

Consiglio Nazionale delle Ricerche  
ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

SYNREC - L'ANALIZZATORE SINTATTICO  
PER L'INTERPRETATORE PANON-1B.

L. Spanedda

Nota Interna B68-22

# SYNREC : L'ANALIZZATORE SINTATTICO PER L'INTERPRETAZIONE

PANON-13

L. Spanella (s)

## 1. Introduzione

Il programma SYNREC è stato studiato e realizzato nell'ambito della definizione di un interprete per un linguaggio per manipolazione di simboli basato su una estensione degli algoritmi generalizzati di Markov: il PANON-13 (Programming Language for Symbol Manipulation).

Questo linguaggio consiste essenzialmente di:

- 1) Un insieme di definizioni di classe, che definiscono un insieme locale di linguaggi liberi a struttura di frono.
- 2) Una sequenza di regole di trasformazione strutturale, che definiscono l'algoritmo.

Una regola di trasformazione strutturale è della forma

$$\varphi' \rightarrow \varphi''$$

dove  $\varphi'$  e  $\varphi''$  sono coppie di espressioni che definiscono strutture di parole su  $V_T \cup V_N$ , dove  $V_T$  è un alfabeto base e  $V_N$  un insieme di simboli non terminali che denotano classi di parole su  $V_T$ .  $\varphi'$  e  $\varphi''$  sono chiamati rispettivamente primo e secondo membro della regola di trasformazione.

L'interpretazione di un programma PANON-13 consiste nell'analisi e la successiva riduzione di un certo numero di espressioni chiamate oggetti.

Una regola di trasformazione si dice che è applicabile ad una stringa oggetto, ad un certo passo  $t$ , se e soltanto se è possibile ottenere una corrispondenza tra le configurazioni definite da  $\langle \sigma \rangle$  ed un segmento della stringa oggetto.

Il problema di accertare l'esistenza di una tale corrispondenza è, ovviamente, il problema principale che un interprete deve risolvere. Il SYNREC, che è una delle parti principali dell'interprete PANOR-1B, è un sottoprogramma che viene chiamato ogni qualvolta si debba decidere l'appartenenza di una sequenza di caratteri della stringa oggetto ad una certa classe, del tipo libero a struttura di frase.

Esso viene inserito nell'interprete PANOR-1B in base al diagramma illustrato nella fig. 1a.

Il ciclo interpretativo chiama successivamente una regola dopo l'altra, facendo esaminare il primo membro dal decompositore sintattico. Se la regola risulta applicabile si esegue, diversamente si passa ad esaminare la successiva. Il decompositore sintattico esegue una serie di partizioni particolari della stringa oggetto in  $k+2$  segmenti consecutivi (sottostringhe), dove  $k$  è il numero dei termini del primo membro, e chiama il SYNREC a decidere, per ognuna delle sottostringhe di posto uguale a quello di un nome di classe nella regola, l'appartenenza a quella classe.

Il SYNREC è quindi un analizzatore sintattico per linguaggi liberi a struttura di frase.







le S-derivazioni sono più di una, allora diremo che il linguaggio è ambiguo.

Data una grammatica  $G$ , chiamiamo ordine di un elemento  $X \in V_N$  il numero di espressioni, o alternative, associate ad  $X$  in  $G$ .

Consideriamo la produzione

$$X \rightarrow x$$

diciamo che  $X$  è un elemento nullificante di  $V$  se è stato associato alla parola vuota, ovvero

$$x = \Lambda$$

Se invece risulta  $x \neq \Lambda$ , ma esiste una derivazione  $X \Rightarrow \Lambda$ , allora diciamo che  $X$  è un elemento indirettamente nullificante.

Un elemento  $X$  si dice ciclico se esistono due parole  $x_1$  e  $x_2$  tali che

$$X \Rightarrow x_1 X x_2$$

In particolare se risulta  $x_1 = \Lambda$  e  $x_2 \neq \Lambda$  si dice ciclico a sinistra, se, invece, risulta  $x_1 \neq \Lambda$  e  $x_2 = \Lambda$ , si dice ciclico a destra.

### 3. L'Algoritmo generale

Un analizzatore sintattico è un algoritmo che, dato un linguaggio  $L(G)$  ed una stringa arbitraria  $k$ , terminale rispetto a  $G$ , decide se  $k$  è o non è una stringa per un dato elemento sintattico  $X$  di  $G$  e, in caso affermativo, fornisce una  $X$ -derivazione di  $k$ , cioè la sua struttura sintattica.

Gli algoritmi generali di analisi sintattica si distinguono in ascendenti e discendenti. Negli algoritmi ascendenti si parte dalla stringa oggetto per risalire al simbolo iniziale; negli algoritmi discendenti, invece, si parte dal simbolo iniziale per arrivare ai simboli della stringa oggetto.

Il SYNRUC è basato su un algoritmo di tipo discendente, descritto alla fine del paragrafo, che non richiede l'ordinamento della grammatica sulla quale deve operare (vedi l'esempio più oltre); cioè non richiede l'introduzione di una ben determinata gerarchia tra le produzioni. L'algoritmo presentato è del tutto generale e consente di operare su grammatiche della massima generalità senza alcuna restrizione. Infatti, non si pongono restrizioni sulle produzioni cicliche, sulle produzioni nullificanti, sul numero di produzioni con lo stesso primo membro, sulla lunghezza delle produzioni, ecc.

Consideriamo, ad esempio, la grammatica  $G_1 = (V, V_T, P, S)$ , dove:

$$V = \{S, M, W, a, b, c, \Lambda\}$$

$$V_T = \{b, a, c\}$$

$$S = S$$

$$P = \left\{ \begin{array}{l} S \rightarrow W, M \rightarrow b, S \rightarrow aMb, S \rightarrow bSW, W \rightarrow M, \\ W \rightarrow c, S \rightarrow SW, W \rightarrow aM, M \rightarrow \Lambda \end{array} \right\}$$



Le produzioni, opportunamente codificate, vengono memorizzate in due tabelle distinte chiamate TP1 e TP2 (vedi fig. 1b). La prima tabella (ad una dimensione) contiene i codici degli elementi dei secondi membri delle produzioni. La seconda tabella (a quattro dimensioni) su una stessa linea e per ogni produzione, contiene le seguenti informazioni:

- il codice del nome di classe (primo membro della produzione);
- l'ordine del nome di classe;
- la lunghezza della produzione;
- la posizione nella tabella TP1 del primo elemento dell'espressione assegnata al nome di classe.

Le due tabelle rappresentano la grammatica del linguaggio. La stringa da esaminare, anch'essa opportunamente codificata, viene memorizzata in una tabella ad una dimensione, che indichiamo con TS. Se consideriamo una stringa di n elementi, allora TS(1) contiene il codice del primo elemento, TS(2) il codice del secondo, ed, infine TS(n) il codice dell'ultimo elemento della stringa.

Il SYNREC utilizza per la memorizzazione dei risultati parziali due stack, chiamati rispettivamente, stack operativo (nel seguito verrà indicato con SPO) e stack sintattico (STS). Nel primo le informazioni vengono memorizzate da destra verso sinistra, nel secondo, viceversa, da sinistra verso destra. Si ha così il vantaggio di sfruttare convenientemente la zona di memoria disponibile. Ogni qualvolta l'analisi sintattica ha esito positivo, STS viene a contenere la struttura sintattica della stringa e, conseguentemente, SPO risulta vuoto.



Indichiamo con  $K$  la posizione del simbolo della stringa, di lunghezza  $LS$ , che si sta esaminando ad un istante  $t$ ; con  $J$  la posizione che un elemento  $X$  occupa nello stack operativo; con  $P$  il numero di elementi non nullificanti alla destra di  $X$  in STO, simboli terminali compresi.

Se un elemento  $X$  viene trasferito dallo stack operativo nello stack sintattico, ad esso viene associata una quadrupla di informazioni che consentono, in caso di insuccesso parziale, di esaminare le altre alternative possibili. Questa quadrupla caratterizza l'elemento  $X$  in STS, ed è indicata con  $X(N, J, K, P)$ , dove  $N$  rappresenta l'ordine dell'elemento, cioè l'alternativa considerata nella costruzione dell'albero sintattico, e  $J, K, P$  sono stati precedentemente definiti. Se  $X \in V_N^*$ , si ha  $N = 0$ .

Def. 1 - Un elemento  $X$  sulla testa dello stack operativo, si dice accettabile se, essendo verificata la relazione  $LS+1 \geq K+P$ , si ha

$$STO(J) = TS(K) \text{ , oppure } STO(J) \in V_N^*$$

Def. 2 - Un elemento ciclico  $X(N, J, K, P)$  sulla testa dello stack sintattico si dice accettabile per ogni  $X'(N', J', K', P')$  se è verificata almeno una delle seguenti condizioni:

- a)  $X \neq X'$  ;
- b)  $X = X'$  , ma  $K \neq K'$  ;
- c)  $X = X'$  ,  $K = K'$  , ma  $P \neq P'$  ;
- d)  $X = X'$  ,  $K = K'$  ,  $P = P'$  , ma  $J < J'$  ;
- e) il numero degli elementi per cui è  $X = X'$  ,  $K = K'$  ,  $P = P'$  ,  $J \geq J'$  , è minore del numero  $(LS-K)$  dei caratteri della stringa ancora da esaminare.

Il diagramma dinamico semplificato del SYNREC è dato in fig. 2.

I vari blocchi del diagramma eseguono le seguenti operazioni:

- 1: Memorizza nello stack operativo il simbolo iniziale.
- 2: Trasferisce l'elemento di testa dello stack operativo nello stack sintattico; se tale elemento è ciclico va a 7, altrimenti continua su 3.
- 3: Trasferisce nello stack operativo l'espressione associata all'elemento di testa dello stack sintattico.
- 4: Si distinguono i seguenti due casi fondamentali:
  - a) il primo elemento dello stack operativo è accettabile; se non è terminale si va a 2, altrimenti esso viene trasferito nello stack sintattico. In questo caso le eventualità sono due:
    - a.1) l'analisi è terminata ed ha dato esito positivo;
    - a.2) l'analisi non è terminata; si ripete a).
  - b) il primo elemento dello stack operativo non è accettabile; si procede su 5.
- 5: Esamina l'esistenza di una eventuale successiva espressione assegnata all'elemento di testa dello stack sintattico; se la trova va a 3, altrimenti continua su 6.
- 6: Ritrasferisce l'elemento di testa dello stack sintattico in quello operativo; se SPS risulta vuoto, l'analisi ha dato esito negativo; altrimenti torna a 5.
- 7: Se l'elemento di testa dello stack sintattico è accettabile, si va a 3, altrimenti si va a 6.

#### 4. Schema generale del funzionamento del SYNFEC

##### Lista dei simboli

##### 1 - Cella di memoria con funzioni speciali

K = numeratore del simbolo in esame nella stringa

J = " dello stack operativo

I = " dello stack sintattico

P = cella contenente il numero di elementi non nullificanti in STO

NC = cella contenente il simbolo iniziale

LS = " " la lunghezza della stringa

X, T, W, L, N, S = celle ausiliarie

##### 2 - $Z(R)$ = contenuto della cella R (con R qualunque).

$Z(T)$  = cella avente indirizzo relativo T, dove Z è un nome di zona di lavoro (ad es.: STS), e T è una cella con funzioni speciali.

$Z(T, n)$  = n.<sup>mo</sup> "campo" della cella  $Z(T)$ , se per la cella Z sono previste delle partizioni in "campi".

$Z(T, n_1, n_2)$  = "campo" della cella  $Z(T)$  risultante dalla unione dei campi  $n_1, n_2$ .

3 -  $\{V\}$  = insieme dei simboli nullificanti

$\{V_G\}$  = " " " ciclici

$\{V_T\}$  = " " " terminali

$\{P\}$  = " delle produzioni



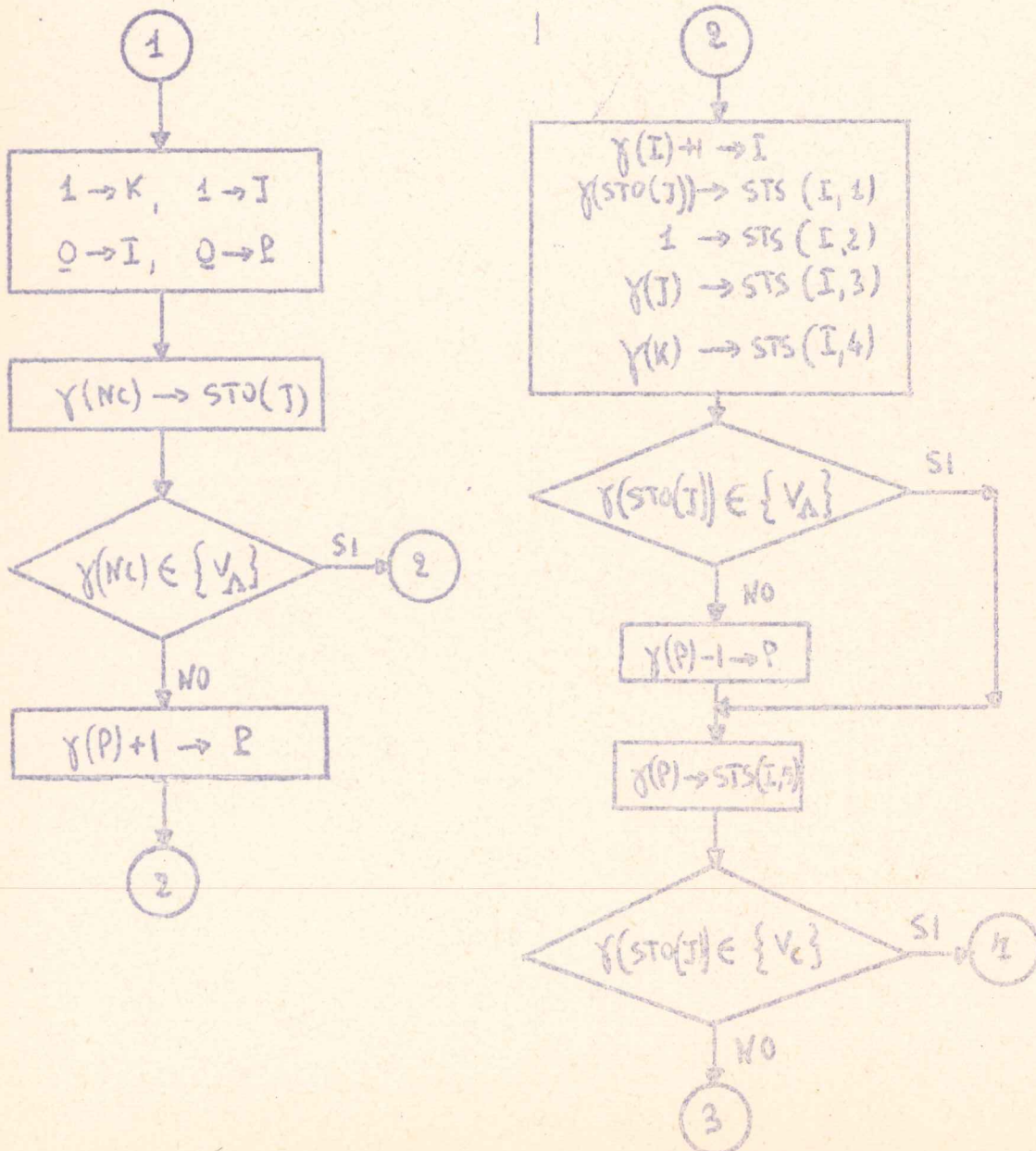
4 - STO = stack operativo

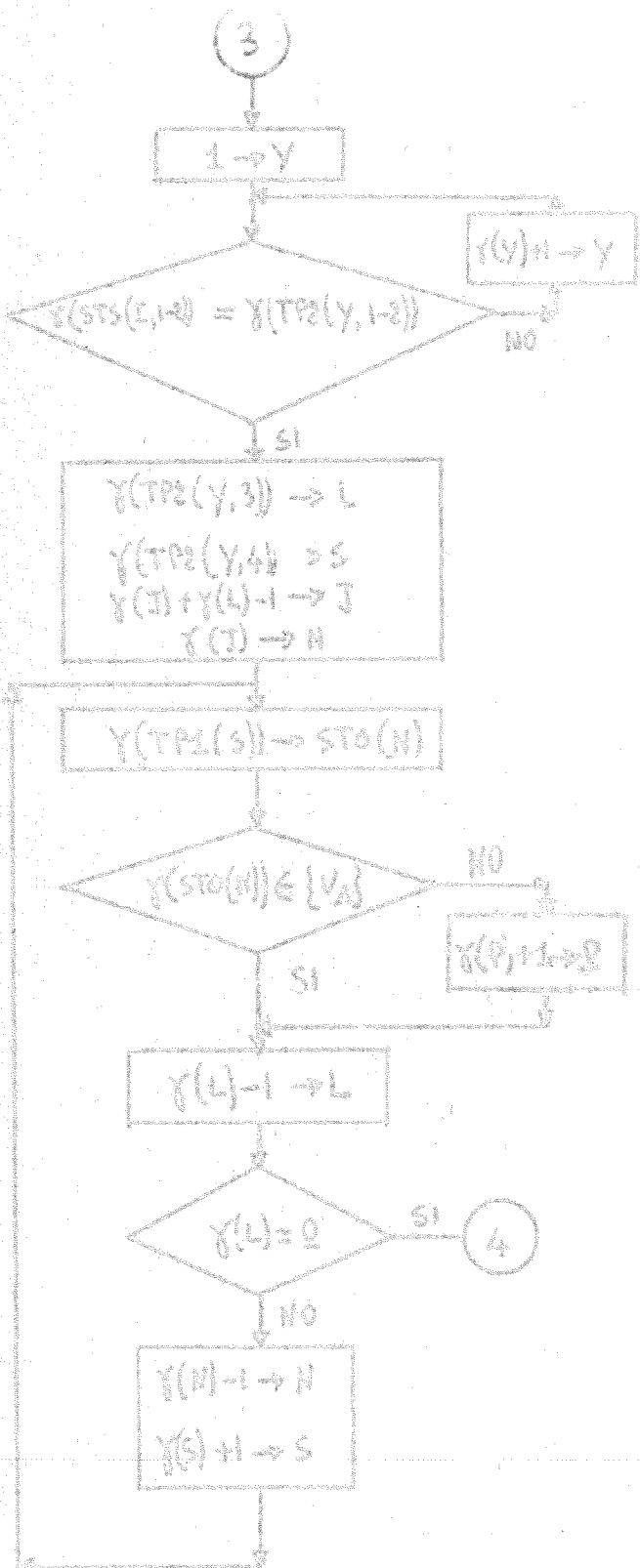
STS = stack sintattico

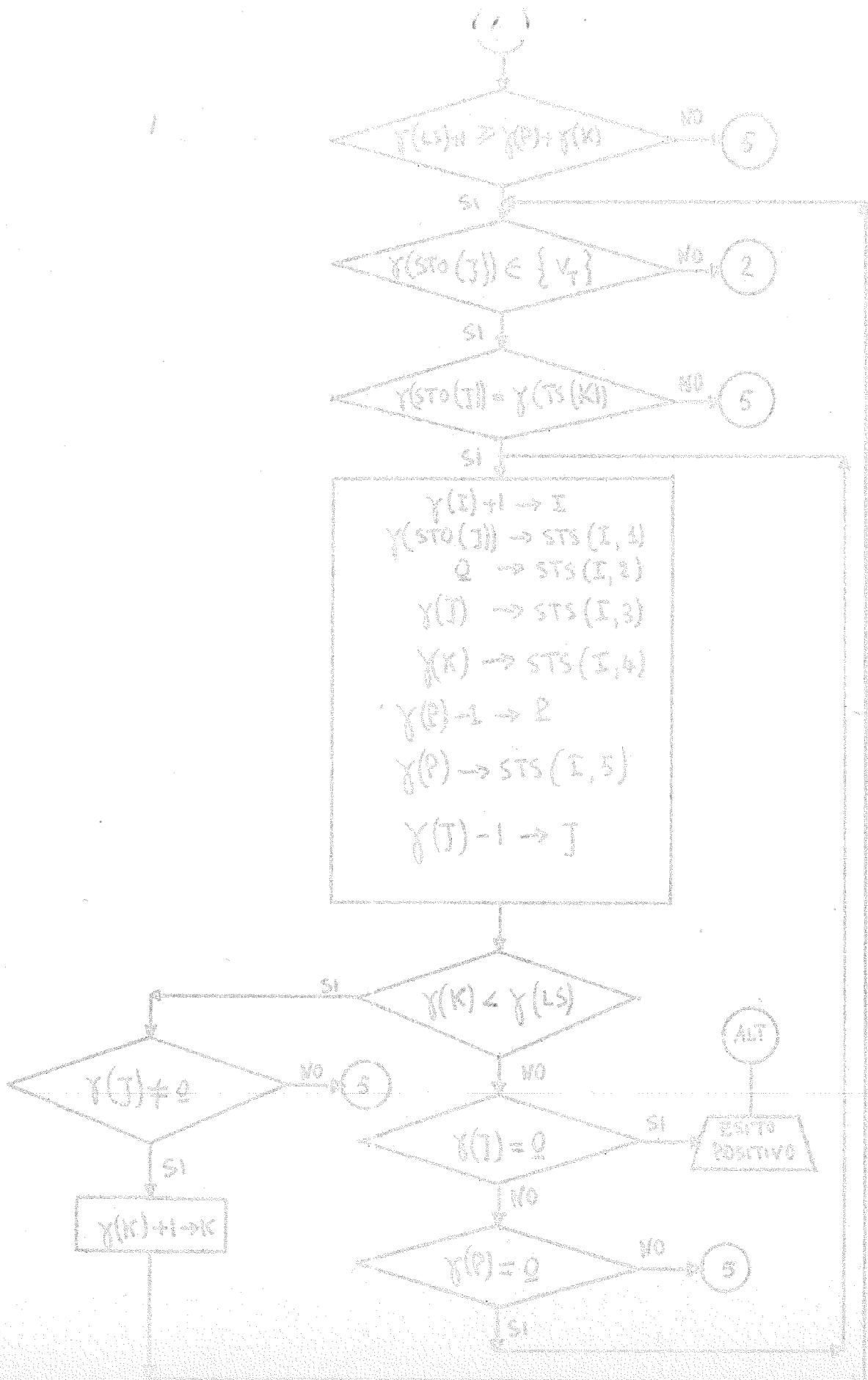
TP1 = tabella dei codici degli elementi che compaiono nei secondi membri delle produzioni.

TP2 = tabella dei codici dei nomi di classe

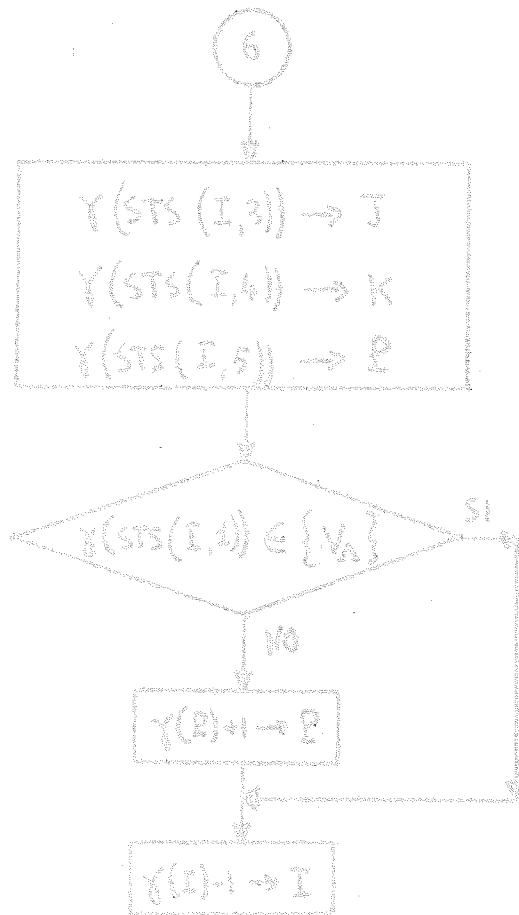
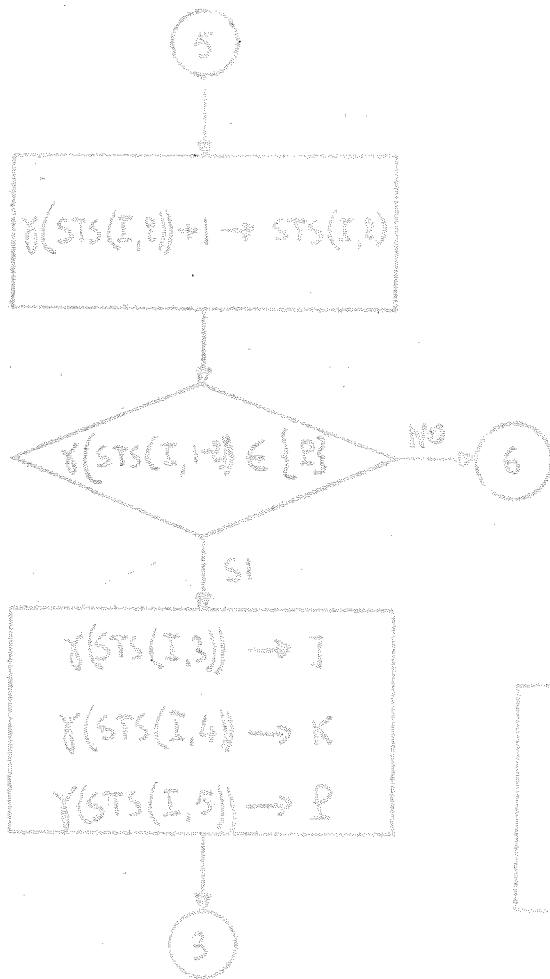
Dettaglio dei blocchi del diagramma semplificato.

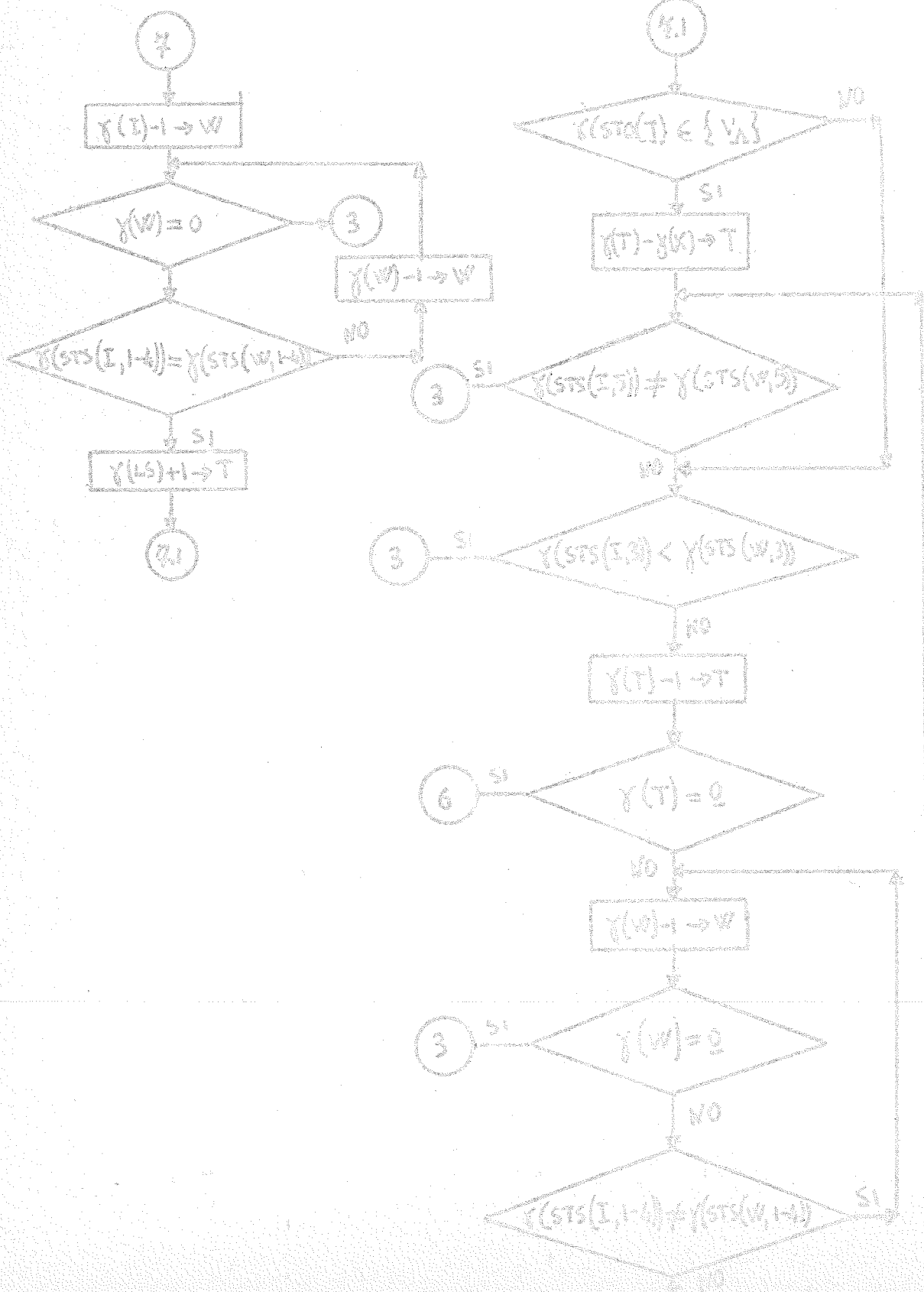












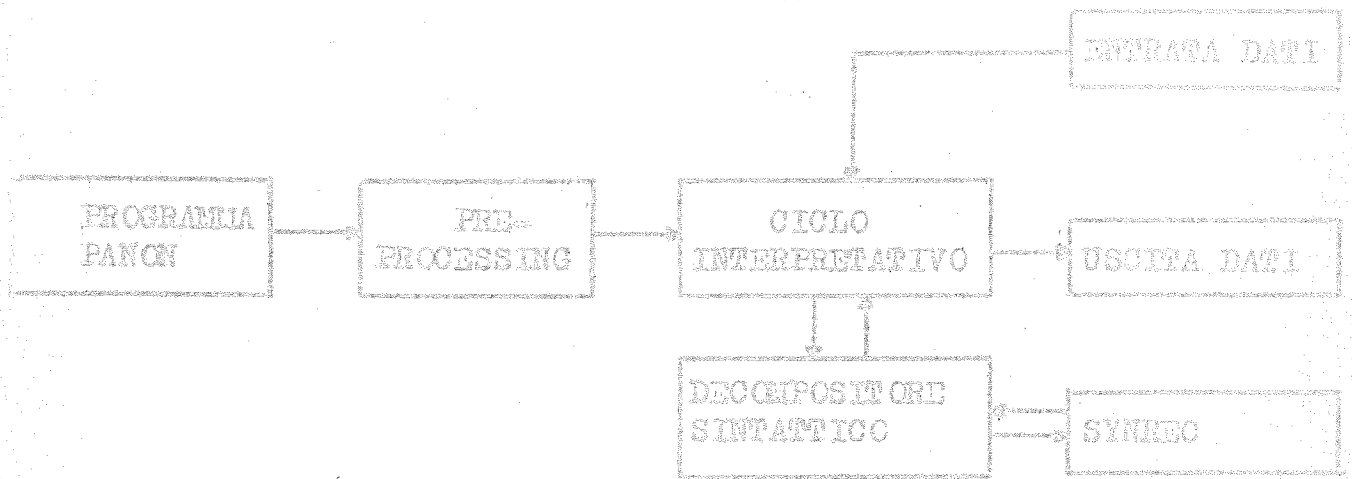


Fig. 1a - Diagramma dell'interprete PANON-13.

TP1

V	b	a	W	c	S	S	W	W	c	S	V	a	W
---	---	---	---	---	---	---	---	---	---	---	---	---	---

TP2

S	1	1	1
W	1	1	2
S	2	3	3
S	3	3	6
W	1	1	9
V	2	1	10
S	4	2	11
W	3	2	12
V	4	0	-

$$G_1 = (V, V_2, P, S)$$

$$V = \{S, W, V, a, b, c, \Delta\}$$

$$V_2 = \{b, a, c\}$$

$$S = S$$

$$P = \left\{ \begin{array}{l} S \rightarrow W, W \rightarrow b, S \rightarrow aW, \\ S \rightarrow bSW, W \rightarrow W, W \rightarrow c, \\ S \rightarrow SW, W \rightarrow aW, W \rightarrow \Delta \end{array} \right\}$$

Fig. 1b - la grammatica  $G_1$ , e le tabelle corrispondenti



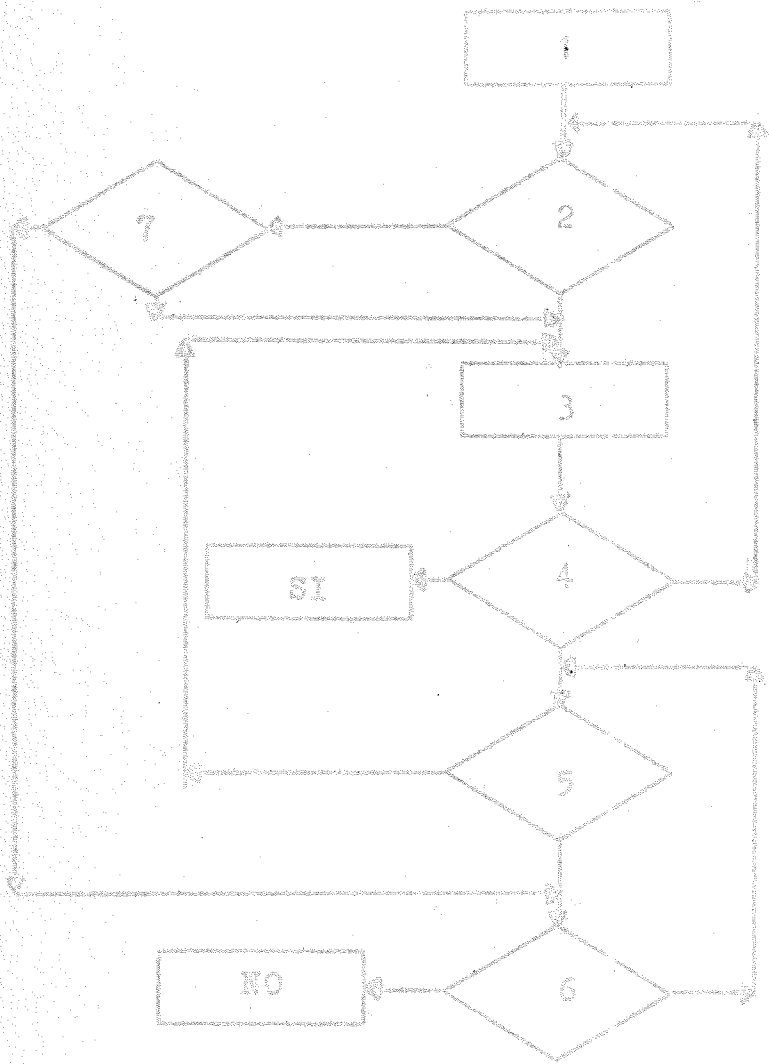


Fig. 2 - SYNREC : diagramma dinamico semplificato.

## BIBLIOGRAFIA

1. A. Caracciolo di Forino - Elementi di teoria formale dei linguaggi. Appunti al corso di Cibernetica presso la Facoltà di Scienze, Università di Pisa, 1964-65. Felici, Pisa, (1968).
2. N. Wolkenstein - Some remarks on the implementation of PANON-1B. SIC-SAM Bulletin 7, (July 1967).
3. J. Eickel, M. Paul, F.-L. Esuer, K. Samelson - A Syntax-controlled generator of formal language processors. Comm. ACM, 6 (1965), 451-455.
4. T.V. Griffiths, S.R. Petrick - On the efficiencies of context-free grammar recognizers. Comm. ACM, 8 (5) (1965).
5. A. Caracciolo di Forino, L. Spanedda, N. Wolkenstein - PANON-1B: A programming language for symbol manipulation. Calcolo, 3 (2) (1966), 245-265.
6. L. Spanedda, N. Wolkenstein - An interpreter for PANON-1B (in preparazione).
7. L. Bolliet - L'écriture des compilateurs. Chiffres, 9 (1) (1966), 47-73.
8. S. Ginsburg - The mathematical theory of context-free languages. New York, McGraw-Hill, (1966).
9. R.W. Floyd - The syntax of programming languages - A survey. IEEE Transactions on Electronic Computers, (1964), 346-353.

10. P.S. Landweber - Decision problems of phrase-structure grammars.  
IEEE Transactions on Electronic Computers, (1964),  
354-362.
11. A. Caracciolo di Porino, L. Spanedda - Analisi multiple per  
linguaggi Context-Free e natura dell'iniezione delle  
strutture sintattiche.  
(in preparazione).
12. H. Brassieur, J. Cohen - Algorithmes d'analyse syntaxique  
pour langages "context-free".  
Chiffres, 8 (2), (1965), 95-119.