N.L.P. - C 4

Bottom-Up:

operația principală: deplasare - reducere.

ex.: Un câine latră.

↳ Gramatica din Cursul 3.

Caract. ale implem. în Prolog:

1. Construcția stivei se face inapoi: introduc cuvintele în stivă de la început la sfârșit.

2. Regulile gramat. se introduc inapoi: ex.: NP → Art. N.

regula ([n, art | X], [np | X]).

ex.: Un câine latră.

	Acțiune	Stiva	Șir intrare
1.	START	[ ]	[un, câine, latră]
2.	Deplasare	[un]	[câine, latră].
3	Reducere ↳ se inlocuiește cuvântul din stivă cu partea de vorbire	[art]	[câine, latră].
4.	Deplasare	[câine, art]	[latră].
5.	Reducere	[n, art]	[latră]
6.	Reducere.	[np]	[latră].
7.	Deplasare	[latră, np]	[]
8.	Reducere	[V, np]	[]
9.	Reducere	[vp, np]	[]
10	Reducere	[s]	[]

la începutul stivei

1/6

Pași alg:

P1: Se deplasează un cuvânt în stivă

P2: Se reduce stiva în mod repetat utilizând regulile și categoriile lexicale ale gramaticii până această op. nu mai e posibilă.

P3: Dacă nu mai ∃ cuv. în șirul de intrare, se merge la P2, altfel STOP.

! regulă ([n, art | X], [np | X]).

n, art ↳ vf. stivei căreia i se aplică regula

↳ ce devine stiva după reducere.

cuvânt (c, categorie).

?- parse ([un, câine, latră], [s]).

sau

?- parse ([un, câine, latră], [X]).

X = s;

---

implementare

↳

```
parse(S, Rez) :- dspl_red(S, [], Rez).

dspl_red(S, Stiva, Rez) :- deplasare(Stiva, S, StivaNoua, S1),
                           reducere(StivaNoua, StivaReducsa),
                           dspl_red(S1, StivaReducsa, Rez).

dspl_red([], Rez, Rez).
```

(dspl_red(S, Stiva, Rez))
- S: șirul de intrare
- Stiva: conf. curentă a stivei
- Rez: rezultatul

S1: ce a rămas în S

↳ caz particular.

```
deplasare(X, [H|Y], [H|X], Y).

reducere(Stiva, StivaRed) :- regula(Stiva, Stiva1),
                             reducere(Stiva1, StivaRed).

reducere(Stiva, Stiva).
```

Anca was HERE! ^_^.

Limitare:

$A \rightarrow \emptyset$

ex.:

NP → Det · N

Det → ∅ (un briat ↓ ∅ ☹) / ☺ briat.

Combinație Bottom-Up – Top-Down

Alg: (Analizează un constituent C).

P1: Se acceptă un cuvânt din șirul de intrare și i se determină categoria.

Fie această categorie W.

P2: Se completează C astfel:

Dacă W == C, STOP

Altfel:

P.2.1: Cu ajutorul regulilor gram. se găsește un constituent a cărui dezvoltare începe cu W.

Fie P. acest constituent.

ex: W ≡ Det

NP → Det . N

(W → Det, P → NP)

P2.2: Se analizează în mod recursiv și utilizând strategia top-down, toate elem. rămase ale dezvoltării lui P.

P2.3: Se înlocuiește W cu P și se merge înapoi la P2. pt. a continua completarea lui C.

Implementare Prolog:

parse (C , S1 , S)

- C: categoria căutată
- S1: șirul de intrare
- S: șirul ieșire.

parse (C , [Cuvânt | S2] , S) :- cuvânt (W , Cuvânt),

W: categoria

completează (W , C , S2 , S).

completează verifică dacă W poate fi primul constituent al lui C, folosind S_2 și ajungând la S.

completează (C , C , S , S).

completează (W , C , S1 , S) :- regula (P , [W | Rest]),

parse_lista (Rest , S1 , S2),

completează (P , C , S2 , S).

def. în Cursul 3.

Def.: Harta = o listă care ține minte ce a fost analizat și funcționează.

Tabel de subșiruri bine formate (TSBF).

Poziții de la 0 la n.

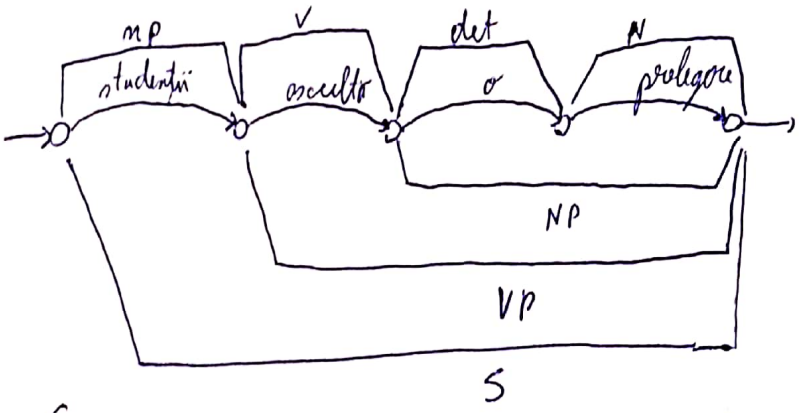
perechi de poziții (i, j) o c i c $j \leq n$

↳ reține constituenții dintre i și j

<START> = întreg (poziție)

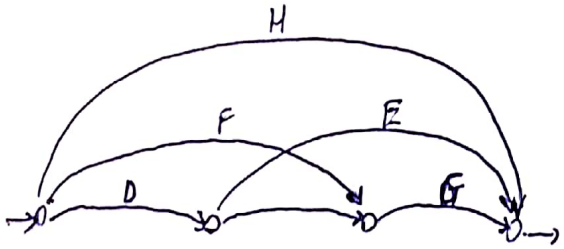
<FINAL> = întreg (poziție)

<ETICHETĂ> = categorie



↳ Arată ordinea constituenților, dar nu arată regula gramaticii

ex.:



↓

NU putem spune dacă H vine din D·E sau F·G

H → DE

H → FG