

Partea de mai jos introduce unele noțiuni privind calculul în sens Turing, ca perspectivă a calculabilității efective.

Relații, funcții și calcul în sens Turing

1 Relații

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

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

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

O relație binară R peste M_1 și 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 N , în timp ce relația $<$ nu este totală (prechile de forma (x, x) nu sunt în relația $<$).

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

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

O relație binară R peste 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 și tranzitivă;
- *ordine parțială*, dacă este de preordine și este antisimetrică;
- *ordine totală*, dacă este de ordine parțială și este totală.
- *echivalență*, dacă este simetrică, reflexivă și tranzitivă.

Dacă R este o relație de echivalență peste o mulțime M , iar x este un element din M , definim clasa de echivalență 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

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

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

Fie R o relație binară peste o mulțime M . Închiderea reflexivă a relației R este cea mai mică¹ 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 și 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 $f: A \rightarrow B$, 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 și 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 și nici în B .

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

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

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 $\text{dom}(f) = A$. Altfel, dacă $\text{dom}(f) \subset A$, spunem că f este parțială (peste A).

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

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

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

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

Două funcții $f: A \rightarrow B$ și $g: A \rightarrow B$ sunt egale dacă și numai dacă au același domeniu și aceeași 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 și g sunt definite în x și au aceeași valoare, fie că
- $f(x) = \perp$ și $g(x) = \perp$

2.1 Funcții primitiv-recursive

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

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

unde x_i , $i=1, n$, sunt variabile cu valori în $\mathbb{N}$, iar v_j este fie o variabilă x_i , $1 \leq i \leq n$, fie o constantă din $\mathbb{N}$. Prin convenție, scriem $f = \text{Tr}[g]$ 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 $h: N^n \rightarrow N$, $g_i: N^n \rightarrow N$, $i=1, n$, dacă pentru orice vector de valori $\bar{x}^n = (x_1, x_2, \dots, x_n) \in N^n$ avem:

$$\begin{aligned} f(\bar{x}^n) &= h(g_1(\bar{x}^n), g_2(\bar{x}^n), \dots, g_n(\bar{x}^n)), \text{ dacă } \forall i \in 1..n \bullet g_i(\bar{x}^n) \neq \perp \\ f(\bar{x}^n) &= \perp, \text{ dacă } \exists i \in 1..n \bullet g_i(\bar{x}^n) = \perp \end{aligned}$$

Prin convenție, scriem $f = \text{Cp}[h, g_1, g_2, \dots, g_n]$ 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:

$$\begin{aligned} f(0, \bar{x}^n) &= g(\bar{x}^n) \\ f(m+1, \bar{x}^n) &= \begin{cases} h(m, \bar{x}^n, f(m, \bar{x}^n)), & \text{dacă } f(m, \bar{x}^n) \neq \perp \\ \perp & \text{altfel} \end{cases} \end{aligned}$$

Prin convenție, scriem $f = \text{Pr}[g, h]$ pentru a indica construcția lui f prin recursivitate primitivă pe baza funcțiilor g și h .

O funcție este *primitiv-recursivă* dacă poate fi obținută printr-un număr finit de pași de aplicare a operațiilor de transformare directă, compunere și recursivitate primitivă pornind de la funcțiile de bază **zero** (funcția constantă $\lambda x. 0$) și **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** și **succ** și este închisă sub operațiile de transformare directă, compunere și recursivitate primitivă. Prin urmare, o funcție primitiv-recursivă poate fi scrisă $f = E$, unde E conține doar transformări **Tr**, **Cp** și **Pr** aplicate funcțiilor **zero** și **succ**.

Se poate demonstra [Tay 98] că orice funcție primitiv-recursivă este totală². De exemplu, funcțiile definite mai jos sunt primitiv recursive și totale.

Convenim să scriem $\lambda x_1, x_2, \dots, x_n. \text{calcul}$ pentru a desemna o funcție n -ară cu parametri x_i , $i=1, n$, și cu valoarea $f(x_1, x_2, \dots, x_n)$ obținută conform calculului **calcul** (parametrizat în raport cu x_i , $i=1, n$). De asemenea, aplicația $f(x)$ va fi scrisă uneori și sub forma $(f \ x)$.

$$\begin{aligned} \text{id}: N &\rightarrow N \text{ id este funcția identitate} \\ \text{id}(0) &= 0 = \text{zero}(0) \\ \text{id}(n+1) &= \text{succ}(\text{id } n) = \lambda n, r. (\text{succ } r) \ (n, \text{id}(n)) \end{aligned}$$

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

$$\text{id} = \text{Pr}[\text{zero}, \lambda n, r. (\text{succ } r)] = \text{Pr}[\text{zero}, \text{Tr}[\text{succ}]]$$

pred: $\mathbb{N} \rightarrow \mathbb{N}$ **pred**(n) calculează predecesorul lui n, cu limitare la 0.
pred(0) = 0 = **zero**(0)
pred(n+1) = n = **id**(n) = $\lambda n, r. (\text{id } n) (n, \text{pred}(n))$

pred = $\text{Pr}[\text{zero}, \lambda n, r. (\text{id } n)] = \text{Pr}[\text{zero}, \text{Tr}[\text{id}]]$
zerop: $\mathbb{N} \rightarrow \mathbb{N}$ **zerop**(n) este 1 dacă n=0 și 0 altfel

zerop(0) = 1 = **succ** 0 = **succ**(**zero** 0) = **Cp**[**succ**, **zero**](0)
zerop(n+1) = 0 = $\lambda n, r. (\text{zero } n) (n, \text{zerop}(n))$

zerop = $\text{Pr}[\text{Cp}[\text{succ}, \text{zero}], \lambda n, r. (\text{zero } n)]$
= $\text{Pr}[\text{Cp}[\text{succ}, \text{zero}], \text{Tr}[\text{zero}]]$

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

plus(0,m) = m = **id**(m)
plus(n+1,m) = **succ**(**plus**(n,m)) = $\lambda n, m, r. (\text{succ } r) (n, m, \text{plus}(n, m))$

plus = $\text{Pr}[\text{id}, \lambda n, m, r. (\text{succ } r)] = \text{Pr}[\text{id}, \text{Tr}[\text{succ}]]$

dif: $\mathbb{N}^2 \rightarrow \mathbb{N}$ **plus**(n,m) calculează diferența n-m, cu limitare la 0

dif(n,m) = **dif'**(m,n) = **Tr**[**dif'**]

unde **dif'**(n,m) calculează diferența m-n, cu limitare la 0

dif'(0,m) = m = **id**(m)
dif'(n+1,m) = **pred**(**dif'**(n,m)) = $\lambda n, m, r. (\text{pred } r) (n, m, \text{dif}'(n, m))$

dif' = $\text{Pr}[\text{id}, \lambda n, m, r. (\text{pred } r)] = \text{Pr}[\text{id}, \text{Tr}[\text{pred}]]$
dif = **Tr**[**Pr**[**dif'**]]

ori: $\mathbb{N}^2 \rightarrow \mathbb{N}$ **ori**(n,m) calculează produsul n * m

ori(0,m) = 0 = **zero**(0)
ori(n+1,m) = **plus**(m, **ori**(n,m)) = $\lambda n, m, r. (\text{plus}(m, r)) (n, m, \text{ori}(n, m))$

ori = $\text{Pr}[\text{zero}, \lambda n, m, r. (\text{plus}(m, r))] = \text{Pr}[\text{zero}, \text{Tr}[\text{plus}]]$

mic: $\mathbb{N}^2 \rightarrow \mathbb{N}$ **mic**(n,m) este 1 dacă n<m și 0 altfel.

mic(n,m) = **dif**(1, **zerop**(**dif**(m,n)))

Dacă $n \geq m$ avem **mic**(n,m) = **dif**(1, **zerop**(0)) = **dif**(1, 1) = 0
Dacă $n < m$ avem **mic**(n,m) = **dif**(1, **zerop**(m-n)) = **dif**(1, 0) = 1

mic(n,m) = **dif**($\lambda n, m. (\text{succ}(\text{zero } n)) (n, m),$
 $\lambda n, m. (\text{zerop}(\text{dif}(m, n))) (n, m))$

$\text{mic} = \text{Cp}[\text{dif}, \text{Tr}[\text{Cp}[\text{succ}, \text{zero}]], \text{Tr}[\text{Cp}[\text{zerop}, \text{Tr}[\text{dif}]]]]$

O altă variantă, mai directă, a funcției mic este:

$\text{mic}(n, 0) = 0$
 $\text{mic}(0, m+1) = 1$
 $\text{mic}(n+1, m+1) = \text{mic}(n, m)$

2.2 Funcții parțial-recursive

O funcție $f: \mathbb{N}^n \rightarrow \mathbb{N}$ este obținută prin minimizare (sau μ -transformare) pe baza unei funcții $g: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$, 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 $\mu_m g(m, \bar{x}^n)$ desemnează cea mai mică valoare a parametrului m pentru care $g(m, \bar{x}^n) = 0$.

O funcție este *parțial-recursive* dacă poate fi obținută printr-un număr finit de pași de aplicare a operațiilor de transformare directă, compunere, recursivitate primitivă și minimizare pornind de la funcțiile de bază **zero** (funcția constantă $\lambda x. 0$) și **succ** (funcția succesor $\lambda x. x+1$). Alternativ, clasa funcțiilor parțial-recursive este cea mai mică clasă care conține funcțiile de bază **zero** și **succ** și este închisă sub operațiile de transformare directă, compunere, recursivitate primitivă și minimizare. Mulțimea funcțiilor parțial recursive include mulțimea funcțiilor primitiv-recursive și, totodată conține și funcții care nu sunt totale³. Funcțiile parțial-recursive care sunt totale se mai numesc și funcții recursive, deși acest termen este folosit uneori pentru a desemna o funcție recursivă fără a preciza clasa ei.

De exemplu, funcția $\text{div}(n, m)$ de mai jos calculează câtul împărțirii întregi dintre numerele naturale n și m și este parțial recursivă.

$\text{div}(n, m) = \mu_t (g(t, n, m))$
 $g(t, n, m) = \text{mic}(\text{plus}(\text{ori}(t, m), m), \text{succ}(n))$

adică $g(t, n, m) = t \cdot m + m < \text{succ}(n)$. 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 și minimizare pornind de la funcțiile de bază **zero** și **succ**.

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

Avem $\text{div}(15,7)=2$. Într-adevăr, $g(t,15,7)=1$ pentru $t=0,1$ și $g(2,15,7)=0$. Deci $\mu_t(g(t,15,7))=2$ și, implicit, $\text{div}(15,7)=2$.

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

3 Calcul în sens Turing

Mașina Turing, în particular mașina 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 științei calculatoarelor folosesc aparatul matematic și proprietățile mașinii Turing ca bază teoretică.

3.1 O descriere informală a mașinii Turing

Deși a premers explozia tehnologiei digitale, mașina Turing este apropiată ca structură și mod de funcționare de un calculator digital modern. Diferența principală este că o mașină Turing este construită în mod special pentru un anumit calcul⁴. Similar cu un calculator numeric, o mașină Turing particulară M este formată dintr-o unitate de control și o memorie.

Memoria este o bandă deplasabilă în fața unui cap H de citire/scriere. Banda mașinii 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 citită și rescrisă, rescrierea distrugând conținutul curent al celulei. În Γ există un simbol special, notat B (și numit "blank"), folosit pentru a marca celulele libere de pe bandă. Mișcarea benzii este în cuante, o cantă de mișcare 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) și mișcarea benzii. Operațiile (sau instrucțiunile) ce pot fi executate de mașină 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 și/sau σ_2 pot fi B .
- $\sigma : R$ Dacă simbolul curent scanat este $\sigma \in \Gamma$ atunci banda se deplasează spre dreapta cu o celulă în fața capului H .
- $\sigma : L$ Dacă simbolul curent scanat este $\sigma \in \Gamma$ atunci banda se deplasează spre stânga cu o celulă în fața capului H .

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

⁴ Sunt prezentate, pentru simplitate, doar astfel de mașini. Generalizarea există, mașina Turing universală fiind programabilă.

Diagrama tranzițiilor efectuate de unitatea de control a mașinii 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 mașină în momentul parcurgerii arcului. Procesul de schimbare a stării mașinii prin execuția unei instrucțiuni constituie un pas al calculului efectuat de mașină.

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

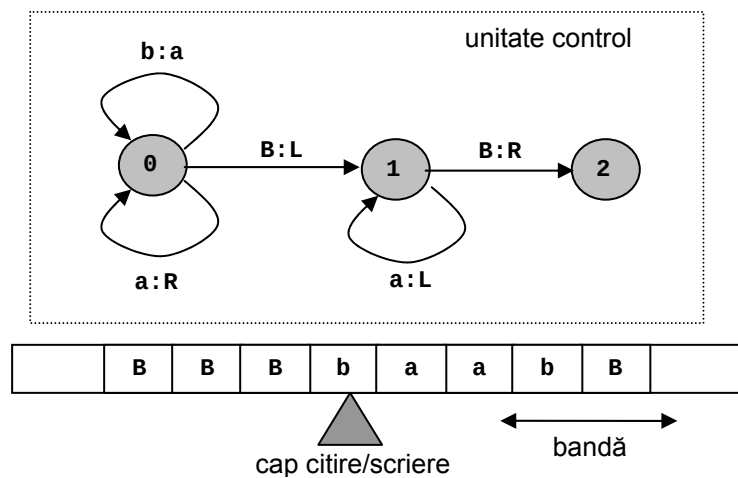


Figura 1 O mașină deterministă Turing

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

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

$$\dots Bsq_1\sigma s'B \dots \xrightarrow{\sigma:\alpha} \dots Bsq_j\sigma's'B \dots$$

Notația arată că plecând de la configurația curentă a benzii $\dots Bsq_1\sigma s'B \dots$, unde $s, s' \in \Gamma^*$, iar $\sigma \in \Gamma$ este simbolul curent scanat, și de la starea curentă q_1 atunci prin execuția instrucțiunii $\sigma:\alpha$ - cu acțiunea $\alpha \in \Gamma \cup \{L, R\}$ - configurația benzii devine $\dots Bsq_j\sigma's'B \dots$ - unde σ' este noul simbol scanat - și starea mașinii devine q_j .

Scrierea $\dots B$ desemnează un șir de simboluri aflate pe bandă la stânga capului H și care este format doar din simboluri B . Scrierea $B \dots$ desemnează un șir de simboluri aflate pe bandă la dreapta capului H și 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 mașina Turing din figura 1 (preluată din [Tay 98]). În configurația inițială a mașinii celulele benzii conțin simbolul **B** (sunt libere) cu excepția unei zone compacte ce memorează simbolurile **a** și **b**. Capul **H** este poziționat pe primul simbol diferit de **B** de pe bandă, iar starea inițială a mașinii este q_0 .

Calculul efectuat de mașină constă în înlocuirea tuturor simbolurilor **a** de pe bandă cu simboluri **b**. De asemenea, mașina trebuie să se oprească cu capul **H** poziționat ca în starea inițială: pe primul simbol non **B** de pe bandă. Calculul este descris de următorii pași:

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

unde $(i)^n$ desemnează aplicarea de n ori a instrucțiunii i . Se observă că pentru orice stare a mașinii cel mult o singură instrucțiune este executabilă într-un pas. Mașina este deterministă.

3.2 Mașini Turing deterministe

O mașină Turing deterministă M este un quintuplu $M = \langle Q, \Sigma, \Gamma, q_0, \delta \rangle$ unde:

Q este o mulțime finită și nevidă de stări. Există o stare $q_0 \in Q$ singulară, reprezentând starea inițială a mașinii M .

Γ și Σ sunt mulțimi finite nevide astfel încât $\Sigma \subseteq \Gamma$. Σ este alfabetul de intrare al lui M (simbolurile ce constituie configurația inițială a benzii lui M). Γ este alfabetul complet al benzii lui M (simbolurile ce pot fi folosite în orice configurație a benzii în cursul calculului executat de M).

$\delta: Q \times \Gamma \rightarrow (\Gamma \cup \{L, R\}) \times Q$ este o funcție parțială peste $Q \times \Gamma$ numită funcție de tranziție a lui M . Funcția δ primește o stare q_i și un simbol $\sigma \in \Gamma$ al benzii și reîntoarce o singură acțiune $\alpha \in \Gamma \cup \{L, R\}$ și o singură stare q_j . Funcția codifică pașii de calcul ce pot fi executați de M . Dacă M este în starea q_i și simbolul curent scanat este σ atunci mașina execută instrucțiunea $\sigma: \alpha$ și trece în starea q_j .

O mașină Turing poate fi folosită pentru a calcula funcții teoretice numerice (number-theoretic functions). Spunem că o funcție f este *Turing calculabilă* dacă există o mașină Turing care pentru fiecare configurație inițială a benzii, reprezentând argumentul x al funcției, fie se oprește cu o configurație a benzii ce corespunde rezultatului $f(x)$, dacă $x \in \text{dom}(f)$, fie nu se oprește, dacă $x \notin \text{dom}(f)$.

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

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

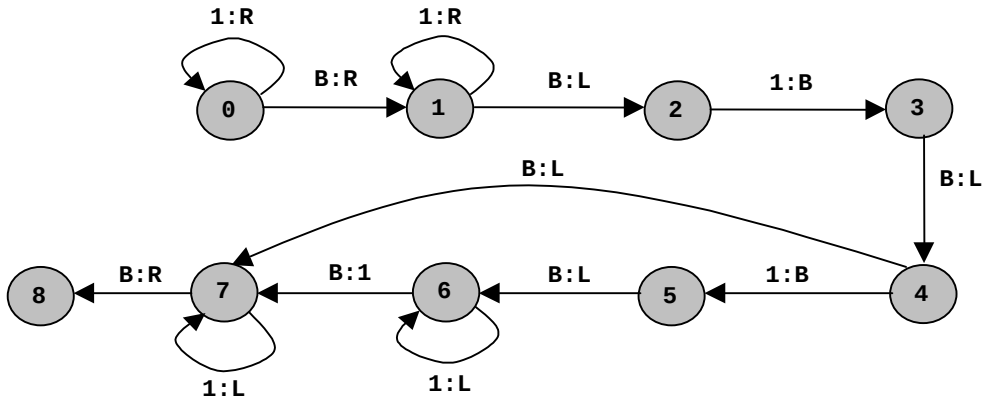


Figura 2. Adunarea a două numere naturale

Configurația inițială a benzii mașinii **plus** conține reprezentările numerelor n și 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.

$$\dots B \underset{\uparrow H}{1^{n+1}} B \underset{\uparrow H}{1^{m+1}} B \dots \xrightarrow{\text{plus}} \dots B \underset{\uparrow H}{1^{n+m+1}} B \dots$$

Pașii calculului $\text{plus}(n,m)$ au un efect simplu: ultimul 1 din m este preschimbă în B , iar simbolul B de separație dintre n și m este transformat în 1 atunci când $m \geq 1$.

Inițial $Bq_0 1^{n+1} B 1^{m+1} B$

- $(1:R)^{n+1} \rightarrow B 1^{n+1} q_0 B 1^{m+1} B$
- $B:R \rightarrow B 1^{n+1} B q_1 1^{m+1} B$
- $(1:R)^{m+1} \rightarrow B 1^{n+1} B 1^{m+1} q_1 B$
- $B:L \rightarrow B 1^{n+1} B 1^m q_2 1 B$
- $1:B \rightarrow B 1^{n+1} B 1^m q_3 B B$

Caz $m=0$ $B 1^{n+1} B q_3 B B$

- $B:L \rightarrow B 1^{n+1} q_4 B B B$
- $B:L \rightarrow B 1^n q_7 1 B B B B$
- $(B:L)^{n+1} \rightarrow q_7 B 1^{n+1} B B B B$
- $B:R \rightarrow B q_8 1^{n+1} 1 B B B B$

Caz $m > 0$ $B1^{n+1}B1^mq_3BB.$ — $B:L \rightarrow B1^{n+1}B1^{m-1}q_41BB$ — $1:B \rightarrow B1^{n+1}B1^{m-1}q_5BBB$ Caz $m=1$ $B1^{n+1}Bq_5BBB$ — $B:L \rightarrow B1^{n+1}q_6BBBB$ — $B:1 \rightarrow B1^{n+1}q_71BBB$ — $(1:L)^{n+2} \rightarrow q_7B1^{n+m+1}BBB$ — $B:R \rightarrow Bq_81^{n+m+1}BBB$ Caz $m > 1$ $B1^{n+1}B1^{m-1}q_5BBB$ — $B:L \rightarrow B1^{n+1}B1^{m-2}q_61BBB$ — $(1:L)^{m-1} \rightarrow B1^{n+1}q_6B1^{m-1}BBB$ — $B:1 \rightarrow B1^{n+1}q_71^mBBB$ — $(1:L)^{n+2} \rightarrow q_7B1^{n+m+1}BBB$ — $B:R \rightarrow Bq_81^{n+m+1}BBB$

Calcululele complicate pot fi descompuse în fragmente mai simple, iar mașinile care calculează fragmentele pot fi combinate pentru a constitui o mașină care realizează întregul calcul. Bunăoară, cuplarea în secvență a două mașini Turing corespunde compunerii funcționale. De exemplu, funcția **dublu**(n)= $2 \cdot n$ poate fi calculată de o mașină **dublu** = **plus** o **copie**, rezultată prin cuplarea în cascadă a mașinilor **plus** și **copie**. Mașina **copie** construiește o copie a numărului n , iar **plus** realizează adunarea **plus**(n, n).

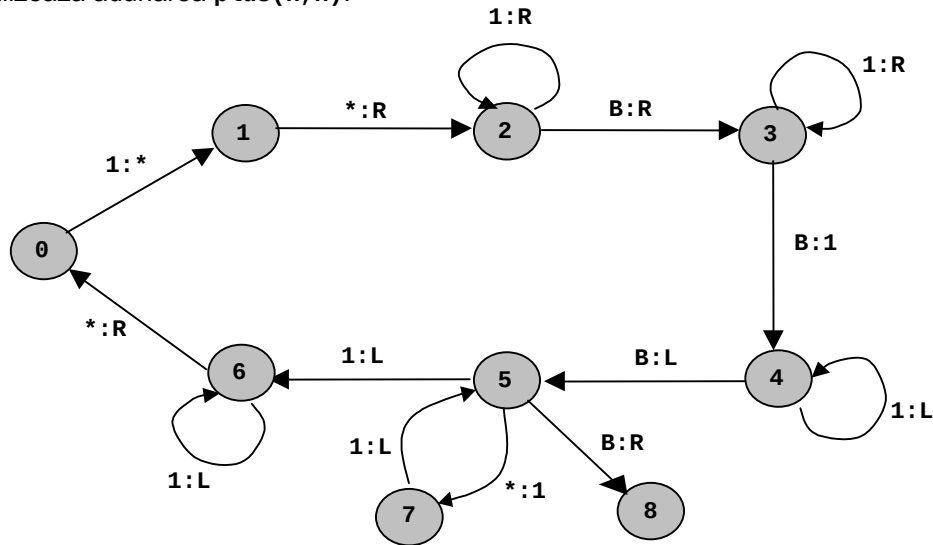


Figura 3. Copierea a unui număr natural

O posibilă implementare a mașinii **copie** este ilustrată în figura 3. Configurația inițială a mașinii este $Bq_01^{n+1}B$. Mașina se oprește cu configurația $Bq_81^{n+1}B1^{n+1}B$. Starea finală este întotdeauna q_8 .

The diagram shows two machines connected in sequence. The first machine, labeled 'mașină de copiere: copie', has an input 'n' and an output '8'. The second machine, labeled 'mașină de adunare: plus', has an input '8' and an output '2n'. A '1:1' connection is shown between the output of the first machine and the input of the second machine.

exemplu adițional, să construim o mașină Turing, numită **count**, care egalitatea dintre numărul de apariții ale două simboluri **a** și **b** într-un șir. Dacă notăm $n_x(s)$ numărul de apariții ale simbolului **x** în șirul **s**, atunci rezultatul de mașina **count** este $n_a(s)=n_b(s)$. Mașina **count** are o configurație inițială astfel încât capul **H** este poziționat pe primul simbol al șirului, iar benzii este **BsB**, unde **s** este șirul de scanat. Dacă $n_a(s)=n_b(s)$ la finală a benzii este **B1B**, iar dacă $n_a(s) \neq n_b(s)$ configurația finală a benzii este **B0B**, unde simbolurile **1** și **0** sunt diferite față de **a** și **b**. O posibilă implementare a mașinii **count** este ilustrată în figura 5. Mașina recunoaște propozițiile limbajului $\{s \mid n_a(s)=n_b(s)\}$: se termină întotdeauna cu **1** dacă $s \in L$ și cu **0** dacă $s \notin L$.

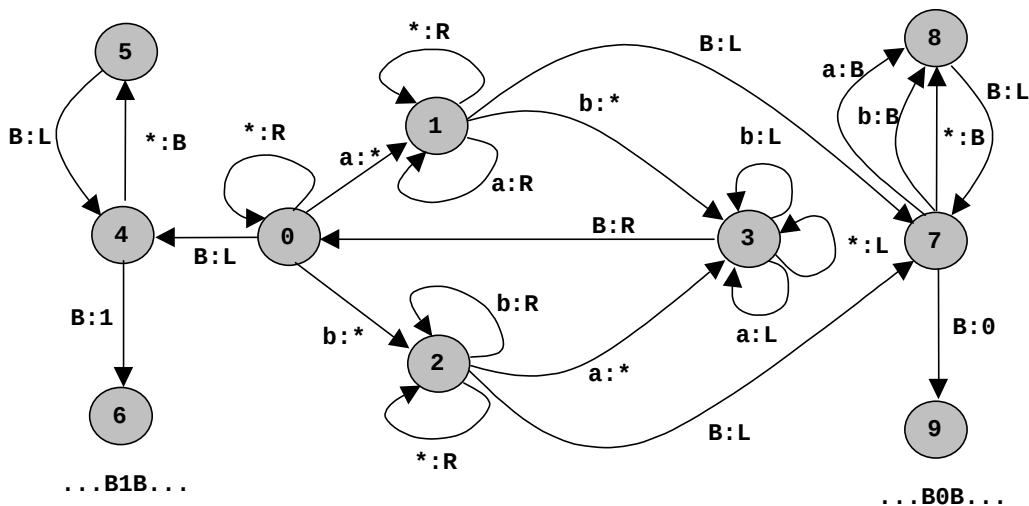


Figura 5. O mașină care recunoaște limbajul $L=\{s \in \{a, b\}^* \mid n_a(s)=n_b(s)\}$

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 și depășește domeniul analizei algoritmilor AA. O variantă a demonstrației poate fi găsită în [Tay 98].

3.3 Mașinile Turing ca modele utile în programare

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

Similar, poate fi caracterizată decidabilitatea unor proprietăți cu importanță practică 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 șirurilor formate cu simbolurile terminale ale gramaticii **G**.
- Considerând două limbaje independente de context $L(G_1)$ și $L(G_2)$, să se decidă dacă $L(G_1) \cap L(G_2) = \emptyset$.

Mașina 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 și valoarea punctului de control al programului la un moment dat în cursul execuției și corespunde configurației unei mașini Turing.

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

⁵ Limbajul $L(G)$ este privit ca mulțime a șirurilor de simboluri terminale (șiruri 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 și R^i , unde i este adresa primei celule din șirul compact de celule asociat variabilei. Starea programului este modificată folosind instrucțiuni de atribuire și de control al fluxului execuției ce sunt macro-expandate în instrucțiunile de bază ale mașinii țintă Turing sau pot fi executate de mașini Turing interconectate ca părți ale mașinii țintă.

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