TMS. LIMBAJ DE SPECIFCARE PENTRU MAŞINI TURING

Silviu Calinoiu

Universitatea Politehnica Bucuresti

Rezumat: Maşinile Turing au fost imaginate din nevoia de a formaliza conceptul de algoritm. Deşi definiţia lor este foarte "puternică" (în sensul definirii frontierei între calculabil şi necalculabil), din punct de vedere al expresivităţii definiţia este greu de folosit. De aceea, matematicienii au introdus noţiunea de maşină Turing compusă specificată prin intermediul unui graf ce conţine în noduri maşini Turing. Limbajul prezentat în articolul de faţă oferă ambele posibilităţi de specificare, maşini simple şi compuse. Totodată, datorită creşterii expresivităţii se oferă posibilitatea punerii în discuţie a unor analogii interesante cu limbajele de programare clasice, cum ar fi: abstractizare procedurală, recursivitate, flux de control ş.a.m.d.

Cuvinte cheie: maşini Turing, maşini elementare, compunere de maşini, conceptul de variabilă.

1. Introducere

Prezentul articol expune aspecte legate de specificarea maşinilor Turing. Motivele iniţiale pentru care a fost necesară studierea acestei probleme sunt legate de scrierea unui simulator de maşini Turing. Acest simulator, TMS, avea nevoie de un limbaj în care să i se specifice maşinile. Limbajul a evoluat de la o primă formă destul de rudimentară în care se puteau descrie maşini elementare, prin tranziţii, până la forma finală în care există posibilitatea de descriere a unor maşini Turing extrem de complexe, prin compunere. Mai mult, în limbajul TMS există conceptul de variabilă care permite crearea dinamică de maşini Turing, maşini ce efectuează operaţii ce depind de tranziţiile anterioare..

Secţiunea 1. Maşini Turing elementare, conţine o prezentare a conceptului de maşină elementară împreună cu modalităţile de specificare în TMS.

Secţiunea 2. Compunerea maşinilor Turing, prezintă fundamentele teoretice, legate de compunerea masinilor, reprezentarea prin grafuri a maşinilor compuse si modalităţile oferite de TMS pentru a compune masini.

Secţiunea 3. Conceptul de variabilă în TMS, continuă prezentarea din secţiunea 2 despre reprezentarea prin grafuri şi explică necesitatea introducerii conceptului de variabilă, precum şi modul de specificare în TMS.

Secţiunea 4. "Programarea" maşinilor Turing, prezintă modul în care se poate aborda cât mai eficient specificarea unei maşini complexe. Concluzia principală este că programarea in TMS se face conform acelorasi principii din limbajele "traditionale", adică modularizare, abstractizare etc.

Secţiunea 5. **Concluzii**, conţine o discuţie a aplicabilitaţii limbajului precum şi o prezentare a unui simulator TMS care a fost dezvoltat o dată cu proiectarea limbajului.

2. Maşini Turing elementare

Aspectele formale, legate de maşinile Turing sunt prezentate conform [1], unde de fapt se face o sinteză a diferitelor lucrări in domeniu. In esenţă, o maşină Turing este formată dintr-o unitate de control şi o bandă. Comunicaţia între cele două se face prin intermediul unui cap de citire/scriere. Unitatea de control operează prin efectuarea de paşi elementari. În fiecare pas se produc două acţiuni, în funcţie de starea curentă şi simbolul aflat sub capul de citire/scriere:

1. unitatea de control trece într-o noua stare;
2. se scrie un simbol pe bandă în poziţia capului de citire/scriere

sau

se mută capul de citire cu o poziţie la stânga sau la dreapta.

Banda are o limită stângă, dar se întinde nelimitat în dreapta. O maşină Turing primeşte ca intrare un şir înscris începând cu capătul din stânga al benzii. Restul benzii conţine iniţial simboluri blanc. Există o stare specială de *halt* utilizată pentru a semnala sfîrşitul operarii, notată cu *h*. Alte simboluri cu semnificaţie specială sunt : # - pentru simbolul blanc, L - pentru mutarea capului în stânga şi R - pentru mutarea capului în dreapta. Definiţia formală pentru o maşină Turing este:

Definiţia 1. (Maşină Turing)

O maşină Turing este un cuadruplu (K,Σ,δ,s), unde

K este o mulţime finită de stări ce nu conţine starea *h;*

Σ este un alfabet de simboluri, ce conţine simbolul #, dar nu conţine simbolurile *L* şi *R*;

$s \in K$ este *starea iniţială*;

δ este o funcţie definită pe $K \times \Sigma$ cu valori în $(K \cup \{h\}) \times (\Sigma \cup \{L,R\})$.

Dacă $q \in K$, $a \in \Sigma$ şi $\delta(q,a)=(p,b)$ atunci când maşina se află în starea *q* şi sub capul de citire/scriere se află simbolul *a*, se intră în starea *p* şi dacă *b* este un simbol din Σ se scrie pe bandă în locul lui *a* sau dacă este *L* sau *R*, atunci se mută capul cu o poziţie în direcţia corespunzătoare. Deoarece δ este o funcţie, operarea maşinii este deterministă şi se termină atunci când se intră în starea halt sau se încearcă mutarea capului în stânga limitei stânga a benzii.

În limbajul TMS, o maşină Turing se specifică astfel:

(**simple** nume_maşină stare_iniţială
tranziţie_1
tranziţie_2
...)

unde,

nume_maşină - numele maşinii, format din litere minuscule şi majuscule, cifre şi caracterele #,_.

Numele pentru o maşină trebuie să înceapă cu o majusculă.

stare_iniţială - numărul corespunzător stării iniţiale. Stările unei maşini sunt referite prin numere întregi, fără semn.

tranziţie - fiecare tranziţie se specifică astfel:

(stare simbol → stare simbol)

unde,

stare - număr sau în cazul stării finale poate fi simbolul *H*.

simbol - un caracter, simbolurile admise de TMS sunt minusculele, cifrele şi simbolul #. In secţiunile următoare se va vedea că, în loc de simbol, se poate folosi şi o construcţie specială, cu care se specifică valoarea unei variabile.

În specificarea unei tranziţii se pot folosi pentru comoditate şi urmatoarele variante:

1. în loc de simbol se specifica semnul * având semnificaţia de orice caracter. O asemenea facilitate este utila când se doreşte specificarea unei tranziţii care se efectueaza indiferent de simbolul curent;

2. când se face tranziţia în starea halt, nu este nevoie să se mai specifice un simbol.

Trebuie remarcat că aceste variante nu modifică cu nimic definiţia iniţială a unei maşini Turing. Ele nu reprezintă decât un mod mai compact de exprimare. Pentru prima variantă, tranziţia:

(0 * → 0 #) exprimă de fapt, mulţimea de tranziţii

{ (0 <s> → 0 #) | unde <s> este un simbol oarecare aparţinând alfabetului de intrare, }. Pentru a doua variantă, tranziţia:

(0 # → H)

s-ar putea expima astfel:

(0 # → H #)

Exemplul 1

Următoarea specificaţie TMS descrie o maşina Turing, care şterge caracterele de pe bandă, adică le înlocuieşte cu simbolul #.

(**simple** Wipe_all 0
(0 # → H)
(0 * → 1 #)
(1 * → 0 R))

Pentru a putea simula această maşină in specificaţia TMS trebuie să se specifice conndiţiile iniţiale. Acest lucru se face cu construcţia:

(**simulate** nume_maşină banda_iniţială poziţia_iniţială)

unde,

nume_maşină - numele maşinii de simulat

banda_iniţială - conţinutul iniţial al benzii, specificat printr-succesiune de simboluri valide (minuscule,cifre,#).

poziţia_iniţială - număr reprezentând poziţia iniţiala a capului de citire/scriere, unde în poziţia externă stînga este 0.

Deci, pentru ca specificaţia în cazul anterior să fie completa, ea trebuie completată, de exemplu, cu:

(simulate WipeAll #abcdefabcdef# 0)

Exemplul 2

Următoarea specificaţie TMS descrie o maşină Turing, care calculează funcţia, f:N→N,
unde f(n) = 0 , n <= 2 şi f(n) = n-2, n > 2.

```
(simple Fun 0
    (0 # → 1 L)      ; mutare stânga
    (1 # → H R)      ; dacă # atunci halt
    (1 i → 2 #)      ; primul i se şterge
    (2 # → 3 L)      ; mutare stânga
    (3 # → H R)      ; dacă # atunci halt
    (3 i → H #))     ; al doilea i se şterge şi halt

(simulate Fun #iiiiiiiiii# 11)
```

Pentru a face specificaţiile TMS mai uşor de înţeles se pot utiliza comentarii. Prin caracterul ; se comenteaza linia respectivă de la caracter până la sfârşitul liniei.

Exemplul 3

Următoarea specificaţie TMS descrie o maşina Turing care decide limbajul,
L = { w ∈ {a,b}* | w conţine cel puţin un a }.

```
(simple AtLeastOne 0
    (0 # → 1 L)
```

```
(1 a → 5 #)      ; 5- stare OK, a găsit
(1 b → 2 #)      ; şterg simbol
(2 # → 1 L)      ; merg mai departe
(1 # → 3 R)      ; am atins limita stânga
(3 # → 3 n)      ; spun că nu
(3 n → H R)
;; stările pentru un a găsit
(5 # → 6 L)
(6 # → 7 R)
(6 * → 5 #)
(7 # → 7 y)
(7 y → H R))

;(simulate AtLeastOne #bbbbbbbbbabbb# 14)
; da - #y#
(simulate AtLeastOne #bbbbbbbbbbbbb# 14)
; nu - #n#
```

3. Compunerea maşinilor Turing

Maşinile Turing sunt nişte dispozitive generale. Pentru a putea exprima operaţii mai complicate trebuie un mecanism mai puternic decât cel oferit de maşinile elementare. De fapt, facând o paralelă cu limbajele de programare "tradiţionale", trebuie făcut pasul de la limbajul de asamblare la limbajul de nivel înalt (funcţional). Maşinile elementare care vor fi folosite au urmatoarele specificatii TMS:

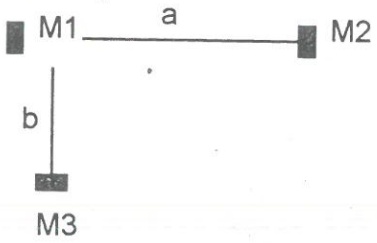
```
(simple Ml 0      (0 * → H L))  ; mută cap în stânga
(simple Mr 0      (0 * → H R))  ;mută cap în dreapta
(simple W# 0      (0 * → H #))  ; scrie #
(simple Wa 0      (0 * → H a))  ; scrie a
```

Maşinile Turing pot fi combinate într-un mod asemanător automatelor finite. Maşinile individuale sunt asemănătoare stărilor, iar maşinile sunt conectate prin arce la fel ca într-un automat finit. Conexiunea între doua maşini este parcursă numai când prima maşină intră în starea halt. Când acest lucru se întâmplă a doua maşină porneşte cu banda şi capul de scriere/citire în starea lăsată de prima maşină. Reprezentarea grafică pentru "înlănţuirea" a două maşini este:

Maşina de mai sus se specifică în TMS astfel:

```
( complex Exemplu
      ( M1 M2 ))
```

Parcurgerea unui arc într-un astfel de graf se poate face condiţionat în funcţie de simbolul curent aflat sub capul de citire/scriere în momentul intrării in starea halt. Iată un exemplu:



După ce maşina M1 intră în starea halt, dacă simbolul curent este *a*, începe rularea maşinii M2, iar dacă este *b*, începe rularea maşinii M3. Această maşină are specificaţia TMS:

```
( complex Exemplu
      ( M1 M2 (if a))
      ( M1 M2 (if b)))
```

În general, construcţia sintactică este:

```
( complex nume_maşină
      arc_1
      arc_2
      ... )
```

unde *arc* este un arc din graful de reprezentare. Singura condiţie este ca primul arc specificat să conţină maşina iniţiala. În funcţie de tipul arcului (etichetat sau nu) există trei posibilităţi :

1. neetichetat (nume_maşină nume_maşină)
2. etichetat cu *simbol* (nume_maşină nume_maşină (**if** simbol))
3. etichetat cu $\neq$ *simbol* (nume_maşină nume_maşină (**if not** simbol))

Iată în continuare cîteva specificaţii pentru nişte maşini ce vor fi utile în viitor:

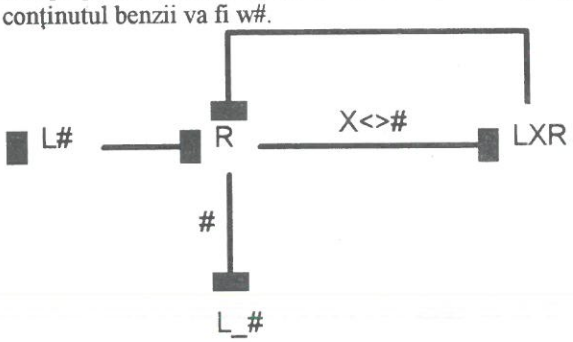
```
(complex R#      (Mr Mr (if not #)))
(complex L#      (Ml Ml (if not #)))
(complex Rn#     (Mr Mr (if #)))
(complex Ln#     (Ml Ml (if #)))
```

R# - găseşte primul simbol # în dreapta poziţiei curente
L# - găseşte primul simbol # în stânga poziţiei curente
Rn# - găseşte primul simbol $\neq$ # în dreapta poziţiei curente
Ln# - găseşte primul simbol $\neq$ # în stânga poziţiei curente

4. Conceptul de variabilă în TMS

În această secţiune se va arăta că, pentru ca notaţia cu grafuri să fie cu adevărat puternică, mai trebuie încă ceva. Mai jos este prezentată o maşină care face deplasarea la stânga cu o poziţie. Deci, dacă primeşte pe bandă #w# unde w este un şir ce nu conţine # şi capul se

află pe primul # după w, atunci când maşina se opreşte conţinutul benzii va fi w#.



Maşinile Turing din noduri sunt :

L# - cea din secţiunea anterioară;

R - mută cap dreapta; în secţiunea anterioară a fost notată Mr, pentru că în TMS caracterul R este rezervat pentru specificarea în tranziţii a mutării capului.

L_# - mută stânga şi scrie #;

LσR - mută stânga, scrie σ, mută dreapta.

Problema constă în scrierea nodului LσR. Se observă că e vorba de o maşină care se schimbă dinamic în funcţie de simbolul pe care se face tranziţia. Acest artificiu este conceptual asemănător noţiunii de variabilă. Tocmai pentru a putea exprima astfel de artificii s-a introdus în TMS conceptul de variabilă prin doua construcţii sintactice, *get* şi *put*. Trebuie remarcat că este de fapt o notaţie mai puternică şi deci, nu este vorba de o creştere a expresivităţii definiţiei iniţiale. Maşina LσR, împreună cu arcul de intrare ar putea fi specificate printr-o mulţime de maşini Turing, de forma L<s>R, unde <s> este orice simbol din alfabetul de intrare.

Put are forma **(put nume_variabila)** şi poate să apară în specificarea unui arc dintr-o maşină complexă pe ultima poziţie. *Put* are semnificaţia unei instrucţiuni de atribuire. Valoarea care se atribuie variabilei este simbolul pe care se face parcurgerea arcului. Numele variabilei respectă aceleaşi reguli ca numele de maşini (începe cu majusculă).

Get are forma **(get nume_variabila)** şi poate să apară oriunde apare un simbol. Deci *get* poate fi folosit în tranziţiile din maşinile elementare sau în condiţiile din specificarea arcelor pentru maşinile complexe. *Get* are semnificaţia de valoare a variabilei la un moment dat. Iată specificaţia TMS pentru maşina de mai sus:

```
( simple LxR  0
        ( 0 * --> 1 L )
        ( 1 * --> 2 ( get Edge ))
        ( 2 * --> H R))
```

```
( complex Shift_to_left
        ( L#    Mr )
        ( Mr    LxR     (if not #)
(put Edge))
        ( LxR   Mr)
        ( Mr    Ml      (if #))
        (Ml     W#))
```

5. "Programarea" maşinilor Turing

Pentru cititorul care are experienţa de programare scrierea de specificaţii TMS ar trebui să fie deja clară. Pentru fiecare nod din graf care nu este o masină elementară se scrie câte o "procedură" adică o definiţie de maşină Turing. Pentru a facilita reutilizarea de maşini Turing scrise anterior s-a introdus în TMS posibilitatea de includere de biblioteci prin construcţia,

#include nume_fişier

Numele de fisier se specifică fără a fi încadrat de ghilimele sau de alte caractere. Astfel, maşinile elementare pot fi puse într-un fişier, de exemplu *library.tm* , iar in specificatie se introduce linia,

#include library.tm

Trebuie subliniat un aspect legat de mecanismul de simulare al maşinilor compuse. Dacă există două sau mai multe noduri în graful de reprezentare care conţin aceleaşi maşini, atunci trebuie neaparat create doua instanţe ale maşinii respective, pentru ca maşina principală să funcţioneze corect. Deci, pentru o maşina de tipul,

specificaţia TMS corectă este,

```
(simple Do_nothing 0  (0 * --> H))
(complex M1           (M  Do_nothing)
(complex M2           (M Do_nothing)
(complex Main         (M1 M2))
```

Cititorul avizat va vedea desigur similitudinea problemei descrise mai sus cu problema înregistrărilor de activare pentru apelurile recursive din limbajele de programare tradiţionale.

În continuare va fi prezentat un nou exemplu de specificaţie TMS, pentru o maşină de copiere. Aceasta primeşte pe bandă şirul #w# şi îl transformă în #w#w#, deci creează un duplicat al şirului iniţial. Specificaţia TMS este:

```
#include library.tm
(simple Copy_aux 0
        (0 * --> 1 #)     ; #
        (1 * --> 2 R)     ; R#
        (2 # --> 3 #)     ;
```

```
        (2* → 2 R)              ;
        (3 * → 4 R)             ; R#
        (4 # → 5 #)             ;
        (4 * → 4 R)             ;
        (5 * → 6 (get X))       ; x
        (6 * → 7 L)             ; L#
        (7 # → 8 #)             ;
        (7 * → 7 L)             ;
        (8 * → 9 L)             ; L#
        (9 # → 10 #)            ;
        (9 * → 9 L)             ;
        (10 * → H (get X)))     ; x
(complex Copy
        (L#             Mr)
        (Mr             R# (if #))
        (Mr     Copy_aux (if not #) (put X))
        (Copy_aux       Mr))
(simulate Copy #abcde# 6)
```

6. Concluzii

Limbajul şi simulatorul asociat au un scop didactic. Ele s-au dovedit un instrument extrem de atractiv şi eficace în înţelegerea raportului dintre maşinile Turing şi maşinile de calcul convenţionale. Cu toate acestea, limbajul este interesant şi sub aspect teoretic pentru că prin introducerea conceptului de variabilă s-a ajuns pînă la a specifica grafuri care au în noduri maşini Turing care acţionează în funcţie de istoria anterioară a parcurgerii grafului, ceea ce este într-un anume sens remarcabil.

Simulatorul TMS este un program care afişează, în mod grafic, în fiecare moment, conţinutul benzii, poziţia capului de citire/scriere şi tranziţia curentă. Maşina Turing din specificaţia primită de simulatorul TMS poate fi rulată pas cu pas (ca un trasor de programe) sau direct, caz în care se opreste doar când ajunge în halt. Referinţele [2] şi [3] au fost introduse mai ales pentru aspectele legate de implementarea simulatorului, el avînd un mic analizor lexical şi unul sintactic care prelucrează specificaţia transmisă într-un fişier.

Bibliografie

1. **LEWIS II., PAPADIMITRIOU C.** - Elements of the Theory of Computation, Prentice-Hall, Inc., Englewood Cliffs, NJ, SUA, 1991.
2. **AHO, A. V., R. SETHI, J. D. ULLMAN:** Compilers-Principles, Techniques, and Tools, Addison -Wesley Publishing Company, Cambridge, MA, SUA, 1986.
3. **LEVINE J., T. MASON, D. BROWN:** Lex & Yacc - O'Reilly & Associates, Inc. 1990.

INSTITUTUL DE CERCETARE IN INFORMATICA
B-dul Maresal Averescu 8-10, sector 1, cod 71316
Bucuresti, tel: 617.79.78, fax: 312.85.39

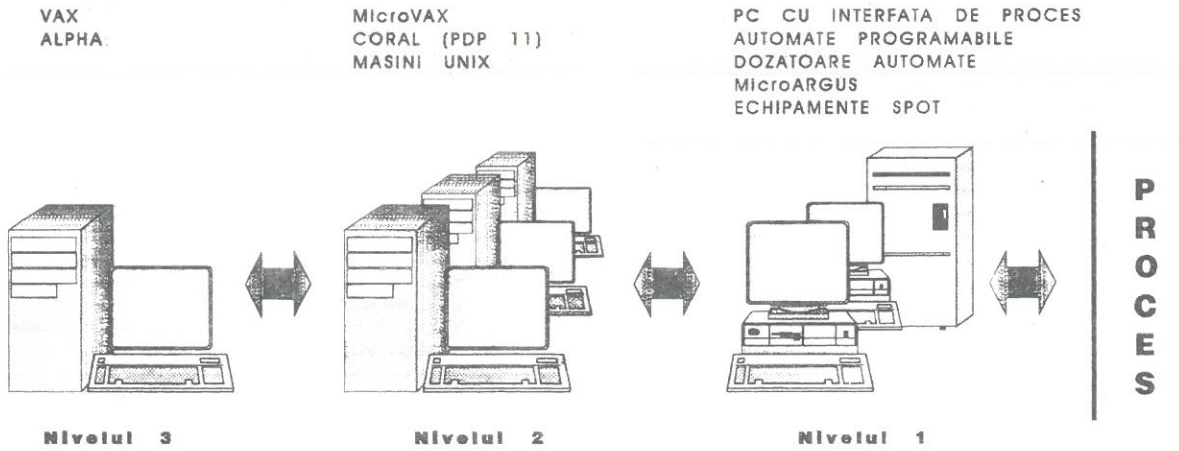
LABORATORUL DE SISTEME IN TIMP REAL
tel: 665.28.90; 665.60.60/168 sau 214

PELET

PELET este un sistem destinat supravegherii/conducerii instalațiilor tehnologice. Produsul, implementat deja la Fabrica de Peletizare a minereului de fier de la Mangalore - India, permite un control permanent al parametrilor tehnologici, cu alarmare în caz de depășire a unor limite tehnologice. Cu toate că este specializat pe aplicație, ***PELET*** se poate adapta cu ușurință și pentru alte aplicații similare. ***PELET*** se aplică în industria metalurgică, în fabricile de peletizare a minereului de fier.

PELET realizează:
- comanda și urmărirea fluxurilor de mecanisme
- supravegherea procesului de peletizare
- alarmarea situațiilor de avarie
- dialogul cu operatorul tehnolog privind starea procesului tehnologic

PELET asigură:
- reducerea timpilor de staționare a utilajelor
- creșterea nivelului de siguranță în exploatarea utilajelor tehnologice
- îmbunătățirea modului de supraveghere și comandă a procesului tehnologic
- optimizarea utilizării materiilor prime



Echipamente:
- minicalculatoare compatibil PDP 11 (CORAL 4030/4021, I 102/106)
- sisteme distribuite MICRO - ARGUS sau microcalculatoare de proces SPOT 83
- automate programabile AP-201

Sistem de operare:
- RSX 11M pentru minicalculatoare
- S 83 (elaborat de ICI) pentru microcalculatoare de proces SPOT 83

Livrare:
Sistemul se livrează "la cheie" (echipamente plus programe de bază și aplicative), în conformitate cu specificațiile beneficiarului.
O dată cu produsul se oferă servicii de :
- instalare și punere în funcțiune a sistemului
- școlarizare a personalului beneficiarului
- asistență tehnică

Garanție:
Se asigură garanția produsului pe o perioadă de 12 luni de la data livrării.