BAZE DE CUNOŞTINȚE

SISTEME DE RESCRIERE

MIHAELA COLHON

PROCESAREA LIMBAJULUI NATURAL

Cele mai multe rezultate s-au obtinut in procesarea de texte redactate in limba engleza, aceasta limba considerandu-se "context-free", ca atare e usor de procesat folosindu-se gramatici formale.

In general, orice modul de NLP trebuie sa aiba urmatoarele componente:

- **Analizator sintactic** al intrarilor primite (contine reguli gramaticale care definesc sintaxa limbii)
- **Analizator semantic** al cunostintelor descrise in text
- **Vocabularul limbii** care se proceseaza

LINGVISTICA COMPUTATIONALA

Există două categorii distincte de limbaje:

- Limbaje artificiale
- Limbaje naturale

"*Dacă orice interacțiune cu un calculator poate fi exprimată în termeni lingvistici atunci de ce nu am putea realiza aceste interacțiuni în limbaj natural în loc de a utiliza limbaje artificial construite (limbaje de programare)?*"

D. Dumitrescu, Principiile inteligenței artificiale, 1999

LINGVISTICA COMPUTATIONALA

- **Limbajele artificiale** (de exemplu, limbajele formale) sunt generate pe baza unor reguli de construcție.
- Din punct de vedere formal, mecanismul de definire al **limbajelor naturale** nu este elucidat până în acest moment. Există însă câteva modele matematice pentru aproximarea limbajelor naturale.

Daca limbajele naturale sunt limbajele pe care le folosesc oamenii pentru a comunica intre ei, limbajele formale sunt formalisme matematice pe care informaticienii, logicienii şi matematicienii le definesc şi le studiază pentru scopuri variate.

GRAMATICI FORMALE. GRAMATICI CONTEXT FREE

O gramatică este definită drept un sistem:

$$G = (V_N, V_T, P, S)$$

- V_N o mulțime de variabile (ex. S, A, B, ...)
- V_T este o mulțime de terminale (ex. 0, 1, a, b, ...)
- P este o mulțime de producții
- S este simbolul inițial al gramaticii, $S \in V_N$

Producțiile : $A \rightarrow w$ pentru

A - variabilă

w - o secvență de variabile și terminale

Exemplu: $G = (V_N, V_T, P , S)$ unde $V_N = \{S, A\}$, $V_T = \{ 0, 1 \}$ și producțiile:

S -> 0A | 0

A -> 10A

Pe baza acestei gramatici se generează construcțiile 0(10)*

GRAMATICI CONTEXT FREE. GRAMATICI CLAUZAL DEFINITE

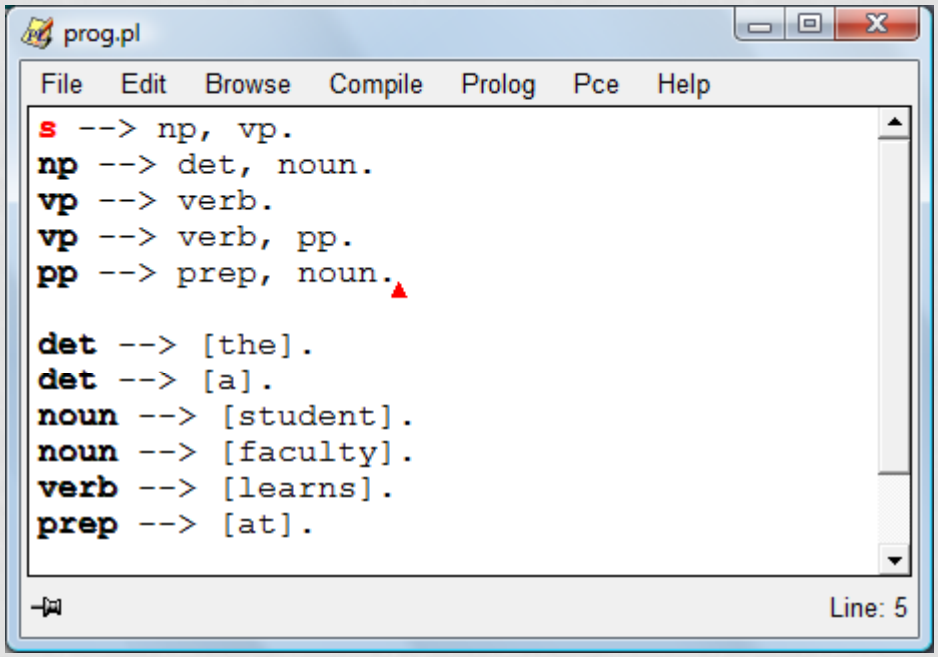
Procesarea limbajului natural in Prolog se face construind Gramatici Clauzal Definite.

NLP în Prolog implică:

- Cunostinte de baza despre Prolog:
  - Ce este un fapt, o regula, cum se defineste recursia, procesarea listelor
- Reprezentarea cunostintelor lingvistice, a regulilor gramaticale
- Implementarea gramaticilor clauzal definite

```
prolog :- tutorial
```

GRAMATICI CLAUZAL DEFINITE



Daca dorim să generăm toate propozitiile corecte în raport cu gramatica definită, atunci interogarea este:
?- s(L, []).
L= [the, student, learns]
…

In locul separatorului :- se foloseste - -⟩
Exemplu de interogare:
?- s([the, student, learns, at, faculty],[]).
true

DEFINITE CLAUSE GRAMMAR

DCG ofera o modalitate mai simpla de scriere de parsing-uri cu diferente de liste. Este sarcina preprocesorului DCG de a transforma clauzele DCG in clauze Prolog prin adaugarea listelor de intrare si de iesire corespunzatoare.

Sintaxa DCG:

- Operatorul --> indica o regula DCG si inlocuieste operatorul :- folosit in clauzele Prolog. Preprocesorul va adauga doi parametri in plus: lista care se proceseaza si lista rezultat.

  De exemplu clauza:

  sentence(S, Lrez):- np(S, Lrez1), vp(Lrez1, Lrez).

  se poate transforma intr-o regula DCG dupa cum urmeaza:

  sentence --> np, vp.

Fiecare element din stanga unei reguli este considerat **goal** si la el preprocesorul va adauga de asemenea cele doua liste ca paramentrii

DEFINITE CLAUSE GRAMMAR

program.pl

File Edit Browse Compile Prolog Pce

```
s --> np(Nr), vp(Nr).
np(Nr) --> det(Nr), noun(Nr).
vp(Nr) --> verb(Nr).
vp(Nr) --> verb(Nr), np(_).

det(sg) --> [the].
det(pl) --> [the].
noun(sg) --> [woman].
noun(sg) --> [man].
noun(pl) --> [women].
noun(pl) --> [men].
verb(sg) --> [shoots].
verb(pl) --> [shoot].
```


```
| ?- s(L,[]).
L = [the, woman, shoots] ;
L = [the, woman, shoots, the, woman] ;
L = [the, woman, shoots, the, man] ;
L = [the, woman, shoots, the, women] ;
L = [the, woman, shoots, the, men] ;
L = [the, man, shoots] ;
L = [the, man, shoots, the, woman] ;
L = [the, man, shoots, the, man] ;
L = [the, man, shoots, the, women] ;
L = [the, man, shoots, the, men] ;
L = [the, women, shoot] ;
L = [the, women, shoot, the, woman] ;
L = [the, women, shoot, the, man] ;
L = [the, women, shoot, the, women] ;
```

Folosirea unor predicate Prolog obisnuite se face prin includerea lor intre acolade { }. De exemplu:

```
np --> noun, {write('found noun')}.
```

Parantezele drepte se folosesc pentru includerea simbolurilor terminale ale gramaticii:

```
noun --> [girl].
```

ANALIZATOARE SINTACTICE. PARSER SINTACTIC

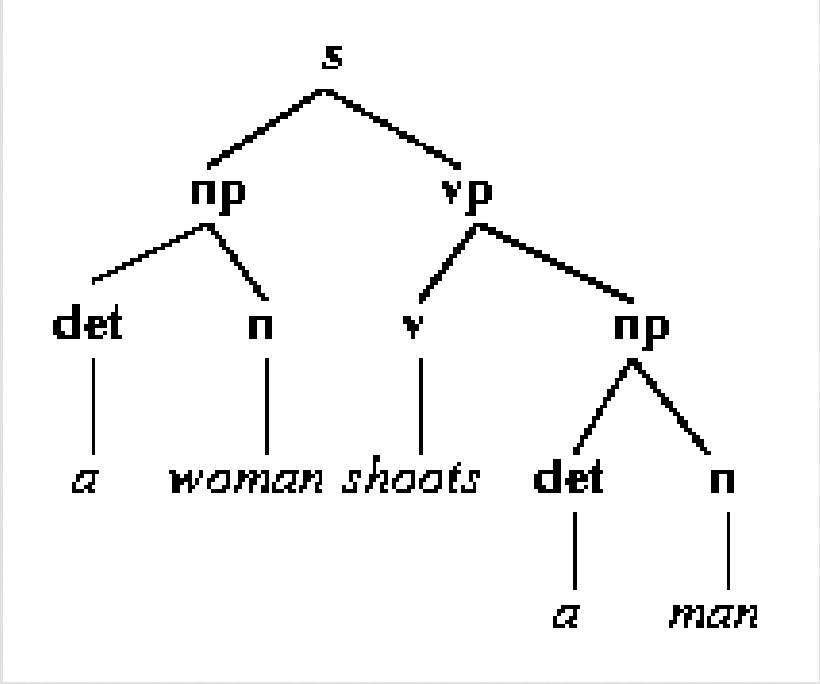
- Un parser se defineste in raport cu o gramatica sau un alt mecanism de analiza a constructiilor unui limbaj natural. Rolul acestuia este de a *identifica* acele reguli (din gramatica) care stau la baza structurii propozitiei ce se analizeaza. Actiunea lui se concretizeaza intr-un proces de cautare a acelor structuri sintactice posibile care se potrivesc cu structura propozitiei.
- Exista mai multe tipuri de cautare, cele mai uzuale fiind **depth-first** si **breadth first**. Cautarea depth-first (cautare in adancime cu revenire prin backtracking) este implementata direct in Prolog.
- Rezultatul cautarilor lansate de parser confirma sau infirma **corectitudinea sintactica** a propozitiei analizate in raport cu gramatica utilizata.
- Traseul rezolutiv al unui algoritm de analiza sintactica (sau al unui parser) este de cele mai multe ori reprezentat sub forma unui arbore – **parse tree**.

ARBORE SINTACTIC. EXEPLIFICARE

Consideram urmatoarea gramatica clauzala:

```
s -> np vp
np -> det n
vp -> v np
vp -> v
det -> a
det -> the
n -> woman
n -> man
v -> shoots
```

Arborele de analiza sintactica corespunzator unei cautari in adancime este:



TIPURI DE GRAMATICI CONTEXT-FREE. GRAMATICI LEFT/RIGHT LINIAR

O gramatică context-free se numește:

- *Left-liniar* dacă producțiile sunt de forma:

  $A \rightarrow Bw$ sau $A \rightarrow w$

- *Right-liniar* dacă producțiile au forma:

  $A \rightarrow wB$ sau $A \rightarrow w$

$A, B \in V_N$ și $w \in V_T^*$

O gramatică *right-liniar* sau *left-liniar* se mai numește gramatică regulară.

TIPURI DE GRAMATICI CONTEXT FREE. GRAMATICI RECURSIVE

În aceste gramatici se pot genera construcții infinite.

De exemplu, producția:

$s \rightarrow s$ and s

poate genera derivările:

$s \rightarrow s$ and $s \rightarrow s$ and s and s and $s \rightarrow \ldots$

Cu aceste producții se pot genera (recunoaște) propoziții de genul:

A woman shoots a man or a man shoots a woman.

GRAMATICI CONTEXT FREE IN PROLOG

Aplicatie1 Mai jos este prezentat un program Prolog care implementeaza o gramatica context-free. In limbajele Prolog acest tip de program este foarte usor de scris.

Vom implementa gramatica context-free prezentata mai sus in SWI-Prolog.

```
propozitie (P):- append(S,V,P),grup_substantival(S), grup_verbal(V).
grup_substantival([S]):-substantiv(S).
grup_substantival([A,S]):-articol(A), substantiv(S).
grup_verbal([V]):-verb(V).
grup_verbal([V|S]):-verb(V),grup_substantival(S).

articol(un).
articol(o).
substantiv(fata).
substantiv(baiat).
substantiv(carte).
verb(citeste).
```

Iata cum vom interoga acest program:

```
?- propozitie([o,fata,citeste,o,carte]).
Yes
?- propozitie([o,fata,citeste,citeste]).
No
```

PARSING IN PROLOG

Procesul de parsing consta in analiza unui stream de intrare. Se pot usor implementa parsere folosind diferentele de liste, in cazul acesta elementele listei reprezentand (dupa caz) token-urile, cuvintele sau caracterele din streamul care se parseaza. Astfel, se poate defini

`parse(+ L`$_{input}$`, ?L`$_{output}$`)`

unde `L`$_{input}$ este stream-ul de intrare iar `L`$_{output}$ este lista diferenta dintre `L`$_{input}$ si lista elemetelor care au fost parsate.

De exemplu, parsarea unei propozitii folosind diferente de liste se poate face de maniera urmatoare:

```
sentence(Lin, Lout) :- noun_phrase(Lin, Lout1),
                       verb_phrase(Lout1, Lout).
```

Pentru ca o propozitie sa fie corecta conform gramaticii folosite la parsing trebuie ca `L`$_{out}$ sa fie lista vida, adica parsingul sa prelucreze toate cuvintele (elementele) din `L`$_{in}$.

AUTOMATE FINITE DETERMINISTE

Un ***automat determinist*** este un sistem de forma $A=(S,\Sigma,\delta,s_0,F)$ unde:

- S este o mulțime **nevidă –** mulțimea stărilor;
- Σ este o mulțime **nevidă** și **finită** – alfabet de intrare;
- δ este o funcție de tranziție $\delta: S \times \Sigma \rightarrow S$
- $s_0 \in S$ – stare inițială
- $F \subseteq S$ – mulțimea stărilor finale

Dacă S este finită, automatul se numește finit.

Limbajul acceptat de un automat $A=(S,\Sigma,\delta,s_0,F)$ este definit prin

$$L(A)=\{\omega \mid \omega \in \Sigma^*, \delta(s_0, \omega) \in F\}$$

AUTOMATE FINITE DETERMINISTE. EXEMPLU

Considerăm **automatul finit determinist**

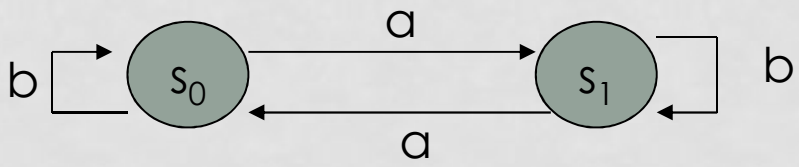
A=({a,b} ,{ s_0,s_1} ,s_0,δ ,{s_1}), unde δ se defineşte prin următorul tabel:

δ	a	b
s_0	s_1	s_0
s_1	s_0	s_1

Avem:

δ (s_0,abb) = δ (δ (s_0,a), bb) = δ (s_1,bb) = δ(δ(s_1, b),b)= δ(s_1, b)= s_1

Reprezentarea grafică a automatului:



AUTOMATE FINITE DETERMINISTE. PROPRIETATI

Lema Fie A = (Σ , S, s_0, δ ,F) un automat finit determinist. Fiind dată starea **δ(t,w)=s** cu w=x_1 … x_n, unde $x_i \in S$, 1≤ i ≤ n, există stările **s_1,s_2, … ,s_{n+1}** astfel încât s_1= t, s_{n+1}=s şi δ(s_i,x_i)= s_{i+1}, oricare ar fi i, 1 ≤ i ≤ n.

Demonstrație. Într-adevăr, notând s_{i+1}= δ(t,x_1…x_i), 1 ≤ i ≤ n, şi s_1=t avem s_{n+1}= δ(t,x_1…$x_{n-1}x_n$)= δ(δ(t,x_1…x_{n-1}), x_n)= δ(s_n, x_n) şi s_{n+1}= s.

Lema Fie A = (Σ, S, s_0, δ , F) un automat finit determinist. Fiind date stările **s_1,s_2,…,s_{n+1}** astfel încât δ(s_i,x_i)=s_{i+1}, oricare ar fi i, 1≤ i ≤ n, atunci **δ(s_1,w)=s_{n+1}** cu w=x_1…x_n, unde $x_i \in S$, 1≤ i ≤ n.

Demonstrație. Vom arăta, prin inducție completă în raport cu i, că δ(s_i,x_1…x_i)=s_{i+1} oricare ar fi i, 1≤ i ≤ n.

Într-adevăr, pentru i=1 avem δ(s_1,x_1)=s_2, ceea ce este adevărat. Presupunând că proprietatea este adevărată pentru *n*, o vom demonstra pentru *n*+1, anume s_{n+1} = δ(s_n,x_{n+1})= δ(δ(s_1,x_1…x_n), x_{n+1}) = δ(s_n, x_{n+1})= s_{n+2} [având δ(s_1,x_1…x_n)= δ(s_1,w) = s_n]

TIPURI DE AUTOMATE

Spunem că un **automat** este **determinist**, dacă din orice stare internă, primind un simbol al alfabetului de intrare, **automatul** trece într-o stare unic determinată.

Spunem că un **automat** este **nedeterminist**, dacă dintr-o stare internă oarecare, primind un simbol al alfabetului de intrare, **automatul** poate trece în nici o stare, într-o singură stare sau în mai multe stări. Trecerea dintr-o stare internă în alta nu este asociată cu probabilități.

Automatul pentru care tranzițiile (trecerile) între stări sunt caracterizate de probabilități se numeşte **automat probabilist**.

Teoremă *Limbajul reprezentabil într-un automat finit determinist este reprezentabil într-un automat finit nedeterminist.*

Teoremă *Limbajul reprezentabil într-un automat finit nedeterminist este reprezentabil într-un automat finit determinist*.

AUTOMATE FINITE NEDETERMINISTE

Un ***automat nedeterminist*** este un 5-uplu $A=(S,\Sigma,\delta,s_0,F)$ unde:

- S este o multime **nevidă –** mulțimea stărilor;
- Σ este o mulțime **nevidă** și **finită** – alfabet de intrare;
- δ este o funcție de tranziție $\delta: S \times \Sigma \rightarrow P(S)$
- $s_0 \in S$ – stare inițială
- $F \subseteq S$ – mulțimea stărilor finale

Dacă S este finită, automatul se numește finit.

Limbajul acceptat de către un automat finit nedeterminist $A=(S, \Sigma, \delta, s_0, F)$ este

$$L(A)=\{p \mid p \in \Sigma^*, \delta(s_0,p) \cap F \neq \Phi\}$$

Deci un cuvânt va aparține limbajului L(A) dacă automatul, pornind din starea inițială, are posibilitatea să ajungă în cel puțin o stare finală.

O definiție mai restrictivă a limbajului acceptat de un automat finit nedeterminist

$$L(A)=\{p \mid p \in \Sigma^*, \delta(s_0,p) \subseteq F\}$$

DE LA O GRAMATICĂ FINITĂ LA O DIAGRAMA DE TRANZIȚIE

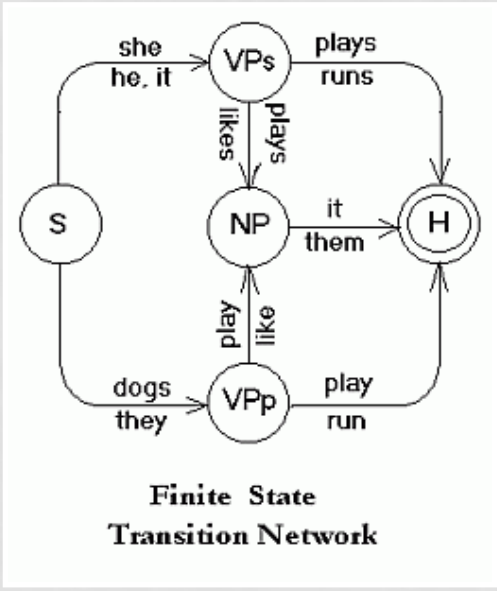
Considerăm următoarea gramatică finită *G* cu simbolurile neterminale și terminale

- V_N = {S, VPs, VPp, NP, H}
- V_T = {she, he, it, they, them, dogs, run, like, play, plays, likes, runs}

și cu mulțimea de producții constând din următoarele reguli:

P = {**S** → she **VPs**, **S** → he **VPs**, **S** → it **VPs**,
S → dogs **VPp**, **S** → they **VPp**,
VPs → likes **NP**, **VPs** → plays **NP**,
VPs → plays **H**, **VPs** → runs **H**,
VPp → like **NP**, **VPp** → play **NP**,
VPp → play **H**,**VPp** → run **H**,
NP → it **H**, **NP** → them **H**,
H → λ }

Producțiile unei gramatici context-free pot fi uşor transpuse într-o diagrama de tranziție.



Finite State Transition Network

TRADUCATOARE FINITE

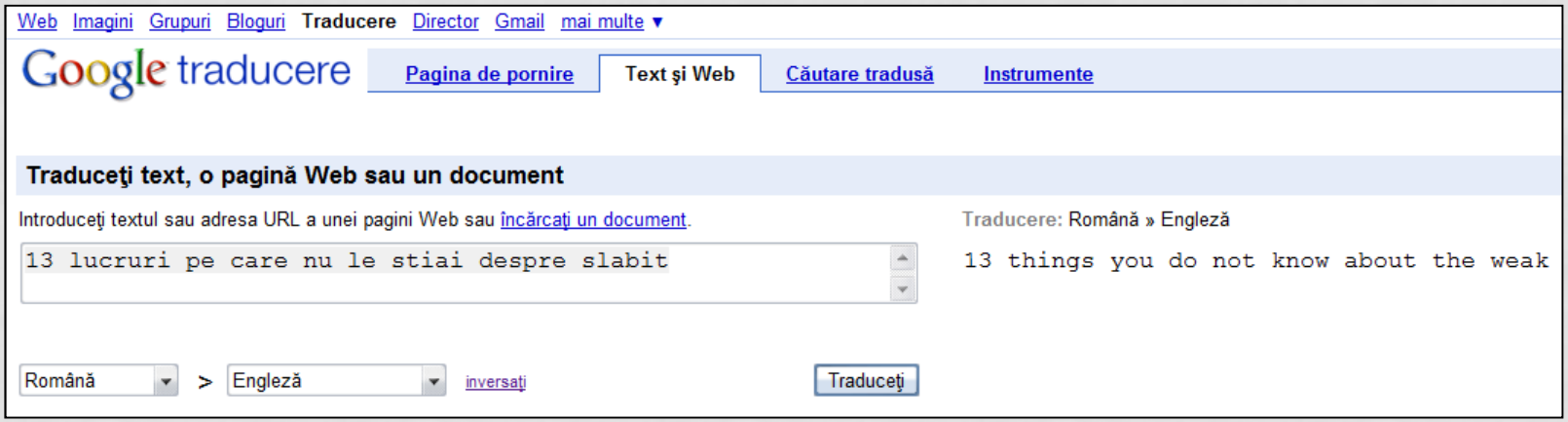
În procesarea limbajului natural putem folosi și traducătoarele finite. Traducătoarele finite sunt automate finite înzestrate cu ieșiri. Un traducător finit pornind dintr-o stare dată trece în altă stare și emite o secvență pe un alfabet de iesire când primește un simbol de intrare.

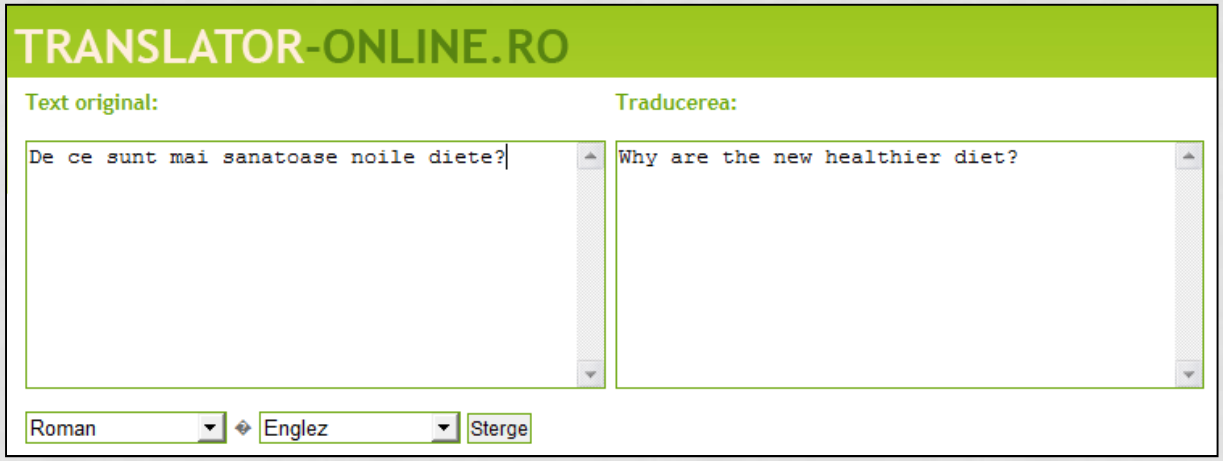
Definitie Un traducător finit este un 6-tuplu $T=(S, \Sigma, \Delta, \delta, s_0, F)$ unde:

- S este o mulțime finită de stări interne
- Σ este alfabetul de intrare
- Δ este alfabetul de ieșire
- $\delta: S \times \Sigma \rightarrow S \times \Delta$ este funcția de tranziție
- $s_0 \in S$ este starea inițială a traducătorului
- $F \subset S$ este mulțimea stărilor finale

Funcția de tranziție δ o definim astfel: $\delta(s, a) = (r, z)$ ceea ce se citește în felul următor: dacă traducătorul T se află în starea *s* și primește la intrare simbolul *a* atunci va trece în starea *r* și va emite la ieșire simbolul *z*.

TRADUCATOARE. EXEMPLE





TRADUCATOARE FINITE. EXEMPLU

Traducător finit care emite forma de plural a expresiei care a primit-o la intrare.

- Pentru numerale avem următoarele rescrieri:
  "un" se rescrie cu "doi" iar
  "o" se rescrie cu "doua".

- Trecerea de la singular la plural pentru substantive și adjective se face în funcție de gen.

TRADUCATOARE FINITE. EXEMPLU

toate substantivele și adjectivele masculine la plural se termină în "i"

- dacă subst. se termină cu o consoană atunci se va adauga "i"
- dacă subst. se termină în "iu" atunci se va inlocui "iu"-l final cu "ii". Exemplu: fiu-fii, vizitiu-vizitii
- dacă subst. se termină cu o vocală atunci se va înlocui vocala cu "i". Exemplu: codru-codri, metru-metri

substantivele și adjectivele feminine nu au o terminație comună la plural. Am considerat numai câteva cazuri, trecerea de la singular la plural în acest caz fiind greu de implementat, deoarece există multe excepții de la regulă:

- dacă subst. se termină în "a" atunci "a"-ul final se va inlocui cu "e". Exemplu: bomboana-bomboane. Excepție:fata-fete, masa-mese.
- dacă subst. se termină în "ie" atunci se va inlocui "ie" cu "ii". Exemplu: familie-familii.
- dacă subst. se termină în "e" atunci "e"-ul final se va inlocui cu "i". Exemplu: floare-flori, carte-carti.

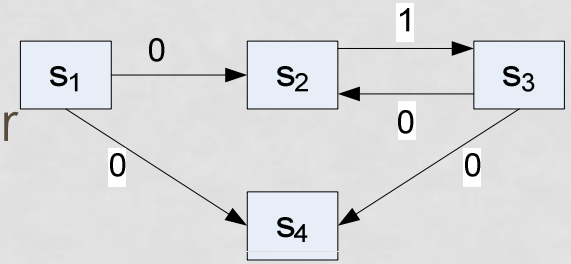
TRADUCATOARE FINITE. EXEMPLU

?- traduce([o,carte,frumoasa],L).
  carte este substantiv masculin?(d/n) n.
  frumoasa este adjectiv masculin?(d/n) n.
  L=[doua,carti,frumoase]
Yes
?- traduce([un,baiat,mare],L).
  baiat este substantiv masculin?(d/n)d.
  mare este adjectiv masculin?(d/n)d.
  L = [doi, baiati, mari]
Yes

RETELE DE TRANZITIE DE BAZA

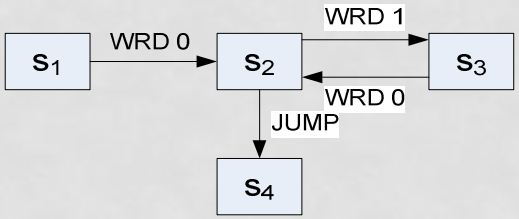
Simulează comportamentul unui automat finit.

Puterea generativă a rețelelor de tranziție de bază este aceeași cu puterea automatelor finite sau, echivalent, cu puterea gramaticilor regulare.



Automat finit

Exemplu: În ambele cazuri limbajul acceptat este $\{0(10)^n\}_{n \geq 0}$ pentru s_1 – starea inițială iar s_4 - starea finală



Retea de tranzitie de baza

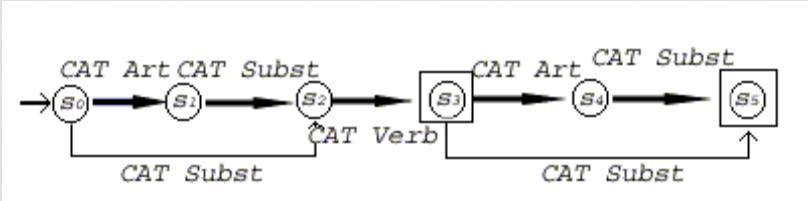
În cazul RTB putem avea următoarele etichete pe arce:

- **JUMP**: salt necondiționat la o altă stare
- **CAT xyz**: tranziția trebuie să consume o entitate aparținând categoriei sintactice xyz
- **WRD w**: tranziția va consuma cuvantul w

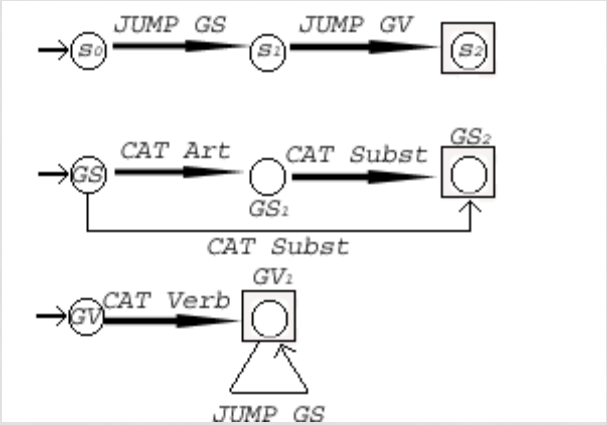
RETELE DE TRANZITIE RECURSIVE

O rețea de tranziție recursivă (RTR) se obține dintr-o rețea de tranziție de bază (RTB) dacă dăm posibilitatea ca cel puțin un arc al acesteia să fie etichetat cu comanda **PUSH nod**.

RTR = RTB + (PUSH nod)



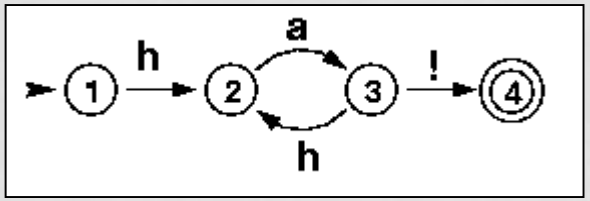
Rețea de tranziție de bază



Rețea de tranziție recursivă

INPLEMENTAREA IN PROLOG A UNEI RETELE DE TRANZITIE RECURSIVE

```
Yes
8 ?- test(X).
Correct to: test1(X)? yes

X = [h, a, !] ;

X = [h, a, h, a, !] ;

X = [h, a, h, a, h, a, !] ;

X = [h, a, h, a, h, a, h, a, !] ;

X = [h, a, h, a, h, a, h, a, h|...]

X = [h, a, h, a, h, a, h, a, h|...]
% Waiting for editor ...
```



1

- initial/1
- final/1
- arc/3

```
initial(1).
final(4).
arc(1,2,h).
arc(2,3,a).
arc(3,4,!).
arc(3,2,a).
```

3

```
test1(Symbols) :-
    initial(Node),
    recognize1(Node,Symbols).
```

2

```
recognize1(Node,[]) :-
    final(Node).
recognize1(Node1,String) :-
    arc(Node1,Node2,Label),
    traverse1(Label,String,NewString),
    recognize1(Node2,NewString).

traverse1(Label,[Label|Symbols],Symbols).
```

- Vă mulțumesc!