Relații, funcții
şi calcul în sens Turing

Capitolele de mai jos introduc noțiuni de bază privind construcția funcțiilor recursive şi unele noțiuni de calculul în sens Turing, ca variantă a calculabilității efective.

1 Relații

O relație $\mathbf{R}$ peste mulțimile M_i, $i=1,n$, este o submulțime a produsului cartezian $M_1 \times M_2 \times ... \times M_n$. Un element al relației $\mathbf{R}$ este un tuplu $(x_1,x_2,...,x_n)$ cu $x_i \in M_i$, $i=1,n$. Dacă $n=2$ şi $M_1=M_2$ spunem că $\mathbf{R}$ este o relație binară peste $\mathbf{M}$ şi pentru $(x,y)\in\mathbf{R}$ putem scrie $x\ \mathbf{R}\ y$ (de exemplu, scriem $2\leq 5$ ca alternativă pentru $(2,5)\in\leq$).

Domeniul $dom(R)$, respectiv codomeniul $ran(R)$, unei relații binare $\mathbf{R}$ peste M_1 şi M_2 corespund mulțimilor:

$$dom(R) =_{def} \{\forall x\in M_1, y\in M_2 \mid (x,y)\in R \bullet x\}$$
$$ran(R) =_{def} \{\forall x\in M_1, y\in M_2 \mid (x,y)\in R \bullet y\}$$

O relație binară $\mathbf{R}$ peste M_1 şi M_2 este totală dacă:

$$\forall x\in M_1, y\in M_2 \bullet (x,y)\in R \vee (y,x)\in R$$

Relația $\leq$ este totală peste $\mathbb{N}$, în timp ce relația $<$ nu este totală (prechile de forma (x,x) nu sunt în relația $<$).

Fie $\mathbf{P}$ o proprietate a elementelor lui $\mathbf{M}$, proprietate care poate fi privită ca o submulțime $P\subseteq M$ a elementelor din $\mathbf{M}$ ce satisfac $\mathbf{P}$, iar $\mathbf{R}$ o relație binară peste $\mathbf{M}$. Spunem că propriatatea $\mathbf{P}$ este păstrată de relația $\mathbf{R}$ dacă:

$$\forall x\in M, y\in M \mid (x,y)\in R \bullet P(x) \Rightarrow P(y)$$

O relație binară $\mathbf{R}$ peste $\mathbf{M}$ este:

- reflexivă, dacă $\forall x\in M \bullet (x,x)\in R$
- tranzitivă, dacă $\forall x\in M, y\in M, z\in M \mid (x,y)\in R \wedge (y,z)\in R \bullet (x,z)\in R$
- simetrică, dacă $\forall x\in M, y\in M \mid (x,y)\in R \bullet (y,x)\in R$
- antisimetrică, dacă $\forall x\in M, y\in M \mid (x,y)\in R \wedge (y,x)\in R \bullet x=y$

Fie R o relație binară peste o mulțime M. Spunem că relația R este de:

- *preordine*, dacă este reflexivă şi tranzitivă;
- *ordine parțială*, dacă este de preordine şi este antisimetrică;
- *ordine totală*, dacă este de ordine parțială şi este totală.
- *echivalență*, dacă este simetrică, reflexivă şi tranzitivă.

Dacă R este o relație de echivalență peste o mulțime M, iar x este un element din M, definim clasa de exhivalență a lui x ca o mulțime:

$$R[a] = \{y \in M \mid (x,y) \in R\}$$

De exemplu, $\leq$ este o relație de ordine totală peste N, iar relația

$$R = \{\forall x \in \mathrm{N}, y \in \mathrm{N} \mid x+y \text{ este par} \bullet (x,y)\}$$

este o relație de echivalență peste N. Pentru orice $x \in \mathrm{N}$ impar avem clasa de echivalență $R[x] = \{y \in \mathrm{N} \mid y \text{ este impar}\}$, iar pentru orice $x \in \mathrm{N}$ par avem $R[x] = \{y \in \mathrm{N} \mid y \text{ este par}\}$.

Fie R o relație binară peste o mulțime M. Închiderea refelexivă a relației R este cea mai mică[1] relație reflexivă R' peste M astfel încât $R \subseteq R'$. Constructivist,

$$R' = \{\forall x \in M \bullet (x,x)\} \cup R$$

Fie R o relație binară peste o mulțime M. Închiderea tranzitivă a relației R, notată R^+, este cea mai mică relație tranzitivă R' peste M astfel încât $R \subseteq R'$. Constructivist,

$$R^+ = \bigcup_i R_i$$

unde, $R_0 = R$

$$R_{i+1} = R_i \cup \{\forall x \in M, y \in M, z \in M \mid (x,y) \in R_i \wedge (y,z) \in R_i \bullet (x,z)\},\ i \geq 1$$

Fie R o relație binară peste o mulțime M. Închiderea reflexiv-tranzitivă a relației R, notată R^*, este cea mai mică relație reflexivă şi tranzitivă R' peste M astfel încât $R \subseteq R'$. Constructivist,

$$R^* = \{\forall x \in M \bullet (x,x)\} \cup R^+.$$

2 Funcții

O funcție f din A în B, notată $f:A \to B$, este o relație binară $f \subseteq A \times B$ care satisface restricția:

$$\forall x \in A, y \in B, z \in B \bullet (x,y) \in f \wedge (x,z) \in f \Rightarrow y=z$$

---

[1] R' este cea mai mică în sensul că pentru orice relație reflexivă R'' există implicația $R \subseteq R'' \Rightarrow R' \subseteq R''$.

Pentru orice $x \in A$ există cel mult o pereche $(x,y) \in f$. Dacă $(x,y) \in f$ spunem că funcția este definită în punctul x şi scriem $y=f(x)$ înțelegând că f are valoarea y în punctul x. Prin convenție, dacă f nu este definită în x scriem $f(x)=\perp$, iar dacă f este definită în x scriem $f(x)\neq\perp$ sau $f(x)=y$, presupunând că simbolul $\perp$ nu este în A şi nici în B.

Domeniul funcției $f:A\rightarrow B$, notat $dom(f)$, este submulțimea punctelor din A în care f este definită. Codomeniul lui f, notat $ran(f)$, este mulțimea valorilor funcției f. La fel ca şi pentru relații,

$$dom(f)=_{def} \{x \in A \mid (\exists y \in B \bullet (x,y) \in f)\}$$
$$ran(f)=_{def} \{y \in B \mid (\exists x \in A \bullet (x,y) \in f)\}$$

Funcția $f:A\rightarrow B$ este *totală* (peste A) dacă $\forall x \in A \bullet (\exists y \in B \bullet (x,y) \in f)$, deci dacă $dom(f)=A$. Altfel, dacă $dom(f)\subset A$, spunem că f este parțială (peste A).

Funcția $f:A\rightarrow B$ este *injectivă* dacă pentru puncte distincte din A are valori distincte, adică

$$\forall x \in dom(f), y \in dom(f) \bullet f(x)=f(y) \Rightarrow x=y$$

Funcția $f:A\rightarrow B$ este *surjectivă* dacă $\forall y \in B \bullet (\exists x \in A \bullet y=f(x))$, deci dacă $ran(f)=B$.

O funcție $f:A\rightarrow B$ totală, injectivă şi surjectivă este *bijectivă* (o bijecție $A\rightarrow B$). În acest caz, spunem că mulțimile A şi B sunt echipotente.

Două funcții $f:A\rightarrow B$ şi $g:A\rightarrow B$ sunt egale dacă şi numai dacă au acelaşi domeniu şi aceeaşi valoare pentru fiecare punct din domeniu. Scriem $f=g$ pentru a specifica egalitatea funcțiilor.

$$f=g \Leftrightarrow \forall x \in A \bullet (f(x)=\perp \wedge g(x)=\perp) \vee (\exists y \in B \bullet f(x)=y \wedge g(x)=y)$$

Prin urmare $f(x)=g(x)$ arată fie că :

- f şi g sunt definite în x şi au aceeaşi valoare, fie că
- $f(x)=\perp$ şi $g(x)=\perp$

2.1 Funcții primitiv-recursive

O funcție $f:N^n \rightarrow N$ este obținută prin transformare directă dintr-o funcție $g:N^m\rightarrow N$ dacă

$$f(x_1,x_2,...,x_n) = g(v_1,v_2,...,v_m)$$

unde x_i, $i=1,n$, sunt variabile cu valori în N, iar v_j este fie o variabilă x_i, $1\leq i\leq n$, fie o constantă din N. Prin convenție, scriem $f=Tr[g]$ pentru a indica posibilitatea construcției lui **f** printr-o transformare explicită a lui **g**, dar fără a preciza detaliile transformării.

O funcție $f:N^n \rightarrow N$ este obținută prin compunere pe baza funcțiilor $h:N^m \rightarrow N$, $g_i:N^n \rightarrow N$, $i=1,m$, dacă pentru orice vector de valori $\bar{x}^n = (x_1,x_2,...,x_n) \in N^n$ avem:

$f(\bar{x}^n) = h(g_1(\bar{x}^n), g_2(\bar{x}^n), ..., g_m(\bar{x}^n))$, dacă $\forall i \in 1..m \bullet g_i(\bar{x}^n) \neq \perp$

$f(\bar{x}^n) = \perp$, dacă $\exists i \in 1..m \bullet g_i(\bar{x}^n) = \perp$

Prin convenție, scriem $f=Cp[h,g_1,g_2,...,g_m]$ pentru a indica construcția lui **f** prin compunere.

O funcție $f:N^{n+1} \rightarrow N$ este obținută prin recursivitate primitivă, pe baza funcțiilor $h:N^{n+2} \rightarrow N$, $g:N^n \rightarrow N$, dacă:

$f(0, \bar{x}^n) = g(\bar{x}^n)$

$$f(m+1, \bar{x}^n) = \begin{cases} h(m, \bar{x}^n, f(m, \bar{x}^n)), & \text{dacă } f(m, \bar{x}^n) \neq \perp \\ \perp & \text{altfel} \end{cases}$$

Prin convenție, scriem **f=Pr[g,h]** pentru a indica construcția lui **f** prin recursivitate primitivă pe baza funcțiilor **g** şi **h**.

O funcție este *primitiv-recursivă* dacă poate fi obținută printr-un număr finit de paşi de aplicare a operațiilor de transformare directă, compunere şi recursivitate primitivă pornind de la funcțiile de bază: $zero_n$, $n \geq 0$ ($zero_n$ este funcția **n**-ară $\lambda x_1,x_2,...,x_n.0$; în particular, funcția nulară $zero_0$ este constanta **0**) şi **succ** (funcția succesor $\lambda x.x+1$). Alternativ, clasa funcțiilor primitiv-recursive este cea mai mică clasă de funcții care conține funcțiile de bază $zero_n$ şi **succ** şi este închisă sub operațiile de transformare directă, compunere şi recursivitate primitivă. Prin urmare, o funcție primitiv-recursivă poate fi scrisă **f=E**, unde **E** conține doar transformări **Tr**, **Cp** şi **Pr** aplicate funcțiilor $zero_n$ şi **succ**. Se poate demonstra [Tay 98] că orice funcție primitiv-recursivă este totală[2].

În exemplele de mai jos, convenim să scriem $\lambda x_1, x_2,...,x_n.calcul$ pentru a desemna o funcție **n**-ară cu parametrii x_i, $i=1,n$, şi cu un rezultat obținut conform calculului **calcul** (parametrizat în raport cu x_i, $i=1,n$). De asemenea, aplicația **f(x)** va fi scrisă uneori şi sub forma **(f x)**.

$id:N \rightarrow N$ este funcția identitate

```
id(0) = 0
id(n+1) = succ(id n) = (λn,r.(succ r) (n,id(n)))
id = Pr[0, λn,r.(succ r)] = Pr[zero0,Tr[succ]]
```

2 Funcțiile de bază sunt totale, iar transformarea directă, compunerea funcțională şi recursivitatea primitivă generează o funcție totală prin aplicare asupra unor funcții totale.

pred:$\mathbb{N} \to \mathbb{N}$ **pred(n)** calculează predecesorul lui **n**, cu limitare la **0**.

pred(0) = 0
pred(n+1) = n = id(n) = λn,r.(id n) (n,pred(n))
pred = Pr[0, λn,r.(id n)] = Pr[$zero_0$,Tr[id]]

zerop:$\mathbb{N} \to \mathbb{N}$ **zerop(n)** este **1** dacă **n=0** şi **0** altfel

zerop(0) = 1 = succ(0)
zerop(n+1) = 0 = λn,r.0
zerop = Pr[succ(0), λn,r.0] = Pr[Cp[succ,$zero_0$],$zero_2$],

unde compunerea **Cp[succ,$zero_0$]** este o rescriere la limită a aplicației **succ(0)**

plus:$\mathbb{N}^2 \to \mathbb{N}$ **plus(n,m)** calculează suma **n+m**

plus(0,m) = m = id(m)
plus(n+1,m) = succ(plus(n,m)) = λn,m,r.(succ r)(n,m,plus(n,m))
plus = Pr[id, λn,m,r.(succ r)]= Pr[id,Tr[succ]]

dif:$\mathbb{N}^2 \to \mathbb{N}$ **dif(n,m)** calculează diferența **n-m**, cu limitare la **0** (**dif(n,m)=0** dacă **n**<**m**).

dif(n,m) = dif'(m,n)= Tr[dif'], unde **dif'(n,m)** calculează diferența **m-n**

dif'(0,m) = m = id(m)
dif'(n+1,m) = pred(dif'(n,m)) = λn,m,r.(pred r)(n,m,dif'(n,m))
dif' = Pr[id, λn,m,r.(pred r)]= Pr[id,Tr[pred]]

ori:$\mathbb{N}^2 \to \mathbb{N}$ **ori(n,m)** calculează produsul **n * m**
ori(0,m) = 0 = $zero_1$(m)
ori(n+1,m) = plus(m,ori(n,m)) = λn,m,r.(plus(m,r))(n,m,plus(n,m))
ori = Pr[$zero_1$, λn,m,r.(plus(m,r))] = Pr[$zero_1$,Tr[plus]]

mic_egal:$\mathbb{N}^2 \to \mathbb{N}$ **mic_egal(n,m)** este **1** dacă **n**≤**m** şi **0** altfel.

mic_egal(n,m)= zerop(dif(m,n)) = Cp[zerop,dif]

2.2 Funcții parțial-recursive

O funcție **f**:$\mathbb{N}^n \to \mathbb{N}$ este obținută prin minimizare (sau μ-transformare) pe baza unei funcții **g**:$\mathbb{N}^{n+1} \to \mathbb{N}$, dacă:

$f(\bar{x}^n) = \mu_m\, g(m, \bar{x}^n)$, dacă $\exists m \in \mathbb{N} \bullet g(m, \bar{x}^n)=0$

$f(\bar{x}^n) = \perp$ altfel

unde $\mu_m\, g(m, \bar{x}^n)$ este cea mai mică valoare a lui **m** pentru care $g(m, \bar{x}^n)=0$.

O funcție este *parțial-recursivă* dacă poate fi obținută printr-un număr finit de paşi de aplicare a operațiilor de transformare directă, compunere, recursivitate primitivă şi minimizare pornind de la funcțiile de bază **zero**$_n$, **n**$\geq$**0**, şi **succ** . Alternativ, clasa funcțiilor parțial-recursive este cea mai mică clasă care conține funcțiile de bază **zero**$_n$ şi **succ** şi este închisă sub operațiile de transformare directă, compunere, recursivitate primitivă şi minimizare. Mulțimea funcțiilor parțial recursive include mulțimea funcțiilor primitiv-recursive şi, totodată conține şi funcții care nu sunt totale[3]. Funcțiile parțial-recursive care sunt totale se mai numesc şi funcții recursive, deşi acest termen este folosit uneori pentru a desemna o funcție recursivă fără a preciza clasa ei.

De exemplu, funcția **div(n,m)** de mai jos calculează câtul împărțirii întregi dintre numerele naturale **n** şi **m** şi este parțial recursivă. Predicatul **mic(n,m)** reîntoarce **1** dacă **n**<**m** şi **0** altfel.

div(n,m) = μ_t**(g(t,n,m)**

g(t,n,m) = mic(plus(ori(t,m),m),succ(n))

adică **g(t,n,m) = t*m+m** < **succ(n)**. Se observă că **g** este o compunere obținută pe baza unor funcții primitiv-recursive. Deci **div** poate fi obținută exclusiv prin operațiile de transformare directă, compunere, recursivitate primitivă şi minimizare pornind de la funcțiile de bază **zero** şi **succ**.

Avem **div(15,7)=2**. Într-adevăr, **g(t,15,7)=1** pentru **t=0,1** şi **g(2,15,7)=0**. Deci μ_t**(g(t,15,7))=2** şi, implicit, **div(15,7)=2**.

Se observă că funcția **div** nu este totală, nefiind definită pentru tuplurile **(n,0)**, **n**$\in$N. Într-adevăr, funcția **g(t,n,0)** nu se anulează pentru nici o valoare a lui **t** şi a lui **n**. Deci, conform definiției operației de μ-transformare, μ_t**(g(t,n,0)) =** $\perp$ şi, implicit, **div(n,0)=**$\perp$.

---

[3] Operația de minimizare aplicată asupra unei funcții totale poate genera o funcție parțială.

3 Calcul în sens Turing

Maşina Turing, în particular maşina Turing deterministă, reprezintă modelul tradițional al calculabilității, diversele modele fiind raportate în cele din urmă la Turing-calculabilitate. Calculabilitatea efectivă este, de asemenea, raportată la Turing-calculabilitate, iar multe domenii ale ştiinței calculatoarelor folosesc aparatul matematic şi proprietățile maşinii Turing ca bază teoretică.

3.1 O descriere informală a maşinii Turing

Deşi a premers explozia tehnologiei digitale, maşina Turing este apropiată ca structură şi mod de funcționare de un calculator digital modern. Diferența principală este că o maşină Turing este construită în mod special pentru un anumit calcul[4]. Similar cu un calculator numeric, o maşină Turing particulară **M** este formată dintr-o unitate de control şi o memorie.

Memoria este o bandă deplasabilă în fața unui cap **H** de citire/scriere. Banda maşinii este divizată în celule, iar lungimea ei (numărul de celule) nu este limitat. Fiecare celulă memorează un simbol dintr-o mulțime finită Γ, numită alfabetul benzii. Fiecare celulă poate fi citiă şi rescrisă, rescrierea distrugând conținutul curent al celulei. În Γ există un simbol special, notat **B** (şi numit "blank"), folosit pentru a marca celulele libere de pe bandă. Mişcarea benzii este în cuante, o cuantă de mişcare corespunzând deplasării benzii cu o celulă, spre stânga sau spre dreapta, în fața capului **H**. Simbolul aflat în celula poziționată în fața capului **H** este denumit *simbolul curent scanat*.

Unitatea de control este un automat finit determinist care controlează operația executată de capul **H** (citire/scriere) şi mişcarea benzii. Operațiile (sau instrucțiunile) ce pot fi executate de maşină sunt:

- $\sigma_1{:}\sigma_2$ Dacă simbolul curent scanat este $\sigma_1 \in \Gamma$ el este rescris cu simbolul σ_2. În particular, σ_1 şi/sau σ_2 pot fi **B**.
- $\sigma{:}\mathbf{R}$ Dacă simbolul curent scanat este $\sigma \in \Gamma$ atunci banda se deplasează spre dreapta cu o celulă în fața capului **H**.
- $\sigma{:}\mathbf{L}$ Dacă simbolul curent scanat este $\sigma \in \Gamma$ atunci banda se deplasează spre dreapta cu o celulă în fața capului **H**.

Se observă că execția instrucțiunilor este condiționată de simbolul curent scanat. De asemenea, cel mult o singură instrucțiune este executabilă la un moment dat pentru orice stare a maşinii (determinism).

---

[4] Sunt prezentate, pentru simplitate, doar astfel de maşini. Generalizarea există, maşina Turing universală fiind programabilă.

Diagrama tranzițiilor efectuate de unitatea de control a maşinii corespunde unui graf orientat în care nodurile reprezintă stări, iar arcele tranziții între stări. Fiecare nod este etichetat cu un *simbol de stare*, notat convențional q_k, dintr-o mulțime finită de asemenea simboluri. Un arc este etichetat cu instrucțiunea executată de maşină în momentul parcurgerii arcului. Procesul de schimbare a stării maşinii prin execuția unei instrucțiuni constituie un pas al calculului efectuat de maşină.

Prin convenție, se presupune că maşina **M** este într-o stare specială q_0 la începutul calculului (stare inițială). Maşina se opreşte atunci când nici o instrucțiune nu mai poate fi executată pornind din starea curentă.

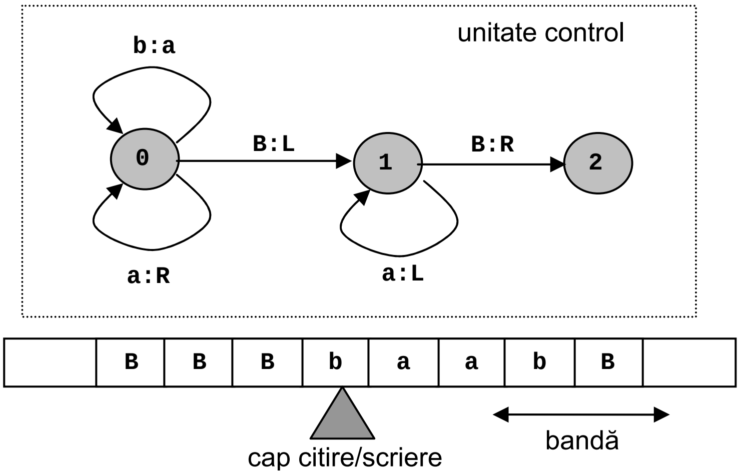

Figura 1 O maşină deterministă Turing

Conținutul benzii şi starea maşinii se schimbă în cursul funcționării. Folosim termenul *configurația benzii* pentru a desemna conținutul benzii (al memoriei maşinii) şi poziția capului **H** la un moment dat de timp. De asemenea, termenul *configurația maşinii* desemnează configurația benzii şi starea curentă a unității de control la un anumit moment în cursul funcționării.

Pentru a descrie calculul efectuat de maşină, un pas de calcul este reprezentat folosind notația:

$$...Bsq_i\sigma s'B... \;—\; \sigma{:}\alpha \rightarrow ...B\underline{s}q_j\sigma'\underline{s'}B...$$

Notația arată că plecând de la configurația curentă a benzii $...Bs\sigma s'B...$, unde $s,s'\in\Gamma^*$, iar $\sigma\in\Gamma$ este simbolul curent scanat, şi de la starea curentă q_i atunci prin execuția instrucțiunii $\sigma{:}\alpha$ - cu acțiunea $\alpha\in\Gamma\cup\{L,R\}$ - configurața benzii devine $...B\underline{s}\sigma'\underline{s'}B...$ - unde σ' este noul simbol scanat – şi starea maşinii devine q_j.

Scrierea **...B** desemnează un şir de simboluri aflate pe bandă la stânga capului **H** şi care este format doar din simboluri **B**. Scrierea **B...** desemnează un şir de simboluri aflate pe bandă la dreapta capului **H** şi care este format doar din simboluri **B**. În cele ce urmează considerăm implicite punctele ce preced sau succed simbolul **B** din specificarea unui pas de calcul.

Ca exemplu, se consideră maşina Turnig din figura 1 (preluată din [Tay 98]). În configuraţia iniţială a maşinii celulele benzii coţin simbolul **B** (sunt libere) cu excepţia unei zone compacte ce memorează simbolurile **a** şi **b**. Capul **H** este poziţionat pe primul simbol diferit de **B** de pe bandă, iar starea iniţială a maşinii este q_0.

Calculul efectuat de maşină constă în înlocuirea tuturor simbolurilor **a** de pe bandă cu simboluri **b**. De asemenea, maşina trebuie să se oprească cu capul **H** poziţionat ca în starea iniţială: pe primul simbol non **B** de pe bandă. Calculul este descris de următorii paşi:

$$
\begin{aligned}
\mathtt{Bq_0baabB} &\text{ — } \mathtt{b{:}a} \rightarrow \mathtt{Bq_0aaabB} \text{ — } \mathtt{(a{:}R)^3} \rightarrow \mathtt{Baaaq_0bB}\\
&\text{ — } \mathtt{b{:}a} \rightarrow \mathtt{Baaaq_0aB} \text{ — } \mathtt{a{:}R} \rightarrow \mathtt{Baaaaq_0B}\\
&\text{ — } \mathtt{B{:}L} \rightarrow \mathtt{Baaaq_1aB} \text{ — } \mathtt{(a{:}L)^4} \rightarrow \mathtt{q_1BaaaaB}\\
&\text{ — } \mathtt{B{:}R} \rightarrow \mathtt{Bq_2aaaaB}
\end{aligned}
$$

unde $(i)^n$ desemnează aplicarea de **n** ori a instrucţiunii **i**. Se observă că pentru orice stare a maşinii cel mult o singură instrucţiune este executabilă într-un pas. Maşina este deterministă.

3.2 Maşini Turing deterministe

O maşină Turing deterministă **M** este un qvintuplu $M = <Q,\Sigma,\Gamma,q_0,\delta>$ unde:

Q este o mulţime finită şi nevidă de stări. Există o stare $q_0 \in Q$ singulară, reprezentând starea initială a maşinii **M**.

Γ şi Σ sunt mulţimi finite nevide astfel încât $\Sigma \subseteq \Gamma$. Σ este alfabetul de intrare al lui **M** (simbolurile ce constituie configuraţia iniţială a benzii lui **M**). Γ este alfabetul complet al benzii lui **M** (simbolurile ce pot fi folosite în orice configuraţie a benzii în cursul calculului executat de **M**).

$\delta: Q\times\Gamma \rightarrow (\Gamma \cup \{L,R\})\times Q$ este o funcţie parţială peste $Q\times\Gamma$ numită funcţie de tranziţie a lui **M**. Funcţia δ primeşte o stare q_i şi un simbol $\sigma\in\Gamma$ al benzii şi reîntoarce o singură acţiune $\alpha\in\Gamma\cup\{L,R\}$ şi o singură stare q_j. Funcţia codifică paşii de calcul ce pot fi executaţi de **M**. Dacă **M** este în starea q_i şi simbolul curent scanat este σ atunci maşina execută instrucţiunea $\sigma{:}\alpha$ şi trece în starea q_j.

O maşină Turing poate fi folosită pentru a calcula funcţii teoretice numerice (number-theoretic functions). Spunem că o funcţie **f** este *Turing calculabilă* dacă există o maşină Turing care pentru fiecare configuraţie iniţială a benzii, reprezentând argumentul **x** al funcţiei, fie se opreşte cu o configuraţie a benzii ce corespunde rezultatului $f(x)$, dacă $x\in dom(f)$, fie nu se opreşte, dacă $x\notin dom(f)$.

Prin convenție, în exemplele următoare reprezentăm numărul natural **n** ca o secvență de **n+1** simboluri diferite de **B**, de exemplu $\underbrace{1\,..\,1}_{n+1}$ sau, prescurtat, 1^{n+1}. Reprezentarea dezambiguează între **0**, care ar corespunde unei celule libere, şi **B**.

Ca un prim exemplu de funcție Turing-calculabilă, să considerăm adunarea a două numere naturale: **plus(n,m)=n+m**, $n\in N$, $m\in N$. Numim **plus** maşina care calculează funcția şi a cărei diagramă de tranziții este ilustrată în figura 2.

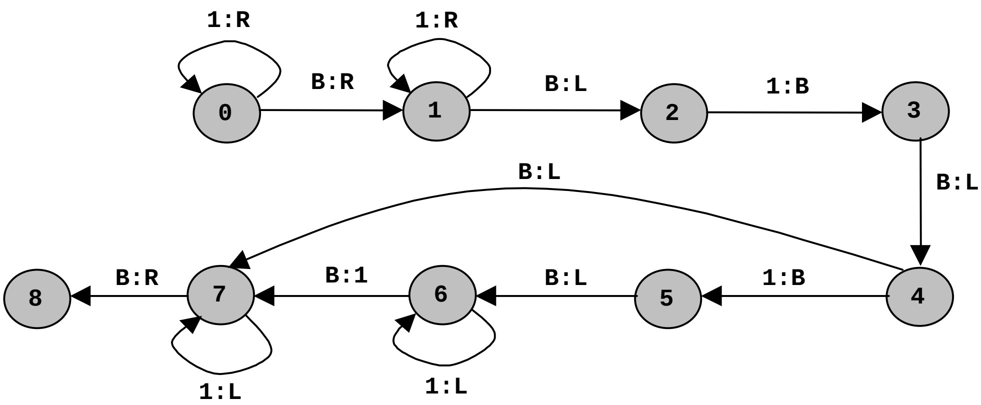

Figura 2. Adunarea a două numere naturale

Configurația inițială a benzii maşinii **plus** conține reprezentările numerelor **n** şi **m**, separate printr-un simbol **B**. Celelalte celule ale benzii sunt **B** (libere). Inițial, capul **H** este poziționat pe primul digit al lui **n** (pe primul simbol **1** din reprezentarea lui **n**). Configurația finală a benzii conține reprezentarea numărului cu valoarea **n+m**, capul **H** fiind poziționat pe primul digit al rezultatului.

$$...B\ \underset{\substack{\uparrow\\ H}}{1}{}^{n+1}\ B\ 1^{m+1}\ B...\ \text{— plus} \to\ ...B\ \underset{\substack{\uparrow\\ H}}{1}{}^{n+m+1}\ B...$$

Paşii calculului **plus(n,m)** au un efect simplu: ultimul **1** din **m** este preschimbat în **B**, iar simbolul **B** de separație dintre **n** şi **m** este transformat în **1** atunci când $m\geq 1$.

Inițial $Bq_0 1^{n+1}B1^{m+1}B$

— $(1{:}R)^{n+1}\to B1^{n+1}q_0B1^{m+1}B$

— $B{:}R\to B1^{n+1}Bq_1 1^{m+1}B$

— $(1{:}R)^{m+1}\to B1^{n+1}B1^{m+1}q_1B$

— $B{:}L\to B1^{n+1}B1^{m}q_2 1B$

— $1{:}B\to B1^{n+1}B1^{m}q_3BB$

Caz **m=0** $B1^{n+1}Bq_3BB$

— $B{:}L\to B1^{n+1}q_4BBB$

— $B{:}L\to B1^{n}q_7 1BBBB$

— $(B{:}L)^{n+1}\to q_7B1^{n+1}BBBB$

— $B{:}R\to Bq_8 1^{n+1}1BBBB$

Caz `m>0` $B1^{n+1}B1^{m}q_3BB$.
— $B{:}L\rightarrow B1^{n+1}B1^{m-1}q_4 1BB$
— $1{:}B\rightarrow B1^{n+1}B1^{m-1}q_5BBB$

Caz `m=1` $B1^{n+1}Bq_5BBB$
— $B{:}L\rightarrow B1^{n+1}q_6BBBB$
— $B{:}1\rightarrow B1^{n+1}q_7 1BBB$
— $(1{:}L)^{n+2}\rightarrow q_7B1^{n+m+1}BBB$
— $B{:}R\rightarrow Bq_8 1^{n+m+1}BBB$

Caz `m>1` $B1^{n+1}B1^{m-1}q_5BBB$
— $B{:}L\rightarrow B1^{n+1}B1^{m-2}q_6 1BBB$
— $(1{:}L)^{m-1}\rightarrow B1^{n+1}q_6B1^{m-1}BBB$
— $B{:}1\rightarrow B1^{n+1}q_7 1^{m}BBB$
— $(1{:}L)^{n+2}\rightarrow q_7B1^{n+m+1}BBB$
— $B{:}R\rightarrow Bq_8 1^{n+m+1}BBB$

Calculele complicate pot fi descompuse în fragmente mai simple, iar maşinile care calculează fragmentele pot fi combinate pentru a constitui o maşină care realizează întregul calcul. Bunăoară, cuplarea în secvenţă a două maşini Turing corespunde compunerii funcţionale. De exemplu, funcţia **`dublu(n)=2*n`** poate fi calculată de o maşină **`dublu = plus o copie`**, rezultată prin cuplarea în cascadă a maşinilor **`plus`** şi **`copie`**. Maşina **`copie`** construieşte o copie a numărului **`n`**, iar **`plus`** realizează adunarea **`plus(n,n)`**.

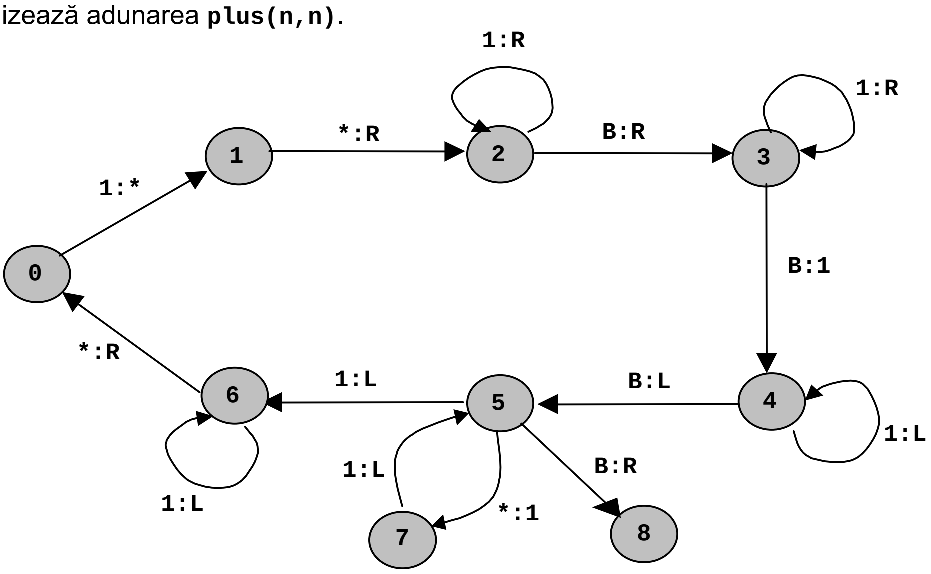

Figura 3. Copierea a unui număr natural

O posibilă implementare a maşinii **`copie`** este ilustrată în figura 3. Configuraţia iniţială a maşinii este $Bq_0 1^{n+1}B$. Maşina se opreşte cu configuraţia $Bq_8 1^{n+1}B1^{n+1}B$. Starea finală este întotdeauna q_8.

Dacă din q_8 a maşinii **copie** are loc o tranziţie în starea q_0 a maşinii **plus**, aşa cum se arată în figura 4, atunci maşina **dublu** rezultată prin compunerea **plus o copie** calculează **2*n**. Tranziţia este $q_{8(copie)}$**1:**$q_{0(plus)}$**1** şi este întotdeauna posibilă. Întradevăr, chiar dacă **n=0**, copia lui **n** are cel puţin un simbol **1**, iar capul **H** al maşinii **copie** se află poziţionat pe primul **1** al lui **n** în starea $q_{8(copie)}$. Starea iniţială a maşinii **dublu** este starea iniţială q_0 a maşinii **copie**, iar starea finală a lui **dublu** este starea finală q_8 a maşinii **plus**.

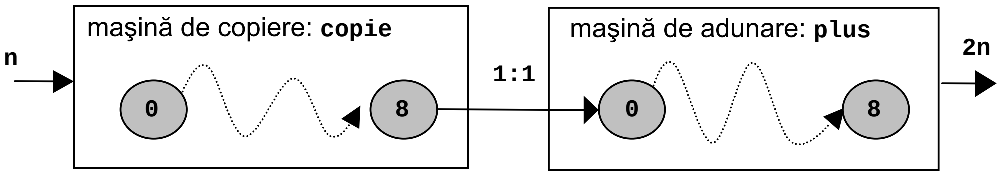

Figura 4. O maşină pentru calculul **2*n = plus(n,copie(n))**

Ca exemplu adiţional, să construim o maşină Turing, numită **count**, care testează egalitatea dintre numărul de apariţii ale două simboluri **a** şi **b** într-un şir **s**$\in$**{a,b}***. Dacă notăm $n_x(s)$ numărul de apariţii ale simbolului **x** în şirul **s**, atunci calculul realizat de maşina **count** este $n_a(s)=n_b(s)$. Maşina **count** are o configuraţie iniţială a benzii astfel încât capul **H** este poziţionat pe primul simbol al şirului, iar conţinutul benzii este **BsB**, unde **s** este şirul de scanat. Dacă $n_a(s)=n_b(s)$ configuraţia finală a benzii este **B1B**, iar dacă $n_a(s)\neq n_b(s)$ configuraţia finală a benzii este **B0B**, unde simbolurile **1** şi **0** sunt diferite faţă de **a** şi **b**. O posibilă implementare a maşinii **count** este ilustrată în figura 5. Maşina recunoaşte propoziţiile limbajului $\mathcal{L}=\{s\in\{a,b\}^*\mid n_a(s)=n_b(s)\}$: se termină întotdeauna cu **1** dacă $s\in\mathcal{L}$ şi cu **0** dacă $s\notin\mathcal{L}$.

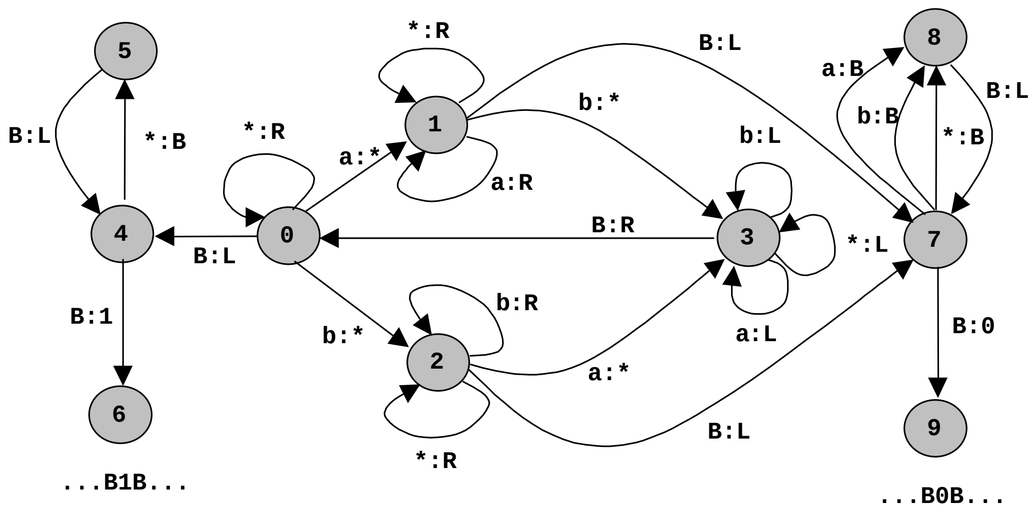

Figura 5. O maşină care recunoaşte limbajul $\mathcal{L}=\{s\in\{a,b\}^*\mid n_a(s)=n_b(s)\}$

Problemele date ca exemplu sunt modelabile prin funcții recursive. Următoarea teoremă, arată că Turing computabilitatea acoperă exact această clasă de funcții.

Teoremă (Turing). Clasa funcțiilor Turing-calculabile este exact clasa funcțiilor parțial-recursive.

Demonstrația este complicată şi depăşeşte domeniul analizei algoritmilor AA. O variantă a demonstrației poate fi găsită în [Tay 98].

3.3 Maşinile Turing ca modele utile în programare

În afara caracterizării funcțiilor numeric-teoretice, maşinile Turing ocupă un loc important în teoria automatelor şi limbajelor formale cu consecințe directe în proiectarea şi implementarea limbajelor de programare. Astfel, limbajele pot fi caracterizate după cum sunt acceptate sau recunoscute de diverse tipuri de Maşini Turing. Bunăoară, maşina **count** din secțiunea 3.2 este capabilă să distingă între şirurile de simboluri ce reprezintă propoziții în limbajul $\mathcal{L}=\{s\in\{a,b\}^*|\ n_a(s)=n_b(s)\}$ şi şirurile care nu sunt propoziții ale limbajului. Spunem că o asemenea maşină recunoaşte limbajul (sau că este un "recognizer" al limbajului). Dacă o maşină **M** se opreşte cu un anumit rezultat doar pentru propoziții $s\in L$ şi nu se opreşte pentru şiruri $s\notin L$, atunci maşina **M** este un acceptor al limbajului L.

Similar, poate fi caracterizată decidabilitatea unor proprietăți cu importanță paractică imediată ale diverselor clase de limbaje. Bunăoară, în cazul unui limbaj independent de context $L(G)$[5], generat conform unei gramatici **G**, problemele următoare nu sunt decidabile:

- Să se decidă dacă $L(G)$ este ambiguu (există cel puțin o propoziție derivabilă prin secvențe diferite de derivare).
- Să se decidă dacă $L(G)=T^*$, unde T^* este mulțimea tuturor şirurilor formate cu simbolurile terminale ale gramaticii **G**.
- Considerând două limbaje independente de context $L(G_1)$ şi $L(G_2)$, să se decidă dacă $L(G_1)\cap L(G_1)=\emptyset$.

Maşina Turing este apropiată de paradigma programării imperative, folosită de majoritatea limbajelor de programare comerciale. La baza unui asemenea limbaj este conceptul de stare a programului definită ca mulțimea valorilor variabilelor şi valoarea punctului de control al programului la un moment dat în cursul execuției şi corespunde configurației unei maşini Turing.

Un program imperativ – scris bunăoară în C – poate fi privit ca o codificare comodă a diagramei tranzițiilor unei maşini țintă Turing, diagramă ce controlează

[5] Limbajul $L(G)$ este privit ca mulțime a şirurilor de simboluri terminale (şiruri numite propoziții) ce sunt formate conform regulilor gramaticii **G**.

modificarea unei benzi direct adresabile ce joacă rolul memoriei datelor. Există o origine convențională a benzii, iar celulele sunt numerotate pornind de la **0** în raport cu originea. Variabilele folosite de program sunt grupuri de celule ale benzii direct accesibile prin instrucțiuni de forma L^i şi R^i, unde **i** este adresa primei celule din şirul compact de celule asociat varibilei. Starea programului este modificată folosind instrucțiuni de atribuire şi de control al fluxului execuției ce sunt macro-expandate în instrucțiunile de bază ale maşinii țintă Turing sau pot fi executate de maşini Turing interconectate ca părți ale maşinii țintă.

De asemenea, cu unele extinderi, ecartul dintre o maşină Turing şi un calculator numeric programabil poate fi diminuat. O asemenea extindere corespunde maşinii Turing universale: o maşină capabilă să preia ca intrare un program P_T, reprezentând codificarea unei maşini Turing **T**, şi o configurație **i** a benzii lui **T**. Maşina emulează execuția maşinii **T** cu configurația inițială **i** a benzii, deci practic execută **T(i)**. Maşina Turing universală este conformă conceptului de calculator cu program memorat.