acest studiu are drept scop o mai bună înţelegere a reţelelor Petri şi a fost iniţiat sub conducerea domnului profesor Gheorghe Ştefănescu. Ceea ce este prezentat aici este doar un început, iar dacă timpul şi inspiraţia ne va ajuta, vom dezvolta o formă mai 'rotundă' a subiectului început aici.

După cum ştim, o lume tridimensională este mai expresivă şi mai frumoasă decât o lume bidimensioanală. Preţul acestei bogăţii semantice este plătit printr-o mai mare complexitate. Limbajele formale lucrează cu cuvinte care sunt secvenţe de 'litere' dintr-un alfabet dat. De exemplu, avem cuvântul *casă*. Să îl scriem: $\rightarrow\boxed{c}\rightarrow\boxed{a}\rightarrow\boxed{s}\rightarrow\boxed{\breve{a}}\rightarrow$. O idee ar fi să construi un *cuvânt 2-dimensional*:

$$\begin{array}{ccccc} \boxed{c} & \boxed{a} & \boxed{s} & \boxed{\breve{a}} & \boxed{,} \\ \boxed{d} & \boxed{r} & \boxed{a} & \boxed{g} & \boxed{\breve{a}} \\ \boxed{c} & \boxed{a} & \boxed{s} & \boxed{\breve{a}} & \boxed{!} \end{array}$$

Această idee este la fel de bătrână ca şi aceea a cuvintelor încrucişate, dar scopul nostru nu este acela de a da o prezentare matematică pentru aceste jocuri de perspicacitate. Ţinta noastră este undeva mai sus. Informatica teoretică a arătat că modelarea fenomenelor sale adeseori necesită mai mult decât 'simple' ordini. În special concurenţa a generat multe modele non-convenţionale (vezi de exemplu în [Gup94] spaţiile Chu cu structura lor algebrică foarte săracă, dar cu o mare putere şi varietate semantică).

Atomii necesari pentru a construi universul nostru sunt elemente având patru interfeţe (*nord, sud, vest,* şi *est*). *Nordul* şi *sudul* le presupunem pe o axă verticală, iar *vestul* şi *estul* pe o axă orizontală. Toate următoarele noastre construcţii vor avea două dimensiuni, de aceea le vom preceda de un *2-*.

Definiţia 2.3.1 Un *2-atom* este un element având două interfeţe de intrare, două interfeţe de ieşire şi o etichetă (care poate fi şi cuvântul vid ε):

$$a : N \times W \longrightarrow S \times E \quad \text{reprezentat} \quad \rightarrow\boxed{a}\rightarrow$$

Un *2-alfabet* este o colecţie de 2-atomi cu etichete distincte. De exemplu,

$$\Sigma 2 = \left\{ \boxed{a_1}, \boxed{a_2}, \ldots \right\}$$

Definiţia 2.3.2 Un *2-cuvânt* peste un 2-alfabet $\Sigma 2$ este o construcţie dreptunghiulară ce foloseşte 2-atomi din $\Sigma 2$. Un 2-cuvânt are două dimensiuni (p, q), unde p şi q sunt întregi nenegativi. Scriem $\varpi : pN \times qW \to pS \times qE$ şi spunem că ϖ este un (p, q)-cuvânt. p este numit dimensiunea orizontală, iar q dimensiunea verticală.

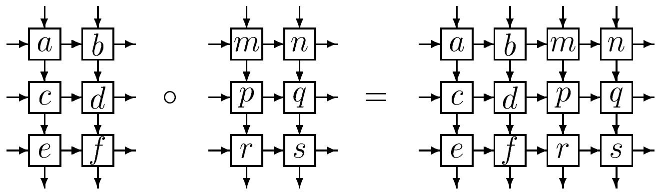
O definiţie recursivă este:

1. un 2-atom $a : N \times W \longrightarrow S \times E \in \Sigma 2$ este un $(1, 1)$-cuvânt;
2. (compunere orizontală $\circ$) dacă ϖ_1 este un (p_1, q)-cuvânt şi ϖ_2 este un (p_2, q)-cuvânt, atunci şi $\varpi_1 \circ \varpi_2$ este un $(p_1 + p_2, q)$-cuvânt;
3. (compunere verticală $\bullet$) dacă ϖ_1 este un (p, q_1)-cuvânt şi ϖ_2 este un (p, q_2)-cuvânt, atunci şi $\varpi_1 \bullet \varpi_2$ este un $(p, q_1 + q_2)$-cuvânt;
4. toate 2-cuvintele se obţin aplicând de un număr finit de ori cei trei paşi de mai sus.

Mulţimea tuturor 2-cuvintelor peste $\Sigma 2$ se notează prin $\Sigma 2^{\circ,\bullet}$.

Un *limbaj* de 2-cuvinte este o submulţime a lui $\Sigma 2^{\circ,\bullet}$.

Un exemplu de compunere orizontală a două $(2, 3)$-cuvinte este dată mai jos:



Următorul pas este acela de a extinde operaţiile pe 2-cuvinte la limbaje de 2-cuvinte şi să obţinem o structură algebrică elegantă peste aceste limbaje.

Înzestrăm mulţimea părţilor lui $\Sigma 2^{\circ,\bullet}$ cu următoarele operaţii şi constante:

$$(\mathcal{P}(\Sigma 2^{\circ,\bullet}), +, \circ, \bullet, ^{\circledast}, ^{*}, \varepsilon^{\circ}, \varepsilon^{\bullet})$$

Fie $L_1, L_2 \subseteq \Sigma 2^{\circ,\bullet}$ două 2-limbaje. Definim:

$$\begin{aligned}
&0 \stackrel{def}{=} \emptyset;\\
&\varepsilon^\circ \stackrel{def}{=} \{(0,1)\text{-cuvântul cu eticheta } \varepsilon\};\\
&\varepsilon^\bullet \stackrel{def}{=} \{(1,0)\text{-cuvântul cu eticheta } \varepsilon\};\\
&L_1 + L_2 \stackrel{def}{=} L_1 \cup L_2;\\
&L_1 \circ L_2 \stackrel{def}{=} \{\, \varpi_1 \circ \varpi_2 \mid \varpi_1 \in L_1 \text{ şi } \varpi_2 \in L_2 \text{ cu aceeaşi dimensiune verticală}\};\\
&L_1 \bullet L_2 \stackrel{def}{=} \{\, \varpi_1 \bullet \varpi_2 \mid \varpi_1 \in L_1 \text{ şi } \varpi_2 \in L_2 \text{ cu aceeaşi dimensiune orizontală}\};\\
&L_1^{\circledast} \stackrel{def}{=} \varepsilon^\circ \cup L_1 \cup (L_1 \circ L_1) \cup (L_1 \circ L_1 \circ L_1) \cup \ldots\\
&L_1{}^* \stackrel{def}{=} \varepsilon^\bullet \cup L_1 \cup (L_1 \bullet L_1) \cup (L_1 \bullet L_1 \bullet L_1) \cup \ldots
\end{aligned}$$

Acum să construim *2-expresiile regulate* peste $\Sigma 2$, notate *Reg2Exp*($\Sigma 2$), în mod recursiv:

$$E \;::=\; E + E \mid E \circ E \mid E \bullet E \mid E^{\circledast} \mid E^* \mid 0 \mid \varepsilon^\circ \mid \varepsilon^\bullet \mid \boxed{a} \in \Sigma 2$$

Fiecărei 2-expresii regulate E îi putem în mod unic asocia un 2-limbaj $|E| \subseteq \Sigma 2^{\circ,\bullet}$ prin inducţie structurală:

$$\begin{aligned}
&|0| \stackrel{def}{=} 0, \quad |\varepsilon^\circ| \stackrel{def}{=} \varepsilon^\circ, \quad |\varepsilon^\bullet| \stackrel{def}{=} \varepsilon^\bullet, \quad |\boxed{a}| \stackrel{def}{=} \{\boxed{a}\},\\
&|E_1 + E_2| \stackrel{def}{=} |E_1| + |E_2|,\\
&|E_1 \circ E_2| \stackrel{def}{=} |E_1| \circ |E_2|, \quad |E_1 \bullet E_2| \stackrel{def}{=} |E_1| \bullet |E_2|,\\
&|E_1^{\circledast}| \stackrel{def}{=} |E_1|^{\circledast}, \quad |E_1{}^*| \stackrel{def}{=} |E_1|^*.
\end{aligned}$$

Putem defini o operaţie de *platizare* # peste 2-cuvinte. De exemplu,

$$\#\left(\begin{array}{cc} \boxed{a} & \boxed{b} \\ \boxed{c} & \boxed{d} \end{array}\right) = \{\, abcd,\ acbd \,\}.$$

Formal, platizarea unui (p,q)-cuvânt ϖ este:

$$\#(\varpi) = \{\, label(\varpi_{ts}) \mid ts \text{ este o sortare topologică a lui } ([p] \times [q], \preceq) \,\},$$

unde $(N \times N, \preceq)$ cu $(i,j) \preceq (i', j')$ ddacă $(i \leq i'$ şi $j \leq j')$.

Teoria automatelor joacă un rol important în aria limbajelor formale. Mecanisme finite sunt în stare să genereze/accepte diferite clase de limbaje infinite. O 'unealtă' similară pentru a produce 2-limbaje ar fi de mare folos în cadrul nostru teoretic.

Un automat are stări, tranziţii etichetate şi un 'control' care se află undeva în reţea. Folosind un automat, a accepta un cuvânt şi a genera un cuvânt, sunt două feţe ale aceleiaşi monede. A face o tranziţie într-un automat ar putea însemna să consumăm o literă a unui cuvânt care este egală cu eticheta tranziţiei sau ar putea însemna să adăugăm eticheta la un cuvânt în formare. Acest proces trebuie să se termine într-o stare finală.

În cazul nostru bidimensional, a genera un 2-cuvânt necesită un mecanism de a pune 2-atomi într-o anumită ordine pentru a obţine o construcţie rectangulară. Un *2-automat* care ar face aceasta ar avea stări (inclusiv o stare iniţială şi o submulţime de stări finale), tranziţii etichetate de două tipuri între stări şi un 'control'. Grafic, avem noduri pentru stări şi săgeţi etichetate drept tranziţii folosind linie punctată pentru tranziţii *orizontale* şi linie normală pentru tranziţii *verticale*.

Cum construim un 2-cuvânt folosind un 2-automat? Ca şi într-un joc de puzzle, începând din colţul din stânga-sus şi punând 2-atomi la sfârşitul unei coloane sau rând. La un pas, dacă alegem să facem o tranziţie orizontală a-etichetată, trebuie să punem un 2-atom a-etichetat la sfârşitul următoarei coloane faţă de coloana ultimului 2-atom adăugat. Dual, dacă alegem să facem o tranziţie verticală a-etichetată, trebuie să punem un 2-atom a-etichetat la sfârşitul următorului rând faţă de rndul ultimului 2-atom adăugat. Ne oprim aunci când am obţinut un 2-cuvânt dreptunghiular şi 'controlul' se găseşte într-o stare finală. Deci, un 2-automat poate genera un 2-limbaj.

Un rezultat important ar fi o teoremă de tip Kleene care să afirme că familia 2-limbajelor generate de 2-automate este aceeaşi cu familia 2-limbajelor asociate 2-expresiilor regulate.

3 O implementare a fluxnoamelor

În cele ce urmează, vom oferi o scurtă prezentare a unei implementări a expresiilor fluxnomiale în Java.

Acest proiect experimental a fost realizat în toamna-iarna anului 1999, împreună cu Cornelia Oancea[15] sub îndrumarea domnului profesor Gheorghe Ştefănescu.

Proiectul are două părţi: cazul interfeţelor sortate şi cazul interfeţelor monosortate sau nesortate. În ambele situaţii, acesta permite:

1. introducerea într-un mod interactiv a unei expresii fluxnomiale, folosind elemente din X şi T;
2. verificarea tipurilor interfeţelor subexpresiilor şi a compatibilităţii acestora în aplicarea diferitelor operaţii;
3. construirea arborelui asociat expresiei fluxnomiale;
4. normalizarea expresiei introduse;
5. verificarea faptului că două expresii sunt echivalente sau bisimilare (modulo BNA).

3.1 Structura generală a proiectului

Limbajul de programare folosit a fost Java, versiunea 1.2. Proiectul se prezintă sub forma unei *miniaplicaţii Java (applet)* (în continuare vom prefera termenul de *applet* celui de *miniaplicaţie*).

Un applet este un program Java (de obicei de dimensiuni mici), creat pentru a fi folosit în sistemul World Wide Web. Applet-urile necesită un program de vizualizare extern, care poate fi ori un *browser Web*, ori un program specializat de vizualizare (*applet viewer*). Reamintim că un *browser Web* este un program ce poate accesa informaţii de la diferite adrese de pe Internet. Informaţiile din reţeaua Internet sunt structurate în fişiere cu extensia *.html*.

[15] studentă la Studii Aprofundate, secţia Informatică Teoretică, în acelaşi an

HTML este un acronim pentru *Hyper Text Markup Language*. Într-un fişier de tip HTML se pot include texte, imagini, sunete şi (ce ne interesa pe noi) applet-uri. Pentru a putea însă executa aceste applet-uri, avem nevoie de browsere care 'ştiu' să interpreteze clasele Java. Exemple de asemenea browsere sunt: Netscape Navigator (începând cu versiunea 2.0) şi Internet Explorer (începând cu versiunea 3.0).

Un applet inclus într-o pagina Web poate fi executat de oricine este conectat la Internet şi are un browser ce poate interpreta Java. Un aspect important ce decurge de aici este *portabilitatea*, adică proprietatea unui produs software de a putea fi executat pe calculatoare diferite, cu sisteme de operare diferite. O tendinţă actuală importantă în lumea informaticii este încercarea de unificare a reprezentării informaţiei şi de operare asupra ei. Formatul HTML al 'paginilor' Web şi applet-urile Java sunt 'corifeii' acestei dorinţe de universalizare.

Adresa de pe Internet unde se găseşte şi poate fi rulat proiectul acesta este:

`http://funinf.cs.unibuc.ro/∼alstef/Flow/Flow.html`

Applet-ul a fost numit **Flow**. El oferă o interfaţă grafică prietenoasă, este uşor de utilizat, este portabil şi prin modularizarea dată de caracteristicile programării orientate pe obiecte permite potenţiale dezvoltări şi îmbunătăţiri.

Appletul **Flow** este inserat într-o pagină HTML.

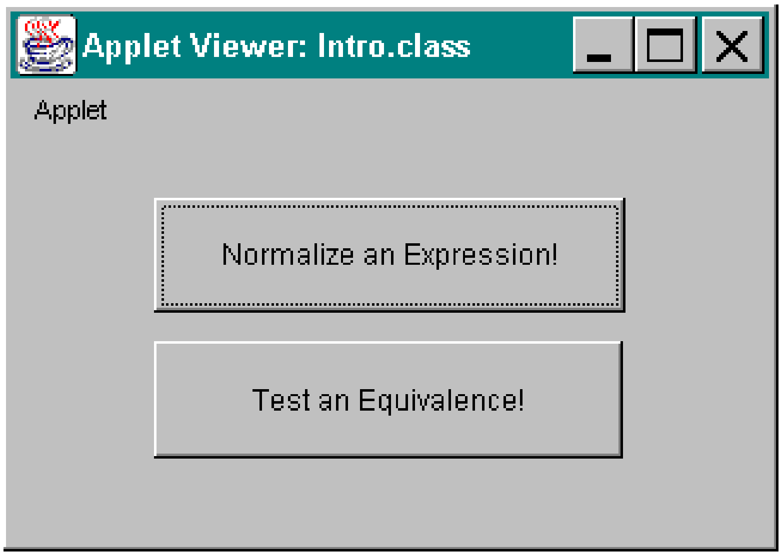
El are trei opţiuni:

- normalizarea unei expresii fluxnomiale - cazul interfeţelor sortate
- testarea echivalenţei a două expresii fluxnomiale - cazul interfeţelor sortate
- normalizarea unei expresii fluxnomiale - cazul interfeţelor nesortate (această opţiune are partea grafică în lucru)

În cazul interfeţelor nesortate, proiectul nu este la fel de dezvoltat ca în cazul sortat, dar este foarte asemănator cu acesta. Avantajul cazului nesortat este faptul ca permite introducerea unor expresii cu interfeţe de dimensiuni mari, de exemplu putem introduce o variabilă $x \in X(12028, 12342)$, fiind însă aproape imposibil în celălalt caz să introducem 12028 de sorturi pentru intrări şi alte 12342 de sorturi pentru ieşiri. Diferenţa este asemănătoare cu diferenţa dintre reprezentarea numerelor în baza 1 (cu alte cuvinte: 'biluţe pe abac') şi cea în baza 10. Din punctul de vedere al implementării pe calculator însă, diferenţele nu sunt esenţiale.

În continuare ne vom ocupa de cazul sortat, celălalt caz fiind similar.

Iată prima 'invitaţie' a proiectului făcută utilizatorului:



Se poate opta între normalizarea unei expresii fluxnomiale şi testarea echivalenţei a două expresii. Spunem că două expresii fluxnomiale sunt *echivalente* dacă sunt bisimilare via bijecţii. Intuitiv, reţelele/hipergrafurile asociate celor două expresii sunt izomorfe.

Utilizarea aplicaţiei este uşoară, iar interfaţa este prietenoasă:

- Dacă am ales să aducem o expresie la forma ei normală, se va deschide o fereastră care permite introducerea expresiei. Funcţionalităţile ferestrelor de dialog le vom descrie puţin mai târziu. După introducerea expresiei, se va apăsa butonul `Compute`, care va deschide o fereastră în care va fi reprezentată grafic forma normală calculată.
- Dacă am ales să testăm o echivalenţă, suntem invitaţi să introducem expresiile (cărora le vom vedea şi formele normale). Simpla apăsare a

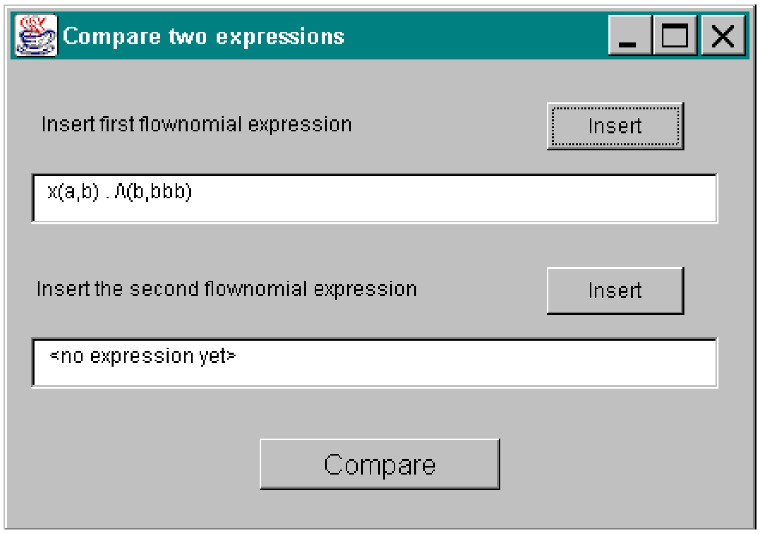
butonului `Compare` ne va oferi unul dintre răspunsurile:
`'There is an equivalence between these expressions'` sau
`'There is no equivalence between these expressions'`

Deşi aparent pare simplu, implementarea a necesitat destul de mult efort.

3.2 Descrierea ferestrelor proiectului

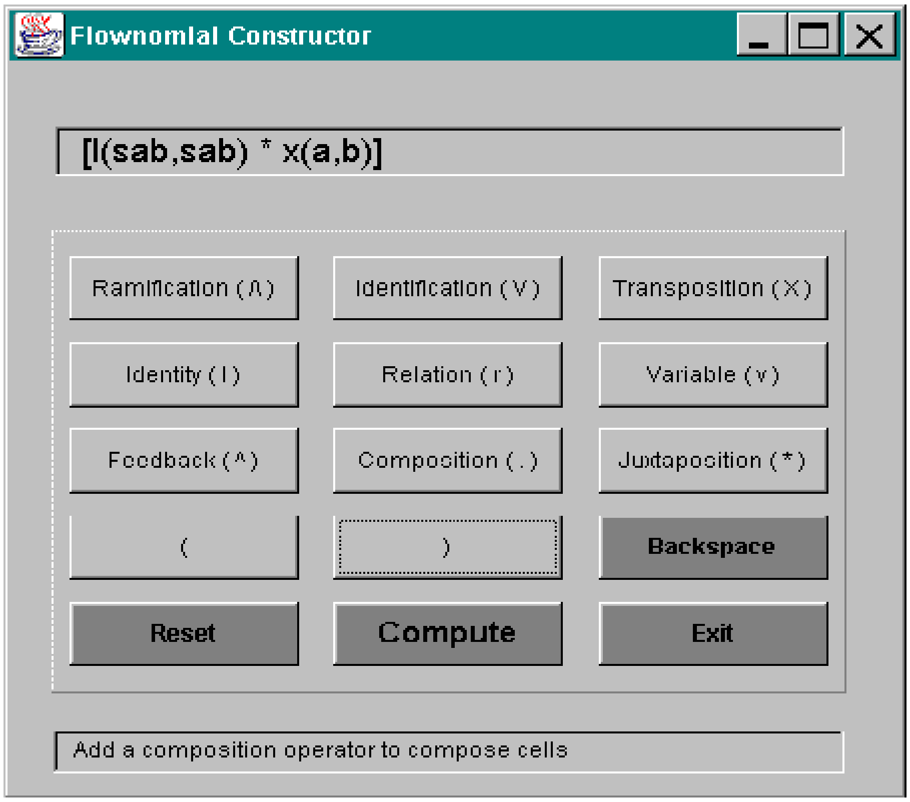
În continuare să descriem mai în detaliu ferestrele de dialog de care dispune applet-ul **Flow**:

- Fereastra 'Comparison' ne ajută să inserăm cele două expresii pe care dorim să le comparăm:



  Utilizarea ei este evidentă.

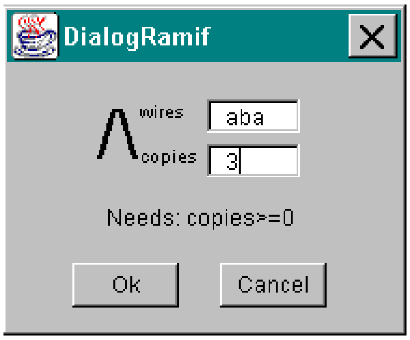
- Fereastra 'Flow Constructor' ne permite introducerea unei expresii fluxnomiale, calculul şi desenarea formei sale normale:

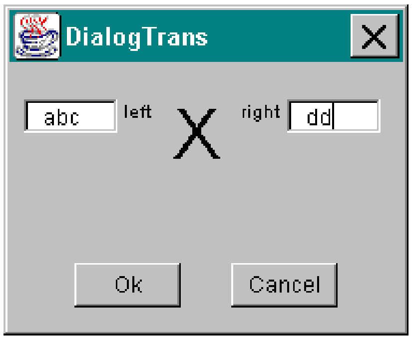


Interfaţa grafică este asemănătoare cu cea a unui calculator de buzunar, iar utilizarea ei este aproape la fel de uşoară. Probabil mai grea va fi interpretarea rezultatelor de către un utilizator neavizat. Observăm că avem la dispoziţie paranteze, constante de tip ramificare, transpoziţie, identificare, identitate, variabile, relaţii (sortate), operaţii de juxtapunere, compunere şi feedback. Avem posibilitatea de a şterge ultimul element introdus (`Backspace`), sau întreaga expresie introdusă (`Reset`). Apăsarea butonului `Compute` duce la calculul formei normale. De observat că expresia $(\mathsf{I}_{sab} \star x)$ e afişată în mod text, elementele având în paranteză interfeţele de intrare şi ieşire.

- Fereastra 'Error' ne afişează diverse mesaje de eroare de tipul: "o compunere nu poate apărea prima într-o expresie", "există paranteze rămase deschise", "incompatibilitatea interfeţelor la compunere" şi alte asemenea.

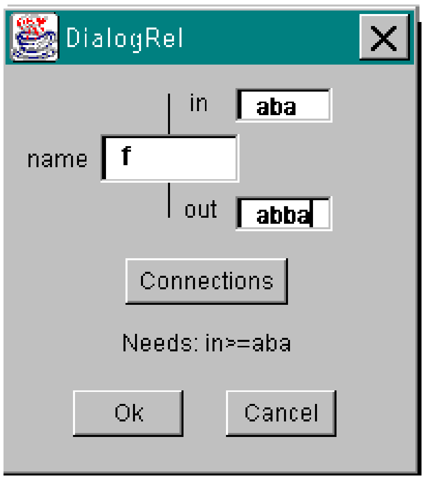
- Fereastrele pentru introducerea constantelor. De exemplu, iată căsuţele de dialog pentru ramificare şi transpoziţie:





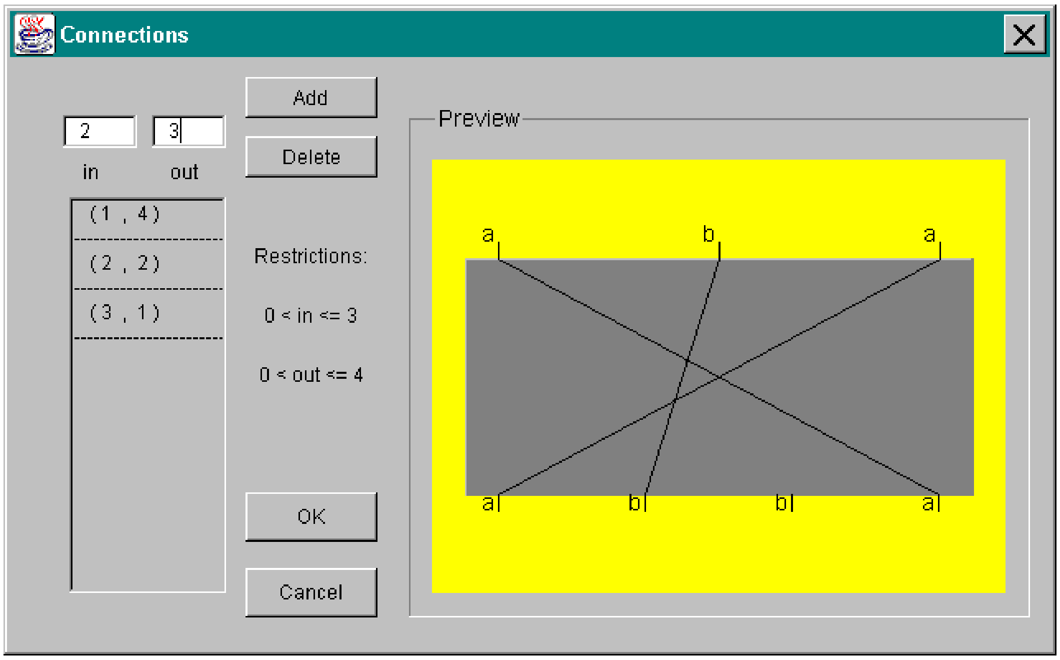
Putem avea şi resctricţii, exprimate după cuvântul `Needs`.

- Fereastra pentru introducerea atributelor relaţiei sortate:



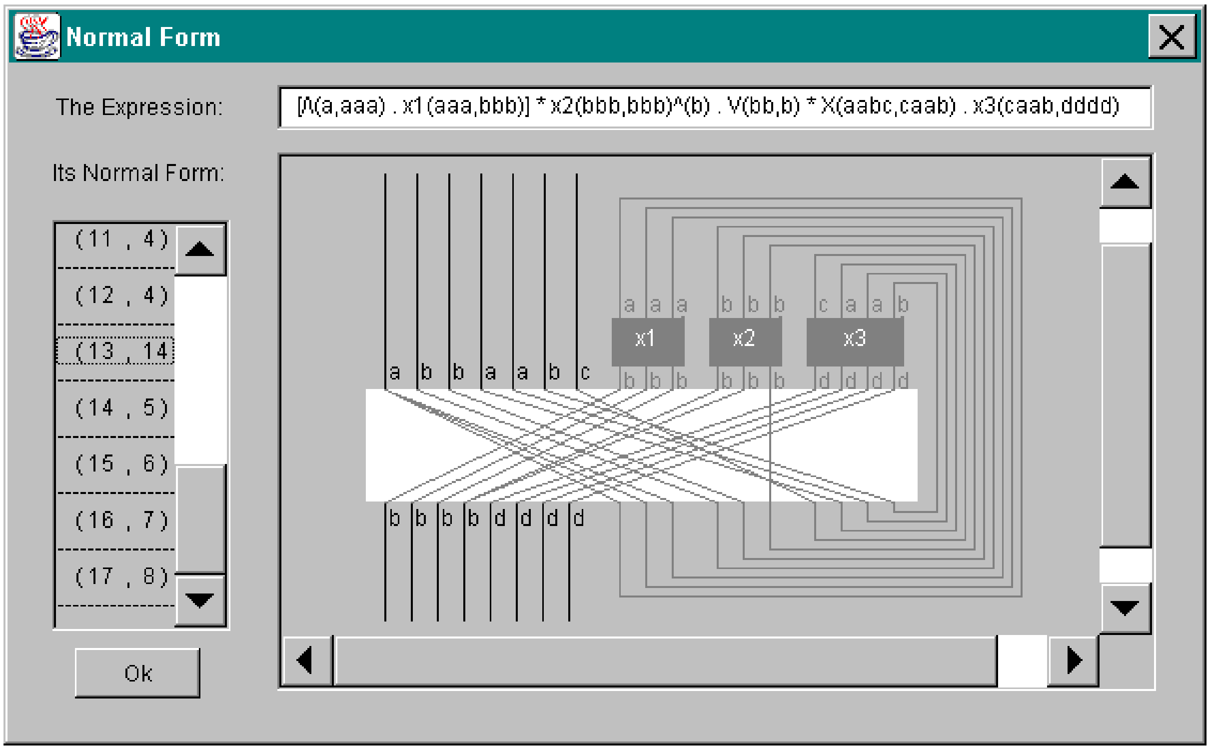
După introducerea numelui şi interfeţelor de intrare şi de ieşire, trebuie apăsat butonul `Connections` pentru a da perechile relaţiei. După aceasta, datele introduse se validează apăsând `Ok` sau se invalidează cu `Cancel`.

- Fereastra 'Relation Constructor' permite introducerea perechilor relaţiei sortate:



Intrările şi ieşirile conectate trebuie să aibă acelaşi sort, altfel primim un mesaj de eroare. Observăm că lista perechilor este uşor de modificat, putând atât să adăugăm în ea, cât şi să ştergem din ea. Pentru a şterge o pereche, scriem perechea şi apăsăm butonul `Delete`. Dacă dăm dublu-click pe o pereche din listă, aceasta va fi afişată în câmpurile de introducere `in` şi `out` şi o vom putea şterge (`Delete`) sau îi putem modifica una din componente şi apoi perechea nou obţinută să o adăugăm (`Add`). Reprezentarea grafică a relaţiei ne oferă o imagine în timp real a stadiului în care se află relaţia.

- fereastra 'Normal Form' ne prezintă grafic forma normală a expresiei introduse, listând în stânga ferestrei şi toate perechile relaţiei asociate formei normale:



În exemplul de mai sus, pentru expresia

$$E = (\wedge_3^a \cdot x1) \star x2 \uparrow^b \cdot \vee_b^2 \star {}^{aab}\mathsf{X}^c \cdot x3.$$

cu $x1 : aaa \longrightarrow bbb$, $x2 : bbb \longrightarrow bbb$ şi $x3 : caab \longrightarrow dddd$ este reprezentată grafic forma normală $nf(E) = (x, f)$ cu

$$\begin{aligned} x &= x1 \star x2 \star x3 \text{ şi} \\ f &= \{\ (1,9),(1,10),(1,11),(2,12),(3,13),(4,16),(5,17),(6,18), \\ &\quad (7,15),(8,1),(9,2),(10,3),(11,4),(12,4),(13,14),(14,5), \\ &\quad (15,6),(16,7),(17,8)\ \}. \end{aligned}$$

Pentru o înţelegere mai bună a rezultatelor, trebuie să menţionăm că expresia care este deasupra imaginii nu este în formă normală, ci este expresia iniţială şi ea este scrisă în mod text, fiecare element al ei având în paranteze interfeţele (pentru a se observa că ele se potrivesc). De exemplu, transpoziţia ${}^{aab}\mathsf{X}^c$ apare scrisă `X(aabc,caab)` şi nu `X(aab,c)`, iar $\vee_b^2$ apare scris `V(bb,b)`.

3.3 Diferite aspecte ale proiectului

Din punct de vedere calitativ, programarea calculatorului este inferioară muncii de cercetare teoretică, însă din punct de vedere cantitativ, cele 4000 de linii de cod ale proiectului sunt într-o anumită măsură sugestive.

Pentru a avea şi un exemplu de aplicare a applet-ului pe varianta de testare a unei echivalenţe, obţinem că următoarele expresii specifică aceeaşi reţea, adică formele lor normale sunt similare via bijecţii:

$$E_1 = ((x \star z) \cdot {}^{b}\mathsf{X}^{c} \cdot y \cdot {}^{dd}\mathsf{X}^{e}) \uparrow^{dd} \text{ şi}$$

$$E_2 = ((x \star \mathsf{I}_c) \cdot {}^{b}\mathsf{X}^{c} \cdot y \cdot (z \star \mathsf{I}_e) \cdot {}^{c}\mathsf{X}^{e}) \uparrow^{c} .$$

unde $x : a \longrightarrow b$, $y : cb \longrightarrow dde$ şi $z : dd \longrightarrow c$.

Algoritmii folosiţi în programarea diverselor aspecte urmăresc îndeaproape contrapartea teoretică. Aducerea la forma normală se face conform construcţiei din secţiunea 1.2. În a doua anexă este prezentat codul efectiv scris în Java al acestei implementări.

Pentru testarea echivalenţei a două expresii, avem nevoie ca prima dată să le aducem pe fiecare la forma ei normală. După ce acest deziderat a fost îndeplinit, prin metoda backtracking generăm bijecţii şi testăm dacă acestea sunt simulări ale celor două forme normale. La sfârşit, afişăm mesajul corespunzător situaţiei date.

Împletirea muncii teoretice cu aplicaţiile practice a dat de cele mai multe ori roade, iar proiectul **Flow** o dovedeşte cu prisosinţă.

4 Concluzii

Această lucrare s-a dorit a fi o exemplificare a modului în care abordarea teoretică a unui fenomen poate modela aspectele practice apărute în informatică şi poate reliefa caracteristici generale pentru diverse clase de entităţi.

Cum de cele mai multe ori prefeţele şi postfeţele pot fi interschimbate şi cum amurgul are multe din luminile răsăritului, ne oprim aici cu concluziile, multe din ceea ce am fi dorit să spunem aici regăsindu-se în rezumatul de la începutul lucrării.

4.1 Mulţumiri

Privind înapoi spre realizarea acestei lucrări, gândurile şi mulţumirile mele se îndreaptă spre domnul profesor Gheorghe Ştefănescu, toate acestea neputând însă a fi exprimate într-o expresie fluxnomială fie ea complexă sau mixtă...

Mulţumiri le aduc şi părinţilor mei, pentru încrederea acordată, sprijinirea neîncetată şi dragostea arătată.

De asemenea, Corneliei, o prezenţă atât de delicată, dar atât de plină de viaţă, îi dăruiesc căldura inimii mele.

Colegului şi prietenului meu Sorin Constantinescu, împreună cu care am parcurs drumul acestor ani de Masterat, îi urez o frumuseţe de viaţă.

Părintelui meu duhovnicesc, preotul Nicolae Tănase de la parohia Valea Plopului, cu ale cărui sfaturi, iertări şi binecuvântări am mers înainte, îi rămân îndatorat întru dragoste.

O anexă separată ar fi necesară pentru numele prietenilor care direct sau indirect au contribuit la efortul depus pentru această lucrare.

În concluzie, cele de mai sus motivează pe deplin schimbarea denumirii (puţin cam reci) de *Lucrare de dizertaţie* în *Împreuna-lucrare a dizertaţiei.*

Vă mulţumesc!

Sfârşit şi lui Dumnezeu slavă!

Referinţe

[BeS94] J.A. Bergstra şi Gh. Ştefănescu. Network Algebra for Synchronous and Asynchronous Dataflow. *Logic Group Preprint Series* No. 122, Department of Philosophy, Utrecht University, 1994.

[BeS95] J.A. Bergstra şi Gh. Ştefănescu. Network algebra with demonic relation operators. Report P9509, Programming Research Group, University of Amsterdam 1995.

[BrS96] M. Broy şi Gh. Ştefănescu. The algebra of stream processing functions. Technical Report I9620 and SFB-Bericht Nr. 342/11/94 A, Department of Computer Science, Technical University Munich, May 1996.

[CaS90] V.E. Căzănescu şi Gh. Ştefănescu. Towards a new algebraic foundation of flowchart scheme theory. *Fundamenta Informaticae*, 13:171–210, 1990.

[GD94] J. Goguen şi R. Diaconescu. Towards an algebraic semantics for the object paradigm. În H. Ehrig and F. Orejas, eds., *Proceedings, Tenth Workshop on Abstract Data Types*, pg. 1-29, Springer, Lecture Notes in Computer Science, Volume 785, 1994

[GTW78] J. Goguen, J. Thatcher şi E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. În R. Yeh, ed., *Current Trends in Programming Methodology, IV*, pg. 80-149, Prentice Hall, 1978

[Gup94] V. Gupta. Chu Spaces: a model of concurrency. *Ph.D. thesis*, Stanford University, August 1994.

[Jac96] B. Jacobs. Objects and classes, co-algebraically. În B. Freitag, C. Jones, C. Lengauer and H.-J. Schek, eds. *Object-Orientation with Parallelism and Persistence*, pg. 83-103, Kluwer, 1996.

[Mey88] B. Meyer. Object-Oriented Software Construction. *Prentice Hall*, 1988.

[Nor97] P. Norton. Ghid de programare în JAVA. *Ed. Teora.* 1997.

[Rei85] W. Reisig. Petri Nets. An introduction. LNCS, Springer 1985.

[Ste90] Gh. Ştefănescu. Feedback theories (a calculus for isomorphism classes of flowchart schemes). *Revue Roumaine de Mathematiques Pures et Applique*, 35:73–79, 1990.

[GLS00] R. Grosu, D. Lucanu, Gh. Ştefănescu. Mixed relations as enriched semiringal categories. *J. of Universal Computer Science*, 6(1) 2000, pg. 112-129.

[Ste96b] Gh. Ştefănescu. A short tour on FEST. *Preprint Series in Mathematics*, Institute of Mathematics, Romanian Academy, No. 38/December 1996.

[Ste00] Gh. Ştefănescu. Network Algebra. *Springer-Verlag* 2000.

A Anexă: implementarea relaţiilor sortate

În continuare prezentăm un fragment de cod scris în Java care reprezintă clasa ce implementează *relaţiile sortate* prezentate în secţiunea 1.1.1.

Clasa are ca atribute numele relaţiei, interfeţele relaţiei şi perechile de intrări şi ieşiri care sunt în relaţie. De asemenea, avem implementări ale operaţiilor cu relaţii sortate: compunere, juxtapunere şi feedback.

```
class Rel {
    // an implementation of sorted relations, using a list of
    // pairs showing which inputs and outputs are connected

    String name;
    String sortIn;  //the sorts of the inputs
    int in;         //the number of inputs of the relation
                    //(the cardinal of ‘domain‘)
    String sortOut; //the sorts of the outputs
    int out;        //the number of outputs of the relation
                    //(the cardinal of ‘codomain‘)
    PairItem pairsList; //the relation‘s pairs

    public Rel(String name, String sortIn, String sortOut) {
        //constructor

        this.name = name;
        this.in = sortIn.length();
        this.sortIn = sortIn;
        this.out = sortOut.length();
        this.sortOut = sortOut;
        this.pairsList = null;
    }

    Rel comp(Rel r) {
        //implementation of composition for relations

        if (out != r.in)
```

```
            return null;
        if (!sortOut.equals(r.sortIn)) return null;
        PairItem tmp1 = pairsList;
        PairItem tmp2 = r.pairsList;
        Rel result = new Rel("",sortIn,r.sortOut);
        PairItem temp = new PairItem(0,0);
        while (tmp1 != null) {
            tmp2 = r.pairsList;
            while (tmp2 != null && tmp2.i <= tmp1.j) {
                if ((tmp2.i == tmp1.j) && (sortOut.charAt(tmp1.j-1)
                                     == r.sortIn.charAt(tmp2.i-1)))
                    result.pairsList = temp.add(result.pairsList,
                                      tmp1.i,tmp2.j);
                tmp2 = tmp2.next;
            }
            tmp1 = tmp1.next;
        }
        return result;
    }

    Rel juxt(Rel r) {
        //implementation of juxtaposition for relations

        Rel result;
        String sI, sO;
        sI = sortIn+r.sortIn;
        sO = sortOut+r.sortOut;
        result = new Rel("",sI,sO);
        PairItem temp = pairsList;
        while (temp != null) {
           result.pairsList = temp.add(result.pairsList,
                                       temp.i, temp.j);
           temp = temp.next;
        }
        temp = r.pairsList;
        while (temp != null) {
            result.pairsList = temp.add(result.pairsList,
```

```
                                in+temp.i, out+temp.j);
            temp = temp.next;
        }
        return result;
    }

    Rel feedback(int c) {
        // implementation of feedback for relation

        if (pairsList == null)
            return null;
        // we suppose that c is lower than in and out
        // (tested elsewhere)
        if (sortIn.substring(in-c).
              compareTo(sortOut.substring(out-c)) != 0){
            return null;
        }
        Rel f = new Rel("",sortIn,sortOut);
        Rel result;
        String sI, sO;
        sI = sortIn.substring(0,in-c);
        sO = sortOut.substring(0,out-c);
        result = new Rel("",sI,sO);
        PairItem temp = pairsList;
        while (temp != null) {
            f.pairsList = temp.add(f.pairsList,
                                    temp.i,temp.j);
            temp = temp.next;
        } //we made ‘f‘ to be a copy of ‘this‘

        int i, j, k;
        for (k=0; k<c; k++)
            for (i=0; i<in-k-1; i++)
                for (j=0; j<out-k-1; j++)
                    if (f.pairsList.contains(1+i,out-k)
                        && f.pairsList.contains(in-k,1+j)
                        && sI.charAt(i) == sO.charAt(j))
```

```
                        f.pairsList = f.pairsList.
                              add(f.pairsList, 1+i, 1+j);
        for (i=0; i<in-c; i++)
            for (j=0; j<out-c; j++)
                if (f.pairsList.contains(1+i,1+j))
                    result.pairsList = f.pairsList.
                         add(result.pairsList, 1+i, 1+j);
        return result;
    }

} //end of class Rel
```

B Anexă: implementarea formei normale a expresiilor fluxnomiale

În această anexă prezentăm fragmentul de cod din proiect care reprezintă clasa ce implementează *forma normală a unei expresii fluxnomiale* prezentată în secţiunea 1.2.

Clasa are ca atribute interfeţele formei normale, iar apoi şirul variabilelor şi relaţia asociată formei normale $nf = (x, f)$. De asemenea, avem implementări ale operaţiilor cu forme normale: forma normală a unei relaţii, a unei variabile, a unei compuneri de expresii în formă normală, a unei juxtapuneri de forme normale şi a feedback-ului unei forme normale.

```
class NormalForm {
// we implement a nf-pair (x,f)
// where x is an array of x[i]‘s
// and f is a sorted relation

    String in;           // input interface
    String out;          // output interface
    static int MAX = 70;
    Variable x[];        // the x[i]‘s array
    int nX;              // the number of x[i] in x
    Rel f;               // the sorted relation

    NormalForm (String in, String out) {
        // a default constructor
        this.in = in;
        this.out = out;
        x = new Variable[MAX];
        nX = 0;
        f = null;
    }

    NormalForm(Variable v) {
        // the nf of a variable
```

```
        in = v.in;
        out = v.out;
        x = new Variable[MAX];
        Transposition tr;
        x[0] = v;
        nX = 1;
        tr = new Transposition(v.in,v.out);
        f = tr.asRel();
    }

    NormalForm(Rel r) {
        // the nf of a relation
        in = r.sortIn;
        out = r.sortOut;
        x = new Variable[MAX];
        nX = 0;
        f = r;
    }

    NormalForm juxt(NormalForm nf){
        // the nf of (nf1 juxt nf2)
        NormalForm nForm = new NormalForm(in+nf.in,out+nf.out);
        int i, j;
        String inX, outX, inXX, outXX;
        // inX is the input for the all x[i]
        // inXX is the input for the all nf.x[i]
        Identity id1, id2, id3, id4;
        Transposition tr1, tr2;

        inX = ""; outX = ""; inXX = ""; outXX = "";
        for (i=0; i<nX; i++)  {
            nForm.x[i]= x[i];
            inX += x[i].in;
            outX += x[i].out;
        }
        for (j=0; j<nf.nX; j++) {
            nForm.x[nX+j] = nf.x[j];
```

```
            inXX += nf.x[j].in;
            outXX += nf.x[j].out;
        }
        // f for the new normal form
        id1 = new Identity(in);
        id2 = new Identity(outXX);
        id3 = new Identity(out);
        id4 = new Identity(inXX);
        tr1 = new Transposition(nf.in,outX);
        tr2 = new Transposition(inX,nf.out);
        nForm.f = id1.asRel().juxt(tr1.asRel()).
                  juxt(id2.asRel());
        nForm.f = nForm.f.comp(f.juxt(nf.f));
        nForm.f = nForm.f.comp(id3.asRel().
                  juxt(tr2.asRel()).juxt(id4.asRel()));
        nForm.nX = nX+nf.nX;
        return nForm;
    }

    NormalForm comp(NormalForm nf) {
        // the nf of (nf1 comp nf2)
        NormalForm nForm = new NormalForm(in,nf.out);
        int i, j;
        String inX, outX, inXX, outXX;
        //inX is the input for the all x[i]
        //inXX is the input for the all nf.x[i]
        Identity id1, id2, id3, id4;
        Transposition tr1, tr2;

        inX = ""; outX = ""; inXX = ""; outXX = "";
        for (i=0; i<nX; i++) {
            nForm.x[i]= x[i];
            inX += x[i].in;
            outX += x[i].out;
        }
        for (j=0; j<nf.nX; j++) {
            nForm.x[nX+j] = nf.x[j];
```

```
            inXX += nf.x[j].in;
            outXX += nf.x[j].out;
        }
        // f for the new normal form
        id1 = new Identity(outXX);
        id2 = new Identity(out);
        id3 = new Identity(inX);
        id4 = new Identity(nf.out);
        tr1 = new Transposition(inX,outXX);
        tr2 = new Transposition(inXX,inX);
        nForm.f = f.juxt(id1.asRel());
        nForm.f = nForm.f.comp(id2.asRel().juxt(tr1.asRel()));
        nForm.f = nForm.f.comp((nf.f).juxt(id3.asRel()));
        nForm.f = nForm.f.comp(id4.asRel().juxt(tr2.asRel()));
        nForm.nX = nX+nf.nX;
        return nForm;
    }

    NormalForm feedback(String wires) {
        // the nf of (nf1 feed(wires))
        NormalForm nForm;
        Rel tmp;
        int i;
        String inX, outX; //inX is the input for the all x[i]
        String iF, oF;
        Identity id1, id2;
        Transposition tr1, tr2;

        i = in.length() - wires.length();
        iF = in.substring(0,i);
        i = out.length() - wires.length();
        oF = out.substring(0,i);
        nForm = new NormalForm(iF,oF);
        inX = ""; outX = "";
        for (i=0; i<nX; i++) {
            nForm.x[i]= x[i];
            inX += x[i].in;
```

```
            outX += x[i].out;
        }
        // f of the new normal form
        id1 = new Identity(iF);
        id2 = new Identity(oF);
        tr1 = new Transposition(outX,wires);
        tr2 = new Transposition(wires,inX);
        tmp = id1.asRel().juxt(tr1.asRel());
        tmp = tmp.comp(f);
        tmp = tmp.comp(id2.asRel().juxt(tr2.asRel()));
        nForm.f = tmp.feedback(wires.length());
        nForm.nX = nX;
        return nForm;
    }
}
```

C Anexă: Axiomele algebrei reţelelor

Tabelul 2: Axiomele standard ale calculului fluxnomial

I.	**Axiome pentru symocat-uri**
B1	$f \star (g \star h) = (f \star g) \star h$
B2	$\mathsf{I}_\varepsilon \star f = f = f \star \mathsf{I}_\varepsilon$
B3	$f \cdot (g \cdot h) = (f \cdot g) \cdot h$
B4	$\mathsf{I}_a \cdot f = f = f \cdot \mathsf{I}_b$
B5	$(f \star f') \cdot (g \star g') = f \cdot g \star f' \cdot g'$
B6	$\mathsf{I}_a \star \mathsf{I}_b = \mathsf{I}_{a\star b}$
B7	${}^a\mathsf{X}^b \cdot {}^b\mathsf{X}^a = \mathsf{I}_{a\star b}$
B8	${}^a\mathsf{X}^{b\star c} = ({}^a\mathsf{X}^b \star \mathsf{I}_c) \cdot (\mathsf{I}_b \star {}^a\mathsf{X}^c)$
B9	$(f \star g) \cdot {}^c\mathsf{X}^d = {}^a\mathsf{X}^b \cdot (g \star f)$
	pentru $f : a \to c,\ g : b \to d$

II.	**Axiome pentru feedback**
R1	$f \cdot (g \uparrow^c) \cdot h = ((f \star \mathsf{I}_c) \cdot g \cdot (h \star \mathsf{I}_c)) \uparrow^c$
R2	$f \star g \uparrow^c = (f \star g) \uparrow^c$
R3	$(f \cdot (\mathsf{I}_b \star g)) \uparrow^c = ((\mathsf{I}_a \star g) \cdot f) \uparrow^d$
	pentru $f : a \star c \to b \star d,\ g : d \to c$
R4	$(f \uparrow^b) \uparrow^a = f \uparrow^{a\star b}$
R5	$\mathsf{I}_a \uparrow^a = \mathsf{I}_\varepsilon$
R6	${}^a\mathsf{X}^a \uparrow^a = \mathsf{I}_a$

III.	**Axiome pentru constante de ramif. (angelice), fără feedback**
A1a	$\wedge^a \cdot (\wedge^a \star \mathsf{I}_a) = \wedge^a \cdot (\mathsf{I}_a \star \wedge^a)$
A1b	$\wedge^a \cdot (\perp^a \star \mathsf{I}_a) = \mathsf{I}_a$
A1c	$(\vee_a \star \mathsf{I}_a) \cdot \vee_a = (\mathsf{I}_a \star \vee_a) \cdot \vee_a$
A1d	$(\top_a \star \mathsf{I}_a) \cdot \vee_a = \mathsf{I}_a$
A2a	$\wedge^a \cdot {}^a\mathsf{X}^a = \wedge^a$
A2b	${}^a\mathsf{X}^a \cdot \vee_a = \vee_a$
A3a	$\top_a \cdot \perp^a = \mathsf{I}_\varepsilon$
A3b	$top_a \cdot \wedge^a = \top_a \star \top_a$
A3c	$\vee_a \cdot \perp^a = \perp^a \star \perp^a$
A3d	$\vee_a \cdot \wedge^a =$
	$(\wedge^a \star \wedge^a) \cdot (\mathsf{I}_a \star {}^a\mathsf{X}^a \star \mathsf{I}_a) \cdot (\vee_a \star \vee_a)$
A4	$\wedge^a \cdot \vee_a = \mathsf{I}_a$
A5a	$\perp^{a\star b} = \perp^a \star \perp^b$
A5b	$\wedge^{a\star b} = (\wedge^a \star \wedge^b) \cdot (\mathsf{I}_a \star {}^a\mathsf{X}^b \star \mathsf{I}_b)$
A5c	$\top_{a\star b} = \top_a \star \top_b$
A5d	$\vee_{a\star b} = (\mathsf{I}_a \star {}^b\mathsf{X}^a \star \mathsf{I}_b) \cdot (\vee_a \star \vee_b)$
A5e	$\perp^\varepsilon = \mathsf{I}_\varepsilon$
A5f	$\wedge^\varepsilon = \mathsf{I}_\varepsilon$
A5g	$\top_\varepsilon = \mathsf{I}_\varepsilon$
A5h	$\vee_\varepsilon = \mathsf{I}_\varepsilon$

IV.	**Feedback pe constante de ramificare (angelice)**
R7	$\wedge^a \uparrow^a = \top_a$
R8	$\vee_a \uparrow^a = \perp^a$
R9	$[(\mathsf{I}_a \star \wedge^a) \cdot ({}^a\mathsf{X}^a \star \mathsf{I}_a) \cdot$
	$\cdot (\mathsf{I}_a \star \vee_a)] \uparrow^a = \mathsf{I}_a$

V.	**Axiome 'tari'**
Sa	$f \cdot \perp^b = \perp^a$
Sb	$f \cdot \wedge^b = \wedge^a \cdot (f \star f)$
Sc	$\top_a \cdot f = \top_b$
Sd	$\vee_a \cdot f = (f \star f) \cdot \vee_b$

VI. **Regula enzimatică** (E este o clasă de relaţii finite, abstracte)

ENZ$_E$: dacă pentru $f : a \star c \to b \star c$ şi $g : a \star d \to b \star d$ există $y : c \to d$ în E a.î. $f \cdot (\mathsf{I}_b \star y) = (\mathsf{I}_a \star y) \cdot g$, atunci $f \uparrow^c = g \uparrow^d$