VRAAGSTUKKEN OMTRENT DE REPRESENTATIE VAN LINGUISTISCHE INFORMATIE

Luc Steels

UNIVERSITEIT ANTWERPEN (U.I.A.)
Dept. Germ. Fil., Afd. Nederlandse Taalkunde
Technisch Rapport. LS/2.
december 1974

0. Probleemstelling.

Deze paper handelt over metoden om linguïstische informatie (vb. syntaktische analyses, semantische representaties) te representeren.

We zijn op dit probleem gestoten bij de programmering van linguïstische experimenten. Programmering vereist immers naast een volledige explicitering van het proces dat men wil simuleren ook een nauw gezette definiëring van de datastruktuur (de representatie, het 'format') van input en output.

Al kunnen we stellen dat er voor de syntaxis een algemene representatie in de linguïstiek gangbaar is (namelijk labelled bracketings of boomdiagrammen), voor hogere taalniveaus is deze eensgezindheid zoek geraakt. En zelfs al is er eensgezindheid, dan nog blijft het probleem wat de door een computerverwerkbare ekwivalenten zijn binnen de mogelijkheden van de programmeertaal die wordt gebruikt.

Wat ons in deze paper vooral zal interesseren is de output, dit wil zeggen wat er uit de syntaktische en semantische komponenten voortkomt. We zullen ons hier niet bezig houden met vraagstukken omtrent de input voor deze komponenten zoals de representatie van een grammatika, de organisatie en het ingeven van een lexikon, de notatie van semantische primitieven. Deze problemen zijn meer systeem gebonden en werden op andere plaatsen behandeld. (Zie bijv. Steels, 1974, b)

In deze paper zullen we ook zelf een voorstel doen om linguïstische informatie te representeren voornamelijk met het oog op automatische verwerking ervan. Ook zullen we expliciet ingaan op de manier waarop er met deze datastruktuur kan gewerkt worden door computers.
Zo zullen programma's voor automatische syntaktische analyse, het automatisch genereren van zinnen via een gegeven grammatika het verkrijgen van een output vanuit een matrixrepresentatie van boomdiagrammen, het uitvoeren van transformaties op gegeven syntaktische strukturen, geïmplementeerd worden.

De tekst kent twee onderdelen. In het eerste deel bespreken we de representaties die momenteel gangbaar zijn in de linguïstiek en stellen dan een alternatief voor.
In het tweede deel bespreken we hoe bomen normaal kunnen voorgesteld worden in computers en welke programmeertaal hiervoor het meest geschikt is. Verder gaan we in op een metode om bij het ontbreken van deze programmeertaal toch over de nodige faciliteiten te kunnen beschikken. Tenslotte geven we praktische toepassingen voor de programmeertaal BASIC met de voorgestelde representatie.

I. 1. Bestaande representaties voor linguïstische informatie.

1. Representaties voor syntaxis.

Bij generatieve grammatika's wordt een onderscheid gemaakt tussen de zwakke generatieve kapaciteit en de sterke generatieve kapaciteit (Chomsky, 1963). De zwakke generatieve kapaciteit slaat op het empirisch bereik, dwz de taal (de verzameling zinnen) die door de grammatika wordt gegenereerd.
Met de sterke generatieve kapaciteit bedoelt men de stukturele deskripties door de grammatika gegenereerd voor de zinnen uit het bereik.
Deze strukturele deskripties worden genoteerd in de vorm van bommidagrammen of van strings met labelled bracketings.
We zullen nu even deze twee representaties definiëren.
(zie Brainerd, 1971, hfdstk 5 voor een uitgebreider behandeling).

a. Labelled bracketings.

Def. Stellen we een kontekstvrije grammatika $G = \langle Vn, Vt, P, S \rangle$
Wanneer een regel $A \rightarrow \alpha$ wordt toegepast op een string $y = \chi_1 A \chi_2$ dan drukken we het resultaat uit als $\chi_1 [\alpha]_A \chi_2$ waarbij $[\alpha]_A$ betekent dat aan α het label A wordt toegevoegd.

Even een voorbeeld. Zij $Vn = \{S, NP, V, DT, N\}$ en $Vt = \{de, hond, slaapt\}$
Zij P =

$$\begin{aligned} S &\rightarrow NP\ V \\ NP &\rightarrow DT\ N \\ V &\rightarrow slaapt \\ N &\rightarrow hond \\ DT &\rightarrow de \end{aligned}$$

Een derivatie in G : $S \Rightarrow (NP\ V)_S \Rightarrow ((DT\ N)_{NP}\ V)_S \Rightarrow$

$$(((DE)_{DT}\ (HOND)_N\ )_{NP}\ V)_S \Rightarrow$$

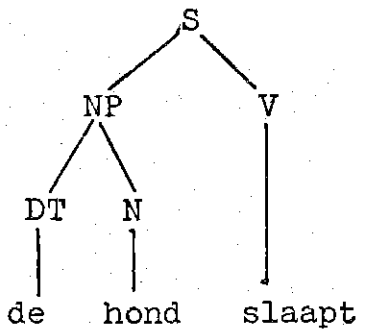
$$(((DE)_{DT}\ (HOND)_N)_{NP}\ (SLAAPT)_V\ )_S$$

Nota: Het is ook mogelijk om de 'label' rechts aan te brengen van het haakje en zelfs rechts en links. V.: $(NP\ V)_S$ of $(_S\ NP\ V)$ of $(_S\ NP\ V\ )_S$.

b. Boomdiagrammen.

Def. Stellen we een kontekstvrije grammatika $G = \langle Vn, Vt, P, S \rangle$
Wanneer een regel $A \rightarrow \alpha$ wordt toegepast op een string y, konstrueer een graf met als beginpunt (wortel) A, konstrueer evenveel knopen als er elementen zijn in α en benoem alle knopen met de elementen uit α .
Verbindt tenslotte de knopen met A.

Voorbeeld: De boomdiagram van bovenstaande derivatie is:



Het is wel bekend dat de generatieve grammatika zoals boven beschreven slechts een onderdeel is van de grammatika's die momenteel gangbaar zijn. Een belangrijke representatie die er o.a. bij gekomen is, zijn de zogen. feature matrices die we nu zullen bespreken.

Tot zover geeft de representatie informatie over de struktuur van de zin (uitgedrukt in de katgeorieën van de grammatika) en de funkties van de elementen (uitgedrukt door de positie t.o.v. de andere). Er is echter nog meer linguïstische informatie die t.a.v. de syntaxis kan gegeven worden.
Volgens de klassieke T.G.G.-opvatting bevat de syntaxis alle informatie die gebruikt wordt in de semantische interpretatie (hoe die semantische interpretatie er zelf zal uitzien wordt in een volgend onderdeel besproken), dit impliceert een bijkomend systeem dat semantisch verkeerde zinnen zou uitrangeren. Om dit te realiseren werden er drie stappen ondernomen (cfr Chomsky, 1965, 75-106).

(i) De syntaktische kategorieën werden scherper onderverdeeld: Bij de derivatie kan men aan elk kategoriaal symbool (vb. N) een zogen. feature matrix, d.i. een verzameling bepaalde syntaktische kenmerken, hangen zodat een komplex symbool ontstaat.

(ii) Een lexikon wordt ingevoerd met regels die niet meer van hetzelfde type zijn als de konstituentengrammatikaregels. Met name gaat het hier om paren (D,C), D is de fonologische distinctive feature matrix (waarvan hier abstraktie wordt gemaakt , en gewoon de spelling van het woord zelf genomen) en C is een verzameling van bepaalde syntaktische kenmerken zoals ze voorkomen in een derivatie. Lexikale insertie is dan juist het vervangen van Q, d.i. een komplex symbool dat in een derivatie voorkomt, door D wanneer C = Q.

(iii) Kontekstgevoelige subkategorisatieregels (selektierestrikties) werden ingevoerd om de samenstelling van de verzameling syntaktische kenmerken door allerlei restrikties, meestal voortvloeiend uit de kontekst, te regelen.

Wat ons hier nu interesseert is het nieuw aspekt van de representatie dat erbij komt, namelijk de feature matrix (voor kort f-matrix).

Def.: Een matrix is een verzameling waarvan de elementen geordend zijn in rijen en kolommen.
Vb;:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ \vdots & & & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{pmatrix}$$

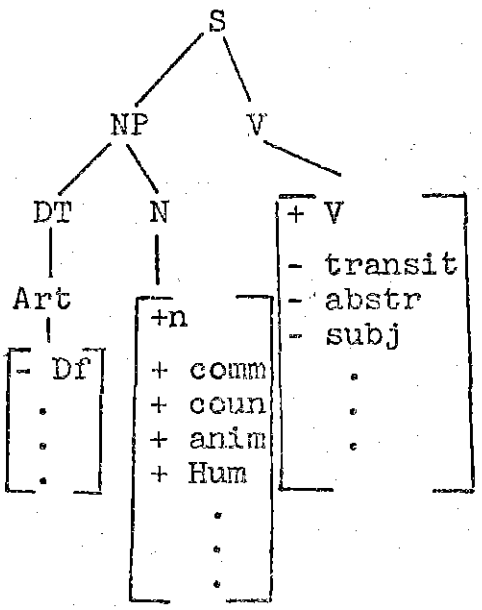
Wanneer nu de elementen van de matrix gedefiniëerde kenmerken zijn (features) spreekt men van een feature matrix.

Het idee om features te gaan gebruiken en om een verzameling symbolen te karakteriseren door een beperkt aantal (primitieve) features werd eerst in de fonologie toegepast. Men noemde de features 'distinctive'. Chomsky heeft deze zelfde notie overgenomen voor het specifiëren van de syntaktische features van een woord.
Zo zal een f-matrix (voor het woord 'boy') er als volgt uit zien

boy

+ N
+ common
+ count
+ animate
+ human

Een boomdiagram met f-matrices:



2. Representaties voor semantische informatie.

In deze paragraaf bespreken we bestaande metoden om semantische informatie te noteren. Hierbij zij opgemerkt dat dit aspekt erg vaag aan bod komt bij de generatieve grammatika's.
Er moet immers een onderscheid gemaakt worden tussen de dieptestruktuur die alle informatie bevat nodig voor een interpretatie door de semantische komponent en het resultaat zelf van de semantische komponent. Is dit een verzameling primitieve kenmerken bekomen door de projektieregels toe te passen op de feature-matrices van de verschillende formatieven? Als dit het geval is, dan kunnen we verwijzen naar het vorige hoofdstuk dat o,m. feature matrices behandelde. De semantische representatie is dan geen nieuw type van representatie en we laten het verder ongemoeid.
Is de semantische interpretatie echter een geordende verzameling primitieven die ook gegenereerd moet worden, dan is de semantische representatie een boomdiagram en ook hiervan werd de datastruktuur besproken in vorig hoofdstuk.
Er kan echter nog veel meer op het vlak van de semantische interpretatie dan de generatieve school laat uitschijnen.

We kunnen stellen dat het centrale probleem van de linguïstiek de betekenis is, dwz. hoe de natuurlijke taal wordt verstaan. Al gauw blijkt dat iemand veel meer weet (d.i. informatie haalt uit) een zin dan wat er exakt in deze zin verteld wordt. Dit verschijnsel noemt men presuppositie of implikatie. Ook blijkt dat die implikaties (of interferenties) niet enkel gebeuren in de zin zelf maar dat even goed informatie meegedeeld in vroegere zinnen mee wordt betrokken in het verstaansproces. De konstruktie van een geheugensysteem dat is gestruktureerd met het oog op interferenties dringt zich dus op. Dit systeem moet kunnen opgebouwd worden vanuit zinnen in natuurlijke taal en moet alle mogelijke interferenties kunnen maken die nodig zijn voor het 'verstaan' van uitdrukkingen.

Nu zijn er verschillende voorstellen gedaan om de semantische informatie te abstraheren en te organiseren in een struktuur zodanig dat de bovengeschetste aktiviteiten mogelijk zijn. Deze voorstellen zijn: het gebruik van de predikatencalculus, semantische netwerken, interferentiële geheugens en speciaal voor dit doel ontwikkelde programmeertalen.
(Zie voor een interessant overzicht Winograd, 1974).

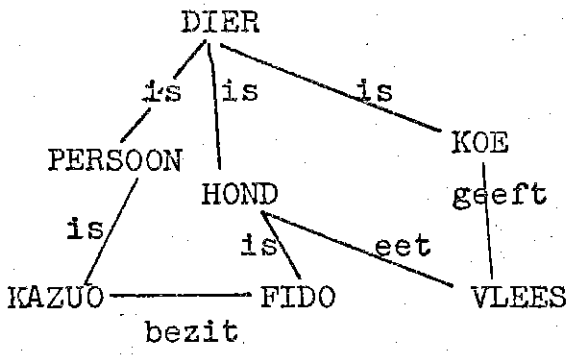
1. Predikatencalculus.

De predikatencalculus (al of niet versterkt door modale logika's) is een eerste mogelijke vorm van representaties van informatie. Het systeem is bekend genoeg om hier voor te moeten stellen. Onderzoek over de relatie tussen predikatenlogika en natuurlijke taal is aan de gang momenteel vooral vanuit de Montague grammars (Creswell, 1973, Partee, 1974, Montague, 19).

Ook op het gebied van de computerlinguïstiek (artificial intelligence) zijn er systemen gemaakt die als informatiebasis uitdrukkingen hebben in de predikatencalculus. Hierop kunnen interferenties gebeuren door een geautomatiseerde bewijsprocedure. Zie voor een voorbeeld van deze benadering : Sandewall, 1969,1974).

2. Semantische netwerken.

Semantische netwerken werden ontwikkeld om het oog op het vatten van de mogelijke implikaties en relaties tussen begrippen.
Een netwerk is een verzameling knopen waarop zich bepaalde elementen bevinden en verbindingen tussen de knopen die de relaties tussen deze elementen weergeven.
Even een voorbeeld (ontleend aan Winograd, 1974)



Dit netwerk kan door zinnen zoals FIDO is een hond, een hond is een dier, Fido eet vlees, een koe geeft vlees, enz... worden opgebouwd. De deduktie-kapaciteiten worden bepaald door de 'links' tussen de elementen. Bijv. op de vraag 'eet Fido vlees?' kan positief geantwoord worden omdat er een verbinding tussen Fido en hond is en tussen hond en vlees via eet-.
Er bestaat reeds een programma dat natuurlijke taal omzet in semantische netwerken en terug (Simmons, 1973), ook zijn er experimenten op gebied van tekstproduktie vanuit semantische netwerken. (Quillian, 1967).

3. Interferentiële geheugens.

De idee dat de verwerking van natuurlijke taal erin bestond om de informatie uit de zinnen te halen en deze op te slaan in een geheugen met interferentiële kapaciteiten komt reeds voor in 1961 bij Lindsay (1963). Al is het onderwerp (de familieverwantschapsnamen) nog summier en gebrekkig uitgewerkt, de problemen werden in alle geval toen reeds gesteld.
Het verschil tussen interferenti"ele geheugens en semantische netwerken is zodanig dat voor het laatste type de implikaties gegeven zijn in het netwerk zelf, dwz in het geheugen, terwijl voor het eerste type de implikaties berekend worden vanuit een semantisch model dat,wat er nodig is om iets op te lossen, opzoekt in het geheugen. Eigenaardig genoeg is de uitwerking van deze ideeën tot op het moment onbestaande.
Zie voor voorbeelden en uitgebreider behandeling Steels (1974 b)

4. Programmeertalen.

Vanuit de historische oproep van Türing: "I propose to consider the question "Can machines think?"" (Türing, 1963,11) is het onderzoek voor de kreatie van 'artificial intelligence' op gang gekomen. Binnen deze tak van de wetenschap houdt men zich niet enkel bezig met schaakspelletjes e.d. maar hoofdzakelijk met het zoeken naar algemene problem solving technieken. Bij het simuleren van intelligentie speelt ook de taal een grote rol, zodat ook op dit punt ernstig onderzoek op gang werd gebracht.
Om een probleem op te lossen moet het gesteld worden. D.w.z. moet er een representatie gegeven zijn die alle elementen bevat nodig voor het oplossen van een probleem. Deze representatie kan vele vormen aannemen. (zie voor een overzicht Ernst & Newell (1969), hfdst 5).

Maar niet alleen het probleem zelf moet goed gesteld zijn, ook voor de procedures zelf die problemen oplossen, moet er a.h.w. een taal bedacht worden zodat we kunnen uitdrukken welke procedures we nodig hebben. Deze taal is ook een programmeertaal in de gewone zin van het woord om-dat de procedures later moeten kunnen uitgevoerd worden. Voorbeelden van deze talen zijn: Planner, Conniver, Sail, QLisp, Popler, e.a. . Het gaat hier dus om een nieuwe generatie programmeertalen.

We verwijzen voor deze zeer belangrijke ontwikkeling naar Hewitt (1973); Bobrow & Raphael (1974) voor een overzicht en uitgebreide bibliografie; en het tijdschrift Artificial Intelligence waarin regelmatig bijdragen zullen verschijnen over de 'representation of knowledge' (cfr, nr 3, 1974, pag. 324).

Nota: Het procedureel definiëren van woorden sluit niet uit dat dit gekombineerd wordt met andere representaties. Voorbeelden hiervan zijn Steels (1974, a) waar een procedurële definitie wordt gekombineerd met interferentiële geheugens en Winograd(1974), hfdstk 6.'Frames: some ideas for a new formalism'. "Looking at the ideas described above at an oversimplified level, we might think of frames as combining a uniform way of expressing facts (like predicate calculus) with a detailed programming language for expressing programs and providing a scheme to tie them together. (ibid., p. 76, 1974)

2. Ongeordende bomen.

We zullen nu een voorstel doen om linguïstische informatie te representeren. Dit voorstel is o.m. gebaseerd op de technieken voor een formele definitie van programmeertalen (zie bijv. Lee (1973)) en op de datastrukturen van de programmeertaal LISP (cfr. infra).

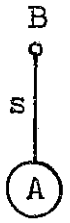
1. Definitie van ongeordende bomen

a. Een Boom.

Een boom wordt beschouwd als een verzameling geordende paren. Elk paar stelt een tak van de boom voor. Een paar bevat twee elementen: een selektor (s) en een objekt (A).

Als er slechts één paar aanwezig is in de verzameling spreken we van een simpel objekt.

Vb.: B = $\{\langle s,A\rangle\}$ of

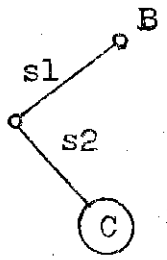


We kunnen nu op twee manieren samengestelde objekten maken:

(i) via inbedding:

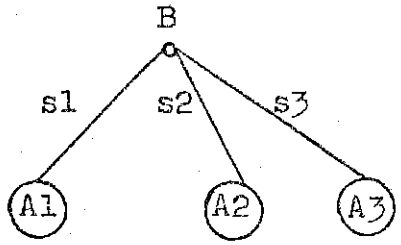
Vb.: B = $\{\langle s1,A\rangle\}$ en A = $\{\langle s2,C\rangle\}$

of B = $\{\langle s1, \{\langle s2,C\rangle\}\rangle\}$



(ii) via akkretie:

Vb.: B = $\{\langle s1,A1\rangle,\langle s2,A2\rangle,\langle s3,A3\rangle\}$



Nota:

1. Mengvormen zijn ook mogelijk. Vb.: A = $\{\langle s1,\{\langle s2,C\rangle,\langle s3,C\rangle\}\rangle\}$
2. Wanneer er geen objekt hangt onderaan spreken we van een lege boom (tak) en het objekt heet dan Ω. Vb.: A = $\{\langle s1,\Omega\rangle\}$

b. Vereenvoudiging van de notatie.

De notatie die werd gebruikt zou al snel onoverzichtelijk worden bij komplexe bomen. vandaar de volgende konventie: We schrijven de geordende paren tussen ronde haakjes en laten de akolades en komma's weg.
Vb.: B = (s1 A) is ekwivalent met {<s1,A>}
Vb. van inbedding: B = (s1 (s2 C)).
Vb. van akkretie: B = (s1 A1)(s2 A2)(s3 A3) .

c. Operaties over bomen.

1. De selektorfunktie (σ)

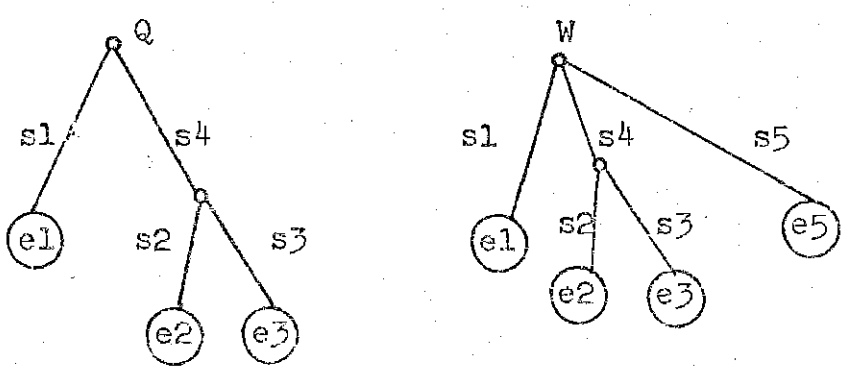
Deze funktie neemt als argumenten een boom en een selektor en geeft als output de s-komponent, d.i. datgene wat hangt aan de boom te beginnen bij de selektor.
Vb.: Zij B = (s1 A), dan is σ(s1 ; B) = A.
De selektor-funktie bevat dus het teken σ , en dan de argumenten gescheiden door een ';', de selektor wordt als eerste argument genoteerd.

Als met de selektor geen objekt korrespondeert, dan is het resultaat van de selektorfunktie gelijk aan nul.
Bij komplexe bomen kunnen we een samengestelde funktie toepassen.
Vb.: σ(s2 ; σ(s1 ; B)) levert C wanneer B = (s1 (s2 C)).
We schrijven: σ(s2 $\circ$ s1 ; B).
De σ-funktie wordt toegepast van rechts naar links, immers we berekenen eerst het resultaat met de hoogste selektor en op dit resultaat passen we de volgende selektor toe.

2. De mutatiefunktie

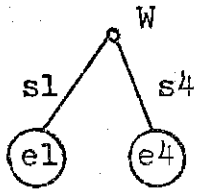
Met deze funktie kunnen we de samenstelling van bomen wijzigen. De mutatiefunktie is van de vorm μ(A ; (s , B)), waarbij μ de operator is, A de naam van de boom waarop we iets zullen veranderen en s,B resp. de selektor en het objekt van de nieuwe boom.

Bij een mutatie kunnen er drie dingen gebeuren:
(i) akkretie:
De selektor van de toe te voegen boom komt nog niet voor in de boom waarop we iets willen wijzigen. Dan wordt het objekt gewoon toegevoegd:
Vb.:



Q = (s1 e1)(s4 (s2 e2) (s3 e3)) wordt W = μ(Q ; (s5,e5))
of W = (s1 e1)(s4 (s2 e2)(s3 e3))(s5 e5). (W is de naam van de nieuw gevormde boom).

(ii) vervanging:
Als de selektor van de nieuw toe te voegen boom wel reeds aanwezig is in de boom waarin iets zal veranderd worden, dan wordt de boom vanaf de selektor vervangen door de nieuwe boom.
We nemen hetzelfde voorbeeld:
Q = (s1 e1)(s4 (s2 e2)(s3 e3)). W = μ(Q ; (s4,e4))
resulteert in W = (s1 e1)(s4 e4)

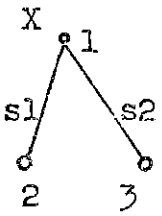


(iii) deletie:
Als de selektor reeds aanwezig is en als het objekt van de nieuwe boom Ω is, dan vindt er een deletie plaats.
Vb.: W = μ(Q; (s1,Ω)) = (s1,Ω)(s4 (s2 e2)(s3 e3)).

Nota: Het is ook mogelijk om een samengestelde selektor als element in de mutatie op te nemen. Zo kunnen we afdalen naar ingebedde bomen en die veranderen. Samengestelde selektors worden voorgesteld met een ^ tussen de selektors. Zij worden geschreven van boven naar beneden.
Vb.: W = μ(Q ; (s4^s3 , e4)), waarbij Q = (s1 e1) leidt tot
W = (s1 e1)(s4 (s3 e4))

d. Nog enkele termen.

- Afhankelijke elementen van een A zijn vertakkingen die beginnen in een bepaald punt A. Veronderstellen we boom X waarvan de knooppunten benoemd zijn:



In 1.beginnen er twee vertakkingen , namelijk naar 2 en 3 met resp. de labels s1 en s2.

- Anker: het anker van een vertakking is het beginpunt van die vertakking. In bovenstaand voorbeeld was het anker van 2 1 en van 3 ook 1.

- De onderste elementen, de objekten, die in een cirkeltje waren genoteerd, noemt men dikwijls elementaire objekten (Bekić (1973)), of atoms (in de programmeertaal LISP). De boom in zijn geheel wordt ook een objekt genoemd.

e. Bespreking.

1. Het verschil tussen geordende en ongeordende bomen is zodanig dat bij geordende bomen de verzameling een n-tupel is, dus een geordende rij.

2. In plaats van de knooppunten worden de verbindingen tussen de knopen benoemd. Dit is enkel om een flexibele verwerking van de datastruktuur te bekomen. Doordat de selektors om zo te zeggen 'pointers' zijn , zijn de deelbomen afhankelijk van deze selektors onmiddellijk beschikbaar.

3. Bovenstaande definitie moet gezien worden als een 'konseptuele' basis. De eigenlijke representatie die voor implementatie in aanmerking komt, bevat impliciet de konseptie van boomidagrammen zoals hier gedefiniëerd.

We gaan nu even in op het gebruik van boomdiagrammen op de onderscheiden niveaus van de linguïstische beschrijving.

2. Toepassingen in de linguïstiek.

a. Syntaxis.

Sinds de opkomst van de T.G.G. is het tekenen van bomen een dagelijkse bezigheid geworden voor linguïsten. Deze bomen hebben echter steeds een lichtelijk anders uitzicht, m.n. worden de knooppunten van de boom benoemd en niet de verbindingsstukken tussen de knopen. Nochtans is het ook mogelijk syntaktische bomen (strukturele deskripties) te beschouwen als bomen zoals in vorige paragraaf gedefiniëerd. Dit wordt meer en meer gedaan vooral door linguïsten die met de programmeertaal LISP werken om hun systemen te konstrueren en uit te testen.

Het belangrijkste onderscheid met de T.G.G.-bomen is, dat de elementen die afhangen van een knoop NIET GEORDEND zijn. Immers we hebben een boom gedefiniëerd als een verzameling geordende paren. De paren zelf zijn wel geordend maar de verzameling niet.

Vb.: $\{\langle s1,\{\langle s2,A1\rangle,\langle s3,A2\rangle\}\rangle\} = \{\langle s1,\{\langle s3,A2\rangle,\langle s2,A1\rangle\}\rangle\}$

De implikaties van deze eigenschap hebben vooral reperkusies op het gebied van de flexibiliteit van de representatie. Men ziet in dat bijvoorbeeld een groot deel van 'plaatsingstransformaties' hierdoor wegvalt.

Vergelijk: Jan is de vader van Karel
De vader van Karel is Jan
(dat) Jan de vader is van Karel
(dat)Jan van Karel de vader is
(dat) van Karel Jan de vader is
(dat) van Karel de vader Jan is

Ook wordt hierdoor een verbinding gemaakt tussen boomdiagramvoorstellingen en labelled bracketings.
Een voorbeeld zal dit laatste punt nog meer verduidelijken.

Vb.: Gegeven de grammatika: G =<Vn,Vt,P,S >

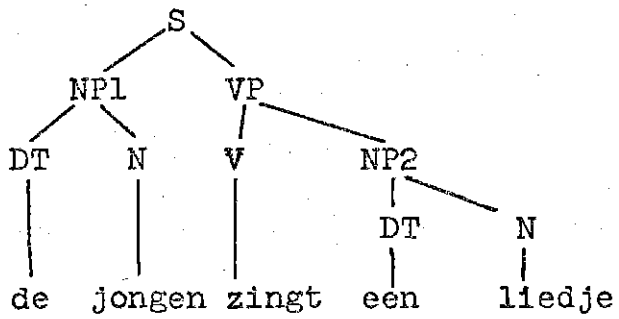
Vn = {NP1, VP, DT, N, V, NP2}

Vt = {de, jongen, zingt, een, liedje}

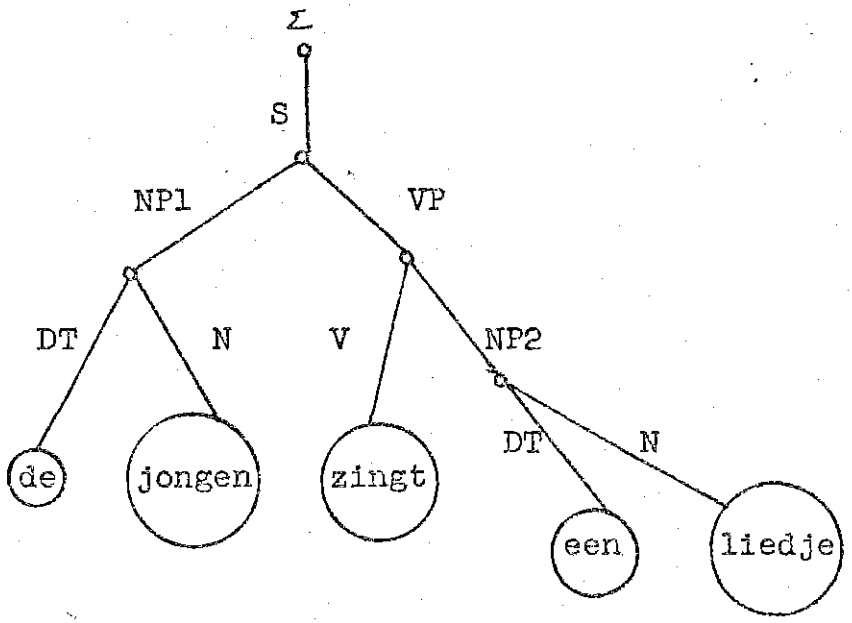
P = { S → NP1 VP
NP1 → DT N
VP → V NP2
NP2 → DT N
DT → de, een
N → liedje, jongen
V → zingt }

Gegeven de zin: 'De jongen zingt een liedje'.
De strukturele deskriptie hiervoor volgens grammatika G is:

Σ =



We schrijven om in een ongeordende boom:

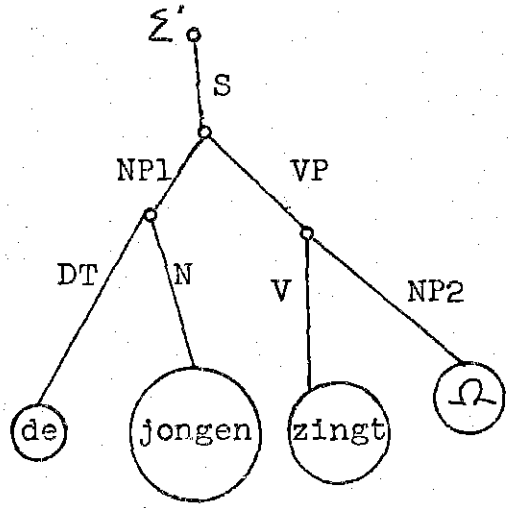


of (S (NP1 (DT de)(N jongen)(VP (V zingt)(NP (DT een)(N liedje))))

De lezer wordt aangeraden enkele boomdiagrammen om te schrijven in ongeordende bomen.

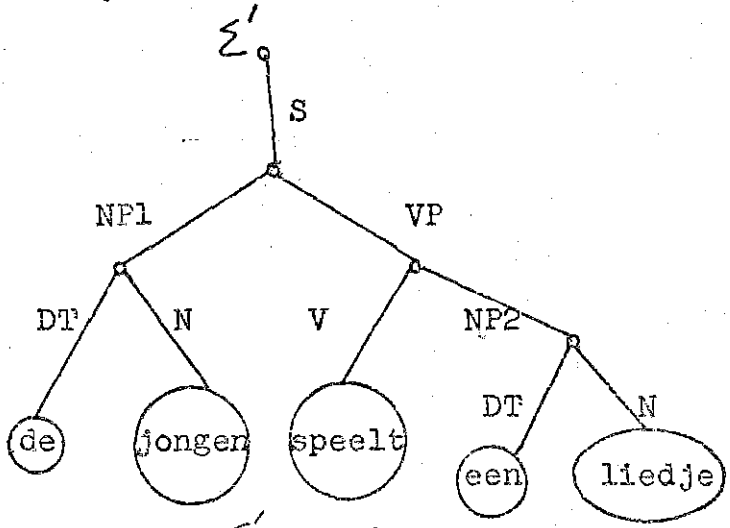
Transformaties die zo'n grote rol spelen in de T.G.G. kunnen ook op een eenvoudige manier worden beschreven binnen dit teoretische kader. We geven hiervan twee voorbeelden: een deletie-transformatie en een substitutie-transformatie.

Vb. 1. Deletie van 'een liedje' in 'de jongen zingt een liedje'. Uitgangspunt is de strukturele deskriptie in het vorige voorbeeld. Resultaat van de deletie is:



Representatie: $\Sigma' = \mu(\Sigma; (S\hat{}VP\hat{}NP2, \Omega))$

Vb. 2. Substitutie van'zingt' door'speelt' in 'de jongen zingt een liedje'. Uitgangspunt is hetzelfde als bij het vorige voorbeeld. Resultaat van de substitutie is:

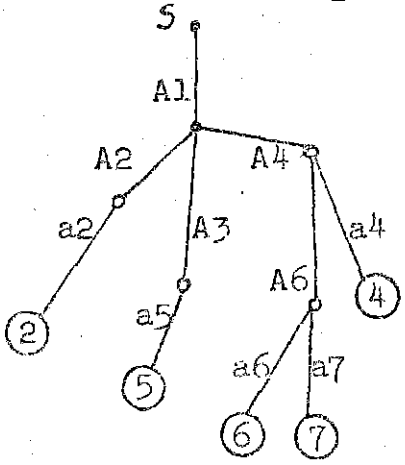


Representatie: $\Sigma' = \mu(\Sigma; (S\hat{}VP\hat{}V, speelt))$

Enkele suggesties voor verdere oefening:

1. Teken boom voor S = (A1 (A2 2)(A3 3))(A2 (A3 (A4 4)))

2. Geef de lineaire uitdrukking voor



3. Beschrijf de substitutie transformatie T rel in de zin "De vrouw de vrouw het volkslied zingt wandelt in het park". (voorbeeld ontleend aan V.de Velde, 1972, 109, e.v.)

4. Beschrijf de permutatietransformatie voor de zin: "de vrouw die zingt het volkslied wandelt in het parkt."

(Nota voor 3 en 4, ontwerp zelf de passende boomdiagrammen)

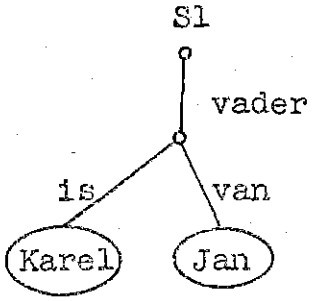
5. Beschrijf de aktief-passief transformatie in termen van mutatiefunkties voor de zin "de jongen speelt een liedje"

b. Semantiek

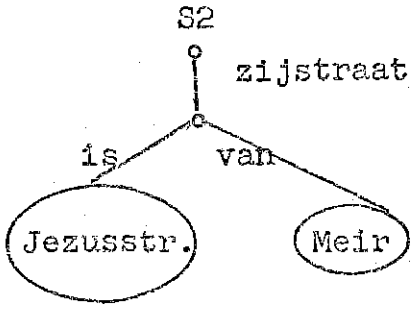
De boomdiagrammenkonseptie zoals voorgesteld is ook interessant bij de uitdrukking van een meer semantisch gerichte syntaktische struktuur die we de semantische normaalvorm zullen noemen. (Ekwivalente begrippen zijn de notie dieptestruktuur, casus-struktuur ,e.d.)

De semantische normaalvorm is de input voor de semantische komponent. Het zijn essentiëel relaties, funkties en predikaten met hun argumenten, geordend in een bepaalde hiërarchie. De snv. wordt opgevat als een boomdiagram waarvan de selektors lexikale elementen vormen, gedefiniëerd in het semantisch model en de objekten argumenten zijn van de lexikale elementen. Enkele voorbeelden zullen dit snel verduidelijken:

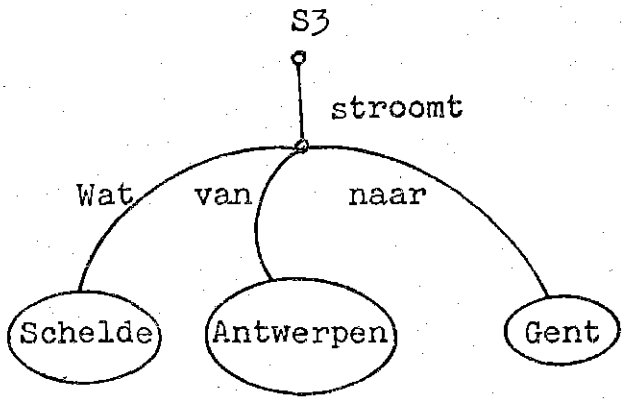
Vb.1: S1 = De vader van Karel is Jan.
de snv van S1 = (vader (is Karel)(van Jan))



Vb. 2. De jezusstraat is een zijstraat van de Meir. = S2
De snv van S2 = (zijstraat (is Jezusstraat)(van Meir))



Vb. 3. S3 = De Schelde stroomt van Antwerpen naar Gent.
De snv. van S3 = (stroomt (wat Schelde) (van Antwerpen)(naar Gent)

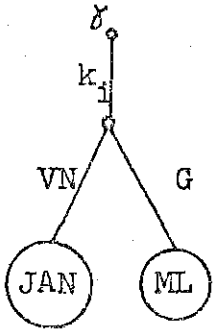


De lezer zal gemakkelijk inzien dat deze representatie de informatie uitgedrukt in een case-grammar kan weergeven en dat ook de semantische struktuur gegenereerd door een generatief semantisch opgezette grammatika in deze vorm kan uitgedrukt worden.

c. Interferentiële geheugens.

In Steels (1974 a) wordt onderzoek beschreven over de opbouw via natuurlijke taal en de interferenties over geheugens die zijn gevuld met informatie. Hierbij werden de ongeordende bomen als datastruktuur gebruikt.
Alhoewel er nog geen implementaties zijn die deze ideeën realiseren, is de datastruktuur in deze eerste faze erg vruchtbaar gebleken.

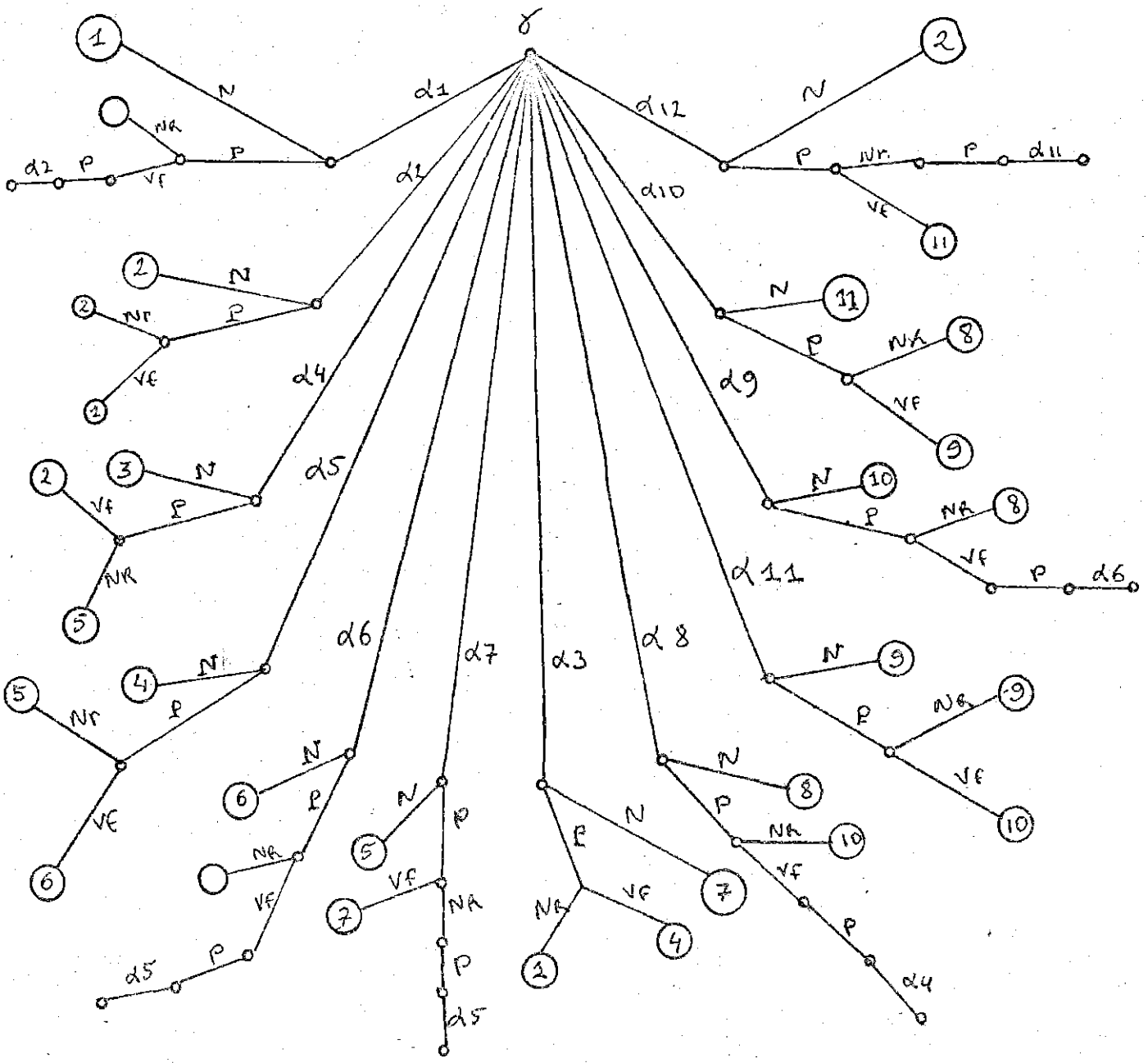
Het geheugen was hierbij een ongeordende boom waar de 'eerste' selektor telkens verwijst naar een objekt (entiteit); de 'tweede' selektors geven een predikaat aan met een waarde. Stel bijvoorbeelde informatie 'Jan is een man'. Het geheugen kunnen we ons dan zo voorstellen:



Er zijn dus twee predikaten: Voornaam (VN) en geslacht (G) waarvan de argumenten resp. zijn: Jan en Ml (mannelijk).

Al lijkt deze gang van zaken op het eerste zicht triviaal, toch komt men hierbij snel tot vrij komplexe strukturen en wat beter is, tot interessante interferenties als de predikaten goed zijn gekozen.

We geven op de volgende pagina even een voorbeeld van een geheugenkonfiguratie waarin allerlei straten zijn opgeslagen met hun positie in de stad. Deze positie is zodanig dat ze kan 'opgepikt' worden uit zinnen in natuurlijke taal en dat er achteraf interferentie mogelijk is. in dit geval meer bepaald dat de weg kan gevraagd worden.
Het voorbeeld is ontwikkeld door Joosen, et.al. en komt uit Steels (1974, a), pag. 29 e.v..



Nadat we in deze paragraaf een teoretisch kader hebben gekreëerd om linguïstische informatie te representeren, zullen we ons in de volgende paragrafen bezig houden met het 'hard maken' van deze definitie: hoe kunnen we bomen representeren in computers en hoe kunnen we operaties over bomen definiëren en implementeren.

II 1. Uitgangspunt: Datastrukturen bij LISP.

LISP ('list'-processing) is een programmeertaal ontwikkeld door Mc Carthy et.al. op M.I.T. rond 1960. De taal is vooral bekend door haar datastruktuur (namelijk lists), en doordat zij de status van een formele mathematische taal heeft waarin zeer vlot funkties kunnen gedefiniëerd worden.
LISP wordt momenteel aanzien als de programmeertaal bij uitstek voor linguïstische aktiviteiten, niet alleen omdat boomdiagrammen op een eenvoudige wijze erin gerepresenteerd worden, maar ook omdat er zo gemakkelijk nieuwe funkties worden gedefiniëerd.
Linguïsten die LISP gebruiken bij hun implementaties zijn bijvoorbeeld: M. Kay, S. Petrick, T. Winograd, W. Woods, R. Simmons, Y. Wilks, etc... .

Omdat we ons hier vooral interes eren voor datarepresentaties zullen we even (oppervlakkig) ingaan op de manier waarop dit in LISP gebeurt. Voor een diepgaande beschrijving van de programmeertaal LISP zie Mc Carthy (1962), Berkeley (1967), Weissman (1967) Woodward (1966).

1. Atomen, lijsten en variabelen.

De primitieve elementen van LISP worden 'atoms' genoemd.
Ze zijn niet verder deelbaar.
Een lijst bestaat uit een samenstelling van atomen.
We schrijven lijsten tussen ronde haakjes met een komma tussen de elementen(in LISP-systemen staan er ook soms punten tussen de elementen of in het geheel niets)
Een voorbeeld van een list is (A,B,C).

Variabelen zijn symbolische tekens (namen) die ofwel kunnen 'gebonden' worden aan een lijst, ofwel aan een atom.
Vb.: x = (A,B,C)
y = A

Merk het verschil op tussen (A) en A, het eerste is een list met het element A, het tweede is het atoom A.
Het spreekt vanzelf dat ingebedde lists ook kunnen voorkomen:
Vb. (A,(B,C),D)
(((A)))
Dit laatste is een list met als element een list met als element een list met als element een atoom.
Men noemt de uitdrukking van een list ook wel S-uitdrukking. (S-expression).

2. Hoofd en staart.

Het hoofd van een list is het eerste element. De staart is alle elementen behalve het eerste. We duiden hoofd en staart aan door Hd en St.
Voorbeelden: x = (A,B,C,D)
Hd(x) = A
St(x) = (B,C,D)
Wanneer een list slechts 1 atom bevat, wordt de staart gedefiniëerd als NIL. Nil is ook een atom.
Voorbeeld: x = (A)
St(x) = NIL.
x = (A,(B,(C)))
Hd(St(Hd(St(x)))) = (C).

De lezer zal reeds inzien dat de hoofdprogrammeerproblemen bij LISP erin bestaan om een element op te sporen via de elementaire hoofd en staart techniek. De LISP-ekwivalenten van Hd en St zijn CAR en CDR.

(i) CAR.
doel: het vinden van een deel van een S-expressie.
argument: één S-uitdrukking.
resultaat: Het eerste element van een S-uitdrukking.
vb. CAR ((A,B)) = A

(ii) CDR
doel: het vinden van een deel van een S-expressie
argument: een S-uitdrukking
Resultaat: het laatste element (de staart) van een S-uitdrukking.
vb. CDR ((A,B)) = B

3. Constructions.

Naast het uit elkaar halen van lists, is het ook interessant om listen te kunnen konstrueren. Dit gaat met de CONS-operatie. Hierbij wordt het eerste argument toegevoegd aan de lijst van het tweede argument.
Enkele voorbeelden:

x = (A,B)
y = (C,D)
CONS(x,y) = ((A,B),C,D)

x = (A,B)
y = (C,D,E)
CONS (Hd(x), Cons(Hd(st(x)),y)) = (A,B,C,D,E)

4. Predikaten

Het is ook mogelijk om funkties te definiëren; zoals predikaten dwz die als resultaat de waarde T (waar) of NIL (vals) hebben:
- Atom (x) = T als x waar is, anders NIL
- Eq (x,y) = T als x en y dezelfde atomen zijn, NIL als ze verschillend zijn en onbepaald als geen van beide atoms zijn.

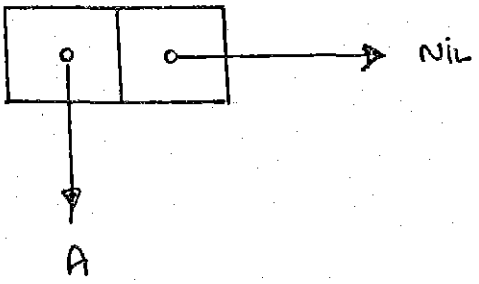
Vb.: x = (A,B)
Atom (x) = NIL

5. De definitie van funkties.

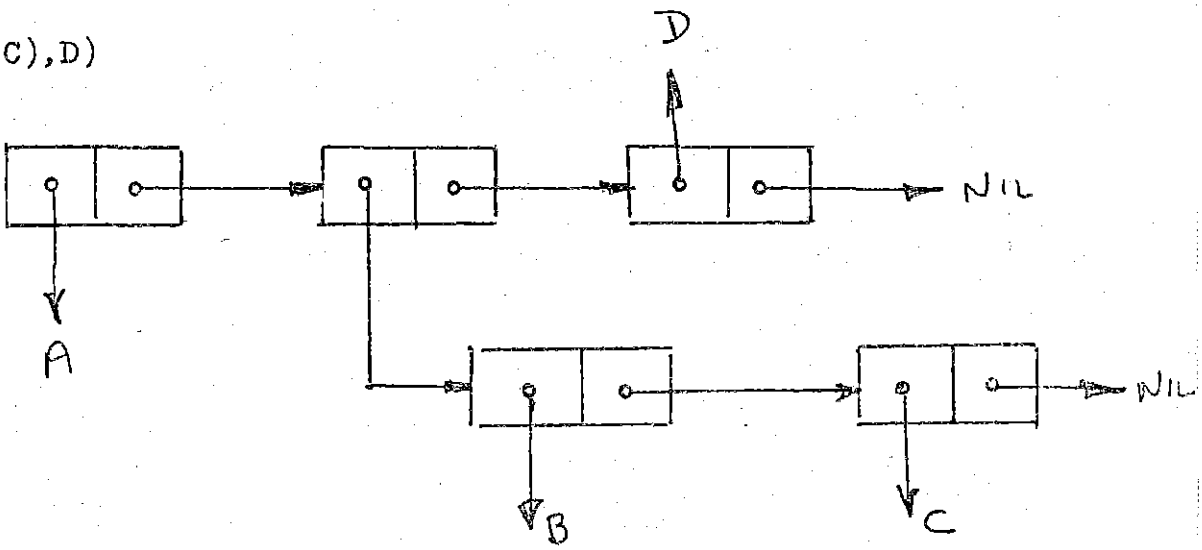
Het bijzonderste aspekt van LISP is wel dat er ruime mogelijkheden zijn om funkties te definiëren. Hiervoor zijn er hulpmiddelen ontwikkeld zoals λ-uitdrukkingen, conditional expressions (steunend op modus ponens), en vooral de mogelijkheid om rekursieve definities te programmeren. Wat ons hierbij echter alleen interesseert is de datastruktuur.

6. Machinerepresentatie.

Lijsten worden in LISP opgeslagen in paren waarvan het eerste element het 'adres' heet en het tweede element 'decrement'.
Vb. De lijst: (A) (herinner U dat het tweede element van een lijst met slechts één element NIL was)

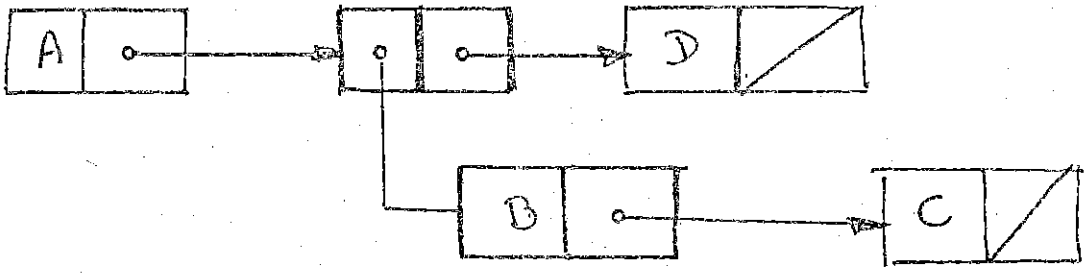


Vb. (A,(B,C),D)



NIL komt telkens op het einde van een list. In het bovenstaande voorbeeld werd de ene list in de andere ingebed.
We zien een pointer naar die lijst die begint met B.

Omdat het een beetje onhandig is met al die pointers geven we een verkorte notatie:



In de volgende paragraaf zullen we zelf een representatie van lijsten definiëren en implementeren die gemeen heeft met deze representatie dat er pointers worden gebruikt naar lists, en sublists. Nochtans zullen we het geheel niet opslagen in paren maar in een matrix.

Op de volgende pagina wordt deze uiteenzetting over LISP afgerond met enkele eenvoudige voorbeelden van LISP-operaties die we hebben geïmplementeerd en uitgevoerd in het Rekencentrum van de Universiteit van Pisa.

```
R; T=0.01/0.02 15.52.25

erase listing steels
R; T=0.01/0.02 15.52.25

erase listing ex1
R; T=0.01/0.02 15.52.35

lispe
EXECUTION BEGINS...
LISP VERSION 124-1
CORE IMAGE:  CL124.1  SOSTAP   P1
VALUE =  1

exf (steels)
EXF (STEELS)
VALUE =  (LISTING STEELS)

print listing steels
ILLEGAL LISP DOUBLET
PRINT LISTING
LISP



fin

END OF LISP
UNWINDCOUNT = 0
R; T=2.31/3310.14 15.53.34

print listing steels




SET (B123 (B A C))
VALUE =  (B A C)

SET (B124 (B C D))
VALUE =  (B C D)

SET (B125 (B (C (D A))))
VALUE =  (B (C (D A)))

SET (B126 (G (B (C (D A)))))
VALUE =  (G (B (C (D A))))

SET (A123 (A B C D))
VALUE =  (A B C D)

SET (A124 (B A C D))
VALUE =  (B A C D)

SET (A125 (((A) (B (C (D))))))
VALUE =  ( ((A) (B (C (D)))) )

SET (A126 ((D (C ((B (A)))))))
VALUE =  ( (D (C ((B (A))))) )

SET (A127 (S (NP (N YOU)) (VP (V PASS) (NP (DET THE) (N TEST)))))
VALUE =  (S
           (NP (N YOU))
           (VP (V PASS) (NP (DET THE) (N TEST))) )
```

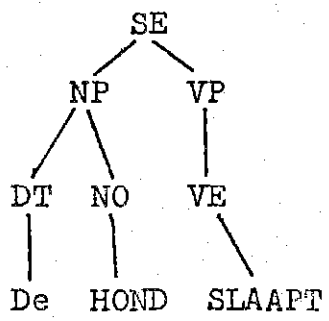
```
PRINT ((LUC STEELS))
(LUC STEELS)
VALUE =  (LUC STEELS)

EVAL ((CAR A123))
VALUE =  A

EVAL ((CADR A124))
VALUE =  A

EVAL ((CAAR A125))
VALUE =  (A)

EVAL ((CADAR (CADADR (CAR A126))))
VALUE =  (A)

DEFINE ( ( (ATEST
             (LAMBDA (L)
               (COND ( (NULL L)  NIL )
                     ( (ATOM (CAR L))
                       (COND ( (EQ (QUOTE A) (CAR L))  T )
                             ( T  (ATEST (CDR L)) ) ) )
                     ( (ATEST (CAR L))  T )
                     ( T  (ATEST (CDR L)) ) ) ) ) ) )
VALUE =  (ATEST)

EVAL ((ATEST B123))
VALUE =  *T*

EVAL ((ATEST B124))
VALUE =  NIL

EVAL ((ATEST B125))
VALUE =  *T*

EVAL ((ATEST B126))
VALUE =  *T*


R; T=0.10/0.51 15.55.15
```

2. Voorstel tot het representeren van lists via n-tupels.

We poneren geordende reeksen n-tupels met
(i) pointers die verwijzen naar andere n-tupels uit de rij
(ii) pointers die verwijzen naar labels van knopen of labels van verbindingen
(iii) informatie over de samenstellingen van n-tupels.

We zullen deze zaken nu uitwerken aan de hand van een voorbeeld:

Veronderstel de boomdiagram:

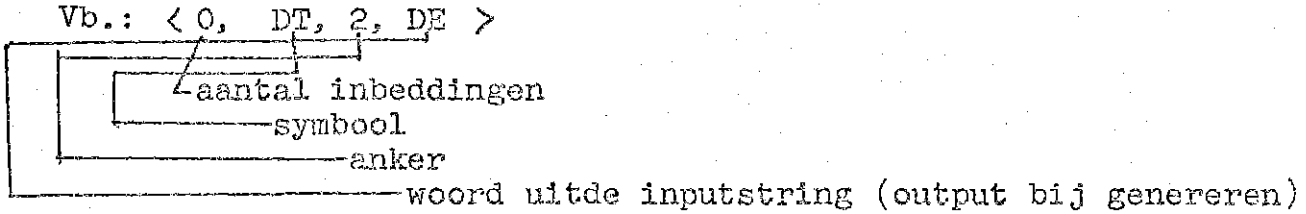


In list-notatie:
(SE (NP (DT de)(NO hond))(VP (VE slaapt)))

Zoals men kan zien vertrekken er twee aftakkingen uit SE die toekomen in nieuwe lists, namelijk NP en VP.
We stellen nu voor Se een n-tupel op.
α_1 : aantal afhankelijke elementen (in dit geval 2)
α_2 : SE zelf of een 'pointer' naar dit symbool.
α_3 : anker (= afkomst) van deze knoop. In dit geval de beginsituatie nul, in andere gevallen het n-tupel waar de aftakking begint die toekomt in het element genoemd in het n-tupel.
α_i : $3 \leq i \leq j$ geeft de inbeddingen aan. (Hier is $j = 2$)
α_i : voor $i > j$ is onbepaald.

Als we ook de n-tupels nummeren krijgen we als eerste:
$1. < 2, SE, 0, u, v >$ en u en v zijn de nummers van de n-tupels voor NP en VP.

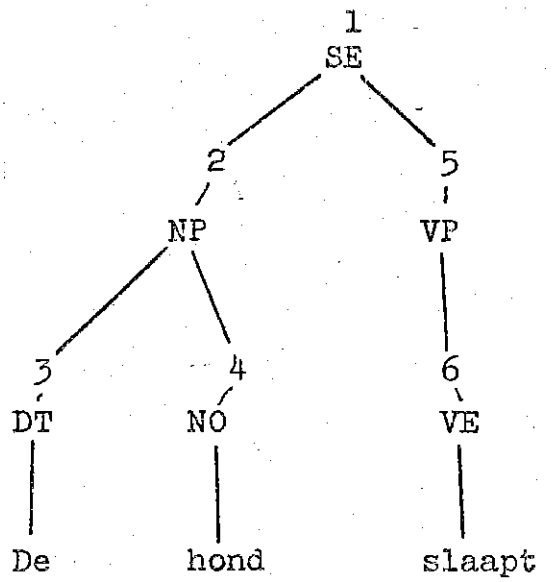
Bij terminale symbolen (waar er een interpretatie gebeurt) zijn er geen verdere inbeddi ngen meer. $\alpha 1$ is bijgevolg geijk aan nul. Bij konventie zetten we in $\alpha 4$ het woord zelf (eigenlijk opnieuw een pointer) dat de interpretatie uitmaakt;



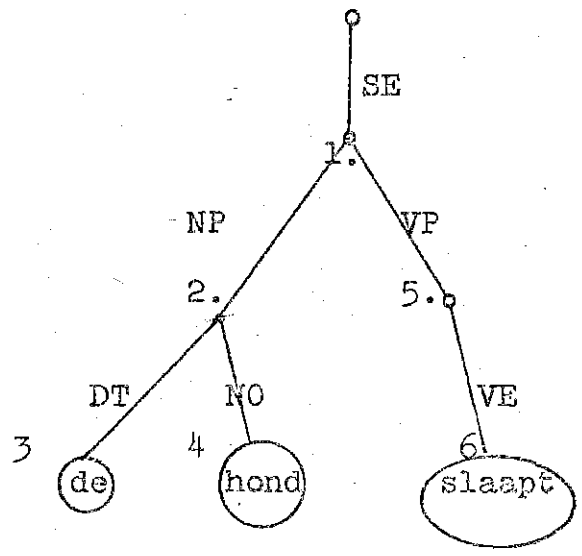
De lezer wordt aangeraden enkele voorbeelden door te werken op deze boom. De ganse boom wordt als volgt gekarakteriseerd.

1. < 2 , SE , 0 , 2, 5 >,
2. <2, NP , 1 , 3 ,4 >,
3. <0, DT , 2 , DE >,
4. <0, NO, 2, HOND >,
5. <1 , VP, 1 , 6 >,
6. <0 , VE , 5 , SLAAPT >.

Men kan nu deze representatie zowel beschouwen als een boom waarvan de knopen benoemd zijn (zoals we tot nu toe gedaan hebben) of waar de verbindingsstukken benoemd worden. In het eerste geval worden de knopen aldus bepaald:



in het tweede geval duidt $\alpha 3$ en $\alpha 1$ het nummer van resp. het begin en het einde van een lijnstuk met het label op $\alpha 2$



We definiëren nu de representatie:
Een boom is een geordende rij n-tupels met m elementen
1.$\langle \alpha 1, \alpha 2, \ldots . \alpha n \rangle$, 2.$\langle \alpha 1, \alpha 2, \ldots . \alpha n \rangle$, ...
m.$\langle \alpha 1, \alpha 2, \ldots \alpha n \rangle$.

zodanig dat αi voor $1 \leq i \leq n$ als volgt is gedefiniëerd.
voor
i = 1 : $n - 3 = \alpha i$
i = 2 : $\alpha 2$ is een positief getal p , voor $1 \leq p \leq$, zijnde zijnde een pointer naar een label, dwz het argument van een substring funktie voor een gedefiniëerde string.
i = 3 : $\alpha i =$ een positief getal p voor , $1 \leq p \leq m$;als k = 1 (k = nummer van het n-tupel in de rij) dan is $\alpha 3 = o$.
i = u: voor , u n-u u is een positief getal $1 \leq p \leq$ m zijnde een pointer naar een andere n-tupel uit de geordende rij van n-tupels.

Linguïsten willen echter niet zozeer mooie definities maar wel goede praktische werktuigen. We zullen ons nu gaan bezig houden met praktische problemen.

Deze problemen zijn:

1. Het konstrueren van boomdiagrammen vanuit syntaktische analyses.
2. Het 'format' van de output van een boom.
3. Het konstrueren van boomdiagrammen via genererïng
5. Het uitvoeren van transformaties op bestaande bomen.

Probleem 1. Het konstrueren van boomdiagrammen vanuit syntaktische analyses.

In Steels (1974 b) wordt een programma (SYNTAXTEST II) besproken dat gegeven een grammatika een strukturele deskriptie konstrueert voor een willekeurige zin uit het bereik van die grammatika, met andere woorden een programma voor automatische syntaktische analyse met de mogelijkheid om een grammatika in te voeren.
De outputprocedure die we toen hebben opgezet was niet erg gesofistikeerd maar wel effektief.
Het probleem was: hoe vinden we in de kladblok die het resultaat was van de syntaktische analyse het pad dat een goede parsing aangeeft en eens zo'n pad gevonden, hoe krijgen we de gepaste output. Dit laatste werd in een rekursiefmechanisme georganiseerd en wel als volgt:

Doorloop het pad in uw kladblok.
Is de syntaktische kategorie in de taak
a) een terminaal symbool:
geef dan als output een rechterhaakje,
gevolgd door de naam van de syntaktische kategorie
gevolgd door het woord uit de inputstring,
en gevolgd door een linkerhaakje.
b) het einde van een nonterminaal (↑)
geef als output een rechterhaakje
c) het begin van een nonterminaal symbool:
geef als output een linkerhaakje, gevolgd door
de naam van de syntaktische kategorie.

Het programma:

```
1860 FOR U=1 TO R
1870 Y=R-U+1
1880 IF D[R[Y],3]=50 THEN 1970
1890 IF D[R[Y],3]=0 THEN 1940
1900 IF D[R[Y],3]#1 THEN 1960
1910 IF D[R[Y],6]#R[Y] THEN 1980
1920 PRINT "(";W$[(D[R[Y],2]*2)-1,D[R[Y],2]*2];
1930 GOTO 1980
1940 PRINT "(";D$[(D[R[Y],2]*2)-1,D[R[Y],2]*2];" ";
1950 PRINT C$[P[D[R[Y],1]],E[D[R[Y],1]]];")";
1960 GOTO 1980
1970 PRINT ")";
1980 NEXT U
1990 PRINT
2000 PRINT
2010 NEXT Q
2020 PRINT
```

De output:

```
AXIOMA :SE

INPUT: KLAAS WERKT ALS ARBEIDER IN EEN FABRIEK .

KLADBLOK:
 1      1      1      0      1      1
 1      2      1      1      1      2
 1      3      1      1      2      3
 2      4      0      0     10      2
 2      5     50      4      0      2
 1      6      1      4     10      6
 2      7      1      5      1      1
 2      8      1      7      1      0
 2      9      1      7      2      9
 2     10      1      8     10     10
 2     11      0      9      0      9
 3     12     50     11      0      9
 3     13      1     12      2      1
 3     14      1     13      1     14
 3     15      1     13      2     15
 3     16      0     14     10     14
 4     17      1     16      0     14
 4     18      1     14     10     10
 4     19      1     17     10     19
 5     20      0     19      0     19
 5     21     50     20     10     19
 5     22     50     21     10     14
 5     23      1      1      1      1
 5     24      1     22      1     24
 5     25      1     23      2     25
 6     26      0     24      0     24
 6     27      1     26     10     24
 6     28      1     24     10     28
 6     29      1     27     10     29
 7     29      0     29     10     29
 7     30      1     30      1     29
 8     31      0     31      1     29
 8     32     50     32      2     29
 8     33     50     33     10     24
 8     34      1      1      1      1
 8     35      1     34      1     35
 8     36      1     35      2     37
 9     37      0     36      0      1
 9     38     50     38     10      1
 8     39      1     36     10     40

AANTAL PARSINGS: 1
STRUKTURELE DESKRIPTIE 1
PAD DOOR KLADBLOK:
 39     38     35     34     33     32     31     30     29     27     26     24
 23     22     21     20     19     17     16     14     13     12     11      9
  7      5      4      2      1
PARSING:
(SE(RA(EN KLAAS))(RN(VE WERKT))(RA(PR ALS)(NC(NO ARBEIDER)))(RA(PR IN)(
NC(DT EEN)(NO FABRIEK)))(PT .))
```

(Voorbeeld ontleend aan Steels (1974,b), pag. 27)

Wat we nu als probleem stellen is: Hoe verkrijgen we vanuit deze zelfde kladblok een boomdiagram zoals hij werd gedefini,eerd in vorige paragrafen.

Het ligt voor de hand om opnieuw te trachten een rekursief algoritme op te stellen. We nemen als matrix waarop de boomdiagram zal komen B.

Een boom werd gekarakteriseerd als een reeks n-tupels zodanig dat:
$\alpha 1$ = aantal afhankelijke elementen
$\alpha 2$ = naam van het symbool
$\alpha 3$ = anker
$\alpha 4$ = woord bij pretermina le $\alpha 2$;
anders αn voor $n > 4$ = afhankelijke elementen

Het algoritme kent 2 delen:
(i) de konsbruktie van de eerste rij op B, die het rekursief systeem in gang moet zetten
(ii) het rekursief algoritmen.

1. De konstruktie van de eerste rij.
Hierop komt het axioma van de grammatika :
$B(1,1) = o$
$B(1,2) = D(R(R),2)$ (= kode voor het axioma)
$B(1,3) = o$
voor $i > 4$, $B(1,i)$ is onbepaald.

2. Het rekursief algoritme
We maken een nieuwe rij op B telkens als er een nieuw symbool wordt ontmoet. Bij een terminaal symbool keren we terug naar de rij vóór we waren toen de nieuwe rij werd gemaakt. Bij het einde van een nonterminaal symbool gaan we een rij hoger dan voor de oorspronkelijke rij. Meer expliciet:

Doorloop het pad van de kladblok.
Is de syntaktische kategorie van de taak
a) een terminaal symbool:
(i) geef aan dat er nog een afhankelijk element is:
$\alpha 1 = \alpha 1 + 1$ en noteer op $\alpha_{i,\alpha 1}$ waar dit element zal terecht komen.
(ii) Konstrueer een nieuwe rij met $\alpha 1 = o$, $\alpha 2$ = het terminaal symbool, $\alpha 3$ = afkomst, $\alpha 4$ = woord uit de inputstring dat ermee overeenkomt.
b) een nonterminaal symbool.
is dit nonterminaal symbool al voorgekomen, doe niets; anders
(i) geef aan dat er nog een afhankelijk element is vertrekkend vanuit de rij waar we op bezig waren; en noteer op $\alpha_{i,\alpha 1}$ waar dit element zal terecht komen.
(ii) konstrueer een nieuwe rij met $\alpha 1 = o$, $\alpha 2$ = het nonterminaal symbool en $\alpha 3$ = de afkomst.

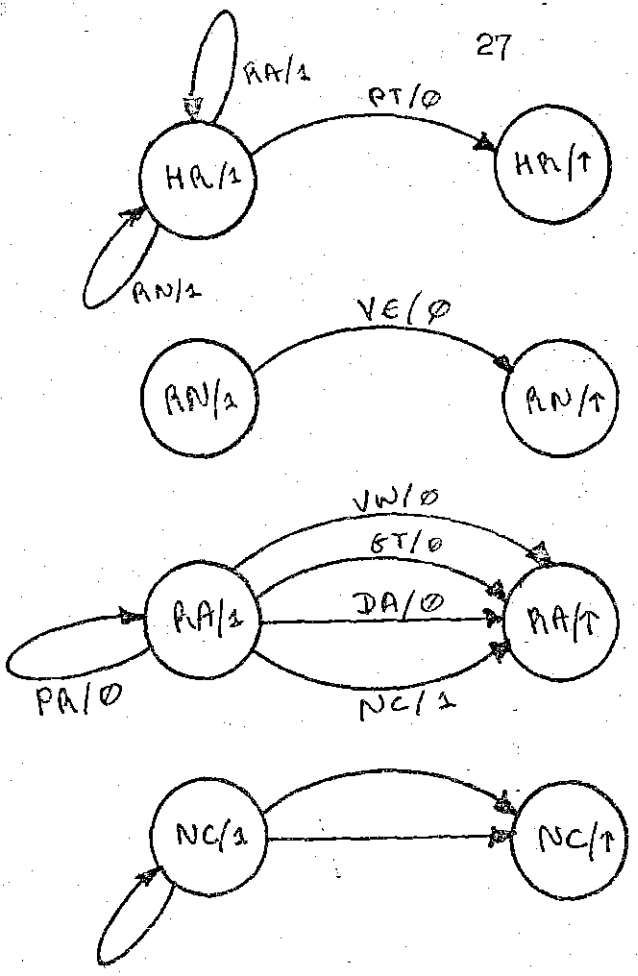
c) een einde van een nonterminaal symbool
Schuif dan op naar de rij aangegeven in $\alpha 3$ (= anker van de inbedding)

In het programma hebben we 2 tellers:
I1 : die de rij aangeeft waar we vertrekken (anker)
I2: waar de nieuwe taken telkens op komen.

Het programma:

```
1540 I1=1
1550 I2=2
1560 B[1,1]=0
1570 B[1,2]=D[R[R],2]
1580 B[1,3]=0
1590 FOR U=2 TO R
1600 Y=R-U+1
1610 IF D[R[Y],3]=50 THEN 1720
1620 IF D[R[Y],3]=0 THEN 1740
1630 IF D[R[Y],2]=B[I1,2] THEN 1810
1640 B[I1,1]=B[I1,1]+1
1650 B[I1,B[I1,1]+3]=I2
1660 B[I2,2]=D[R[Y],2]
1670 B[I2,1]=0
1680 B[I2,3]=I1
1690 I1=I2
1700 I2=I2+1
1710 GOTO 1810
1720 I1=B[I1,3]
1730 GOTO 1810
1740 B[I1,1]=B[I1,1]+1
1750 B[I1,B[I1,1]+3]=I2
1760 B[I2,1]=0
1770 B[I2,2]=D[R[Y],2]
1780 B[I2,3]=I1
1790 B[I2,4]=D[R[Y],1]
1800 I2=I2+1
1810 NEXT U
```

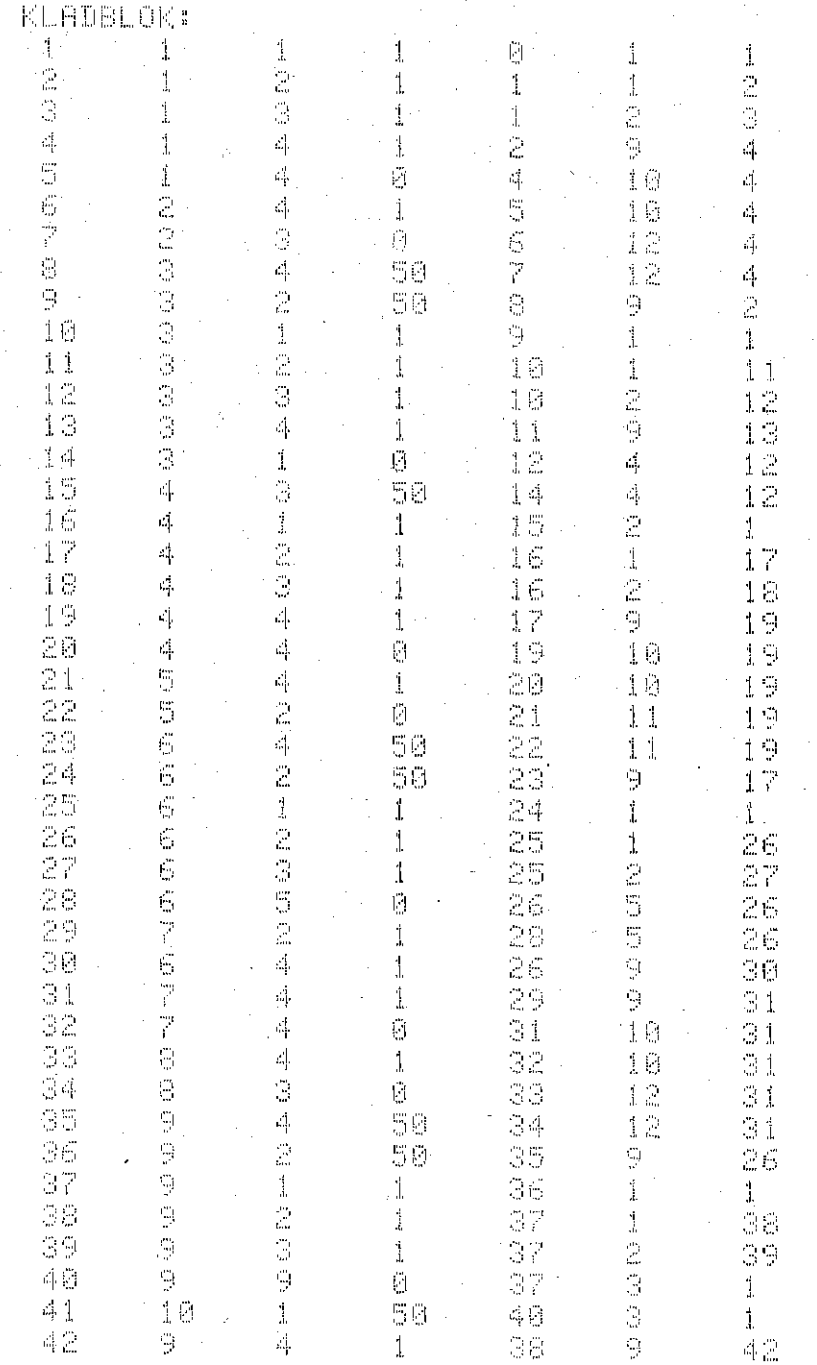
We proberen nu enkele 'runs'. Het programma is gekoppeld aan SYNTAXTEST II, i.p.v. een strukturele deskriptie wordt er nu een boomdiagram voor de ontleding gegeven. Deze boomdiagram is genoteerd op een matrix zoals die werd gedefini¨eerd in vorige paragraaf.
De grammatika is enigzins anders dan degene die gebruikt werd om SYNTAXTEST II uitgebreid te testen. We geven op de volgende bladzijde even de statetransitiediagram van de grammatika en daarna de interpretaties en de transitiematrix.




```
L.STEELS 1.12.74 TAPE 21 FILE 7
SYNTAXTEST II (NONDET. ATN-PARSER)
 2       1       1       1       1
 3       1       1       1       1
 9       0       1       1       50
 1       0       3       1       50
 5       0       2       1       1
 0       0       2       1       50
 7       0       2       1       50
 6       0       2       1       50
 4       1       2       1       50
 4       0       4       1       1
 2       0       4       1       50
 3       0       4       1       50
INPUT INTERPRETATIES:
VE =  IS GEBEURT WERKT SINGT ZINGEN SPEELT BIJT STREELT
NO =  VADER MOEDER DOCHTER ARBEIDER FABRIEK ZIJSTRAAT HOND KAT POES VISSEN SO
EN =  JAN PIET KLAAS KAREL AMEDEE LIESJE ANNIE RIA FRANKRIJKLEI KEIZERLEI ANT
DT =  DE HET EEN GEEN VEEL SOMMIGE ENKELE
PR =  VAN ALS IN OP NA VOOR TUSSEN EN OF TEGEN ON
DA =  1-12-1974
GT =  1 2 3 4 5 6 7 8 9 0 10 11 12 13 14 15 1000 100 10.000
VW =  WIE WAT WAAR WANNEER HOEVEEL
PT =  . , ? ! ;
↑  =
TRANSITIEMATRIX
AXIOMA :SE
```

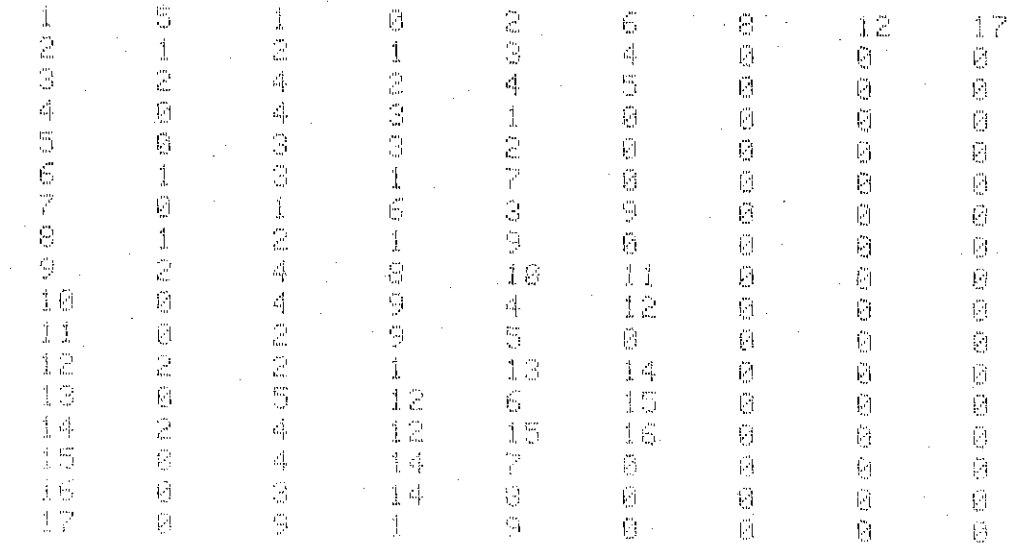
INPUT: DE FRANKRIJKLEI IS EEN ZIJSTRAAT VAN DE PAARDEMARKT .

KLADBLOK:



AANTAL PARSINGS: 1

MATRIX VAN SYNT.ANAL.



INPUT: HOEVEEL IS DE SOM VAN 1 EN 2 ?

KLADBLOK:

1	1	1	1	0	1	1
2	1	2	1	1	1	2
3	1	3	1	1	2	3
4	1	8	0	2	6	2
5	2	2	50	4	6	2
6	1	4	1	2	9	6
7	2	1	1	5	1	1
8	2	2	1	7	1	8
9	2	3	1	7	2	9
10	2	4	1	8	9	10
11	2	1	0	9	4	9
12	3	3	50	11	4	9
13	3	1	1	12	2	1
14	3	2	1	13	1	14
15	3	3	1	13	2	15
16	3	4	1	14	9	16
17	3	4	0	16	10	16
18	4	4	1	17	10	16
19	4	2	0	18	11	16
20	5	4	50	19	11	16
21	5	2	50	20	9	14
22	5	1	1	21	1	1
23	5	2	1	22	1	23
24	5	3	1	22	2	24
25	5	5	0	23	5	23
26	6	2	1	25	5	23
27	5	4	1	23	9	27
28	6	7	0	26	7	23
29	7	2	50	28	7	23
30	6	4	1	26	9	30
31	7	1	1	29	1	1
32	7	2	1	31	1	32
33	7	3	1	31	2	33
34	7	5	0	32	5	32
35	8	2	1	34	5	32
36	7	4	1	32	9	36
37	8	7	0	35	7	32
38	9	2	50	37	7	32
39	8	4	1	35	9	39
40	9	1	1	38	1	1
41	9	2	1	40	1	41
42	9	3	1	40	2	42
43	9	9	0	40	3	1
44	10	1	50	43	3	1
45	9	4	1	41	9	45

AANTAL PARSINGS: 1

MATRIX VAN SYNT.ANAL.

1	6	1	0	2	4	6	10	13
2	1	2	1	3	4	0	0	0
3	0	8	2	1	5	0	0	0
4	1	3	1	5	0	0	0	0
5	0	1	4	2	0	0	0	0
6	1	2	1	7	0	0	0	0
7	2	4	6	8	9	0	0	0
8	0	4	7	3	0	0	0	0
9	0	2	7	4	11	0	0	0
10	2	2	1	11	12	0	0	0
11	0	5	10	5	0	0	0	0
12	0	7	10	6	14	0	0	0
13	2	2	1	14	15	0	0	0
14	0	5	13	7	16	0	0	0
15	0	7	13	8	0	0	0	0
16	0	9	1	9	0	0	0	0

Probleem 2. De output van boomdiagrammen.

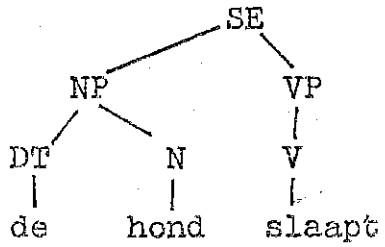
Een matrix met allerlei getallen op is voor menselijke gebruikers van het systeem niet direkt de interessantste manier van voorstellen. We zullen nu trachten om de informatie aanwezig in een boom-matrix weer te geven op een bevattelijke en aanschouwelijke manier.

Er zijn hiertoe twee mogelijkheden:
- ofwel in de vorm van een string met labelled bracketings
- ofwel in een boomdiagram.

We hebben reeds gezien (namelijk voor SYNTAXTEST II) dat het gemakkelijk gaat om labelled bracketings als output te konstrueren. Boomdiagrammen echter zijn een erg ingewikkelde zaak, niet zozeer teoretisch, wel om te bladschikking te organiseren. Het is bijna niet te voorspellen hoeveel inbeddingen er zullen komen, hoeveel van elkaar de verschillende symbolen nu moeten staan om onderaan geen moeilijkheden te krijgen, enz... . De output van bomen is slechts mogelijk wanneer we over een 'plotter' kunnen beschikken wat momenteel niet het geval is.
We gaan dan ook een oplossing volgen die het midden houdt tussen labelled bracketings en boomdiagrammen, een oplossing die overigens meestal gebruikt wordt bij output van syntaktische analyses en die min of meer overeenkomt met de output in LISP.

De idee is om te werken met inspringende lijnen telkens als er een diepere inbedding plaats vindt. De elementen die hiërarchisch van gelijke waarde zijn staan onder elkaar. Een voorbeeld zal dit snel verduidelijken:

Gegeven:



S.D. met lab. brac. : (SE (NP (DT de)(N hond))(VP (V slaapt)))

We stellen als nieuwe output(die we edited labelled bracketings zullen noemen, afgekort E.L.B.)

```
(SE    (NP    (DT    de)
              (N     hond))
       (VP    (V     slaapt)))
```

Analyse van het systeem:

We hebben 3 mogelijkheden:
a) het begin van een nonterminaal symbool, dan verschijnt er als output: '(A' en A ε Vn
b) het einde van een nonterminaal symbool, dan krijgen we enkel ")"
c) een terminaal symbool met de interpretatie, dan verschijnt '(a α)' en a ε Vt en α ε C$ (de inputstring)

Daarbij komt nog dat telkens er een nieuwe inbedding begint, we verder naar rechts moeten en telkens we een hiërarchisch evenwaardig element ontmoeten,wen naar onder gaan.

Om deze problemen op te lossen voeren we een soort nieuwe kladblok in, namelijk O. Hierop komt in O (I1,1) het nummer van de matrix van de boom en in O(I1,2) het aantal afhankelijke elementen per nonterminaal symbool.
O wordt gevormd (en afgebroken) tijdens het outputproces.
Het heeft de eigenschap van een pushdownstore (last in first out).
Iedere keer als we dieper gaan met een inbedding gaan we ook een rij verder op O, wanneer het einde van een nonterminaal is bereikt, stijgen we op in O. Het einde van een nonterminaal kunnen we vaststellen als het aantal afhankelijke lelementen is opgebruikt. We trekken dus iedere keer van O(i1,2) af als er een inbedding wordt uitgeprint.

Het programma valt uiteen in 2 delen: één voor terminale en één voor nonterminale.
Wanneer een terminaal symbool wordt uitgeprint gaan we na of er nog afhankelijke elementen aanwezig zijn die nog niet werden verwerkt. Is dit het geval, schuiven we een regel naar onder op de output en gaan we naar de faze waar de verwerking van een inbedding begon. Is dit niet het geval, dan geven we als output nog een haakje (het einde van een nonterminaal werd immers bereikt).

Het programma:

```
1950 I2=1
1960 I1=0
1970 IF B[I2,1]=0 THEN 2050
1980 PRINT TAB(I1*6)+1;"(";N$[(B[I2,2]*2)-1,B[I2,2]*2];
1990 I1=I1+1
2000 O[I1,1]=I2
2010 O[I1,2]=B[I2,1]
2020 O[I1,2]=O[I1,2]-1
2025 IF B[I2,B[I2,1]-O[I1,2]+3]=0 THEN 2020
2030 I2=B[I2,(B[I2,1]-O[I1,2]+3)]
2040 GOTO 1970
2050 PRINT TAB(I1*6)+1;"(";D$[(B[I2,2]*2)-1,B[I2,2]*2];" ";
2060 PRINT O$[P[B[I2,4]],E[B[I2,4]]];")";
2070 IF I1=0 THEN 2160
2080 IF O[I1,2]#0 THEN 2120
2090 PRINT ")";
2100 I1=I1-1
2110 GOTO 2070
2120 PRINT
2130 PRINT
2140 I2=O[I1,1]
2150 GOTO 2020
2160 PRINT
```

Bij wijze van voorbeeld geven we nu alle boomdiagrammen van de syntaktische analyses van vorige paragraaf.

```
STRUKTURELE DESKRIPTIE:
 (SE   (RA   (NC   (DT DE)
                   (EN FRANKRIJKLEI))
       (RN   (VE IS))
       (RA   (NC   (DT EEN)
                   (NO ZIJSTRAAT)))
       (RA   (PR VAN)
             (NC   (DT DE)
                   (EN PAARDEMARKT)))
       (PT .))
```

```
STRUKTURELE DESKRIPTIE:
 (SE   (RA   (VW HOEVEEL))
       (RN   (VE IS))
       (RA   (NC   (DT DE)
                   (NO SOM)))
       (RA   (PR VAN)
             (GT 1))
       (RA   (PR EN)
             (GT 2))
       (PT ?))
```

```
AXIOMA :SE
INPUT: SOMMIGE VISSEN ZINGEN VOOR LIESJE .

AANTAL PARSINGS: 1
STRUKTURELE DESKRIPTIE:
 (SE   (RA   (NC   (DT SOMMIGE)
                   (NO VISSEN))
       (RN   (VE ZINGEN))
       (RA   (PR VOOR)
             (NC   (EN LIESJE)))
       (PT .))
```

```
AXIOMA :NC
INPUT: EEN HOND

AANTAL PARSINGS: 1
STRUKTURELE DESKRIPTIE:
 (NC   (DT EEN)
       (NO HOND))
```

```
AXIOMA :SE
INPUT: DE DOCHTER VAN AMEDEE IS DE MOEDER VAN KAREL .

AANTAL PARSINGS: 1
STRUKTURELE DESKRIPTIE:
 (SE   (RA   (NC   (DT DE)
                   (NO DOCHTER)))
       (RA   (PR VAN)
             (NC   (EN AMEDEE)))
       (RN   (VE IS))
       (RA   (NC   (DT DE)
                   (NO MOEDER)))
       (RA   (PR VAN)
             (NC   (EN KAREL)))
       (PT .))
```

Probleem 3. Het uitvoeren van transformaties op bestaande bomen

Dit probleem zullen we slechts oppervlakkig behandelen. Met oppervlakkig bedoelen we echter geenszins niet exakt, wel dat er geen appart programma zal opgezet worden. Er zal een gerealiseerde mogelijkheid worden geboden om in een bestaand programma rechtstreeks in te grijpen op de opgeslagen informatie.

Uitgangspunt in se strukturele deskriptie voor de zin
"De Frankrijklei is een zijstraat van de paardemarkt".
Door hierop transformaties uit te voeren zullen we het principe illustreren.

1. Deletie

We zouden bijvoorbeeld de uitdrukking "van de paardemarkt" (met de strukturele deskriptie erbij) willen deleteren. Dit kan door eenvoudig in te geven "B(1,7) = o", "Execute". In het BASIC-systeem kan men regels ingeven (die worden uitgevoerd) en dan ergens in het programma verder gaan (via de Cont-funktie).
Wat hier meer bepaald gebeurt,is dat de 'pointer' vanuit SE naar de deelboom waar "van de paardemarkt" aanhangt op nul wordt gezet.
Vergelijk even de matrix van de syntaktische analyse, en de strukturele deskriptie van de twee bomen, de ene voor de deletie-transformatie, de andere erna. Het programma is Syntaxtest II met daaraan een systeem om boomdiagrammen te konstrueren en een systeem om een output te krijgen in E.L.B..

```
STRUKTURELE DESKRIPTIE: (1)
(SE   (RA   (NC   (DT DE)
                  (EN FRANKRIJKLEI))
      (RN   (VE IS))
      (RA   (NC   (DT EEN)
                  (NO ZIJSTRAAT))
      (RA   (PR VAN)
            (NC   (DT DE)
                  (EN KEIZERLEI)))
      (PT .))
```

A

MATRIX VAN SYNT.ANAL.(1)

1	5	1	0	2	6	0	12	17
2	1	2	1	3	0	0	0	0
3	2	4	2	4	5	0	0	0
4	0	4	3	1	0	0	0	0
5	0	3	0	2	0	0	0	0
6	1	3	1	7	0	0	0	0
7	0	1	5	3	0	0	0	0
8	1	2	1	9	0	0	0	0
9	2	4	8	10	11	0	0	0
10	0	4	9	4	0	0	0	0
11	0	2	9	5	0	0	0	0
12	2	2	1	13	14	0	0	0
13	0	2	12	6	0	0	0	0
14	2	4	12	15	16	0	0	0
15	0	4	14	7	0	0	0	0
16	0	3	14	8	0	0	0	0

MATRIX VAN SYNT.ANAL.(2)

1	5	1	0	2	6	0	0	17
2	1	2	1	3	0	0	0	0
3	2	4	2	4	5	0	0	0
4	0	4	3	1	0	0	0	0
5	0	3	0	2	0	0	0	0
6	1	3	1	7	0	0	0	0
7	0	1	5	3	0	0	0	0
8	1	2	1	9	0	0	0	0
9	2	4	8	10	11	0	0	0
10	0	4	9	4	0	0	0	0
11	0	2	9	5	0	0	0	0
12	2	2	1	13	14	0	0	0
13	0	2	12	6	0	0	0	0
14	2	4	12	15	16	0	0	0
15	0	4	14	7	0	0	0	0
16	0	3	14	8	0	0	0	0

STRUKTURELE DESKRIPTIE: (2)

```
(SE   (RA   (NC   (DT DE)
                  (EN FRANKRIJKLEI)))
      (RN   (VE IS))
      (RA   (NC   (DT EEN)
                  (NO ZIJSTRAAT)))
      (PT .))
```

2. Substitutie

We gaan uit van de s.d. van de zin'de Frankrijklei is een zijstraat van de Keizerlei'. Nu willen we 'De Frankrijklei' vervangen door 'De Keizerlei' en omgekeerd.
Dit kan door de volgende reeks ter uitvoering in te geven:

```
Z =B(2,4)
B(2,4) = B(12,5)
B(12,5) = Z
```

We veranderen dus de 'pointers' naar de NC afhangend van de eerste RA door de 'pointer' naar de NC afhangend van de laatste RA en omgekeerd:

MATRIX VAN SYNT.ANAL. (1)

1	5	1	0	2	6	0	12	17
2	1	2	1	3	0	0	0	0
3	2	4	2	4	5	0	0	0
4	0	4	3	1	0	0	0	0
5	0	3	3	2	0	0	0	0
6	1	3	1	7	0	0	0	0
7	0	1	6	3	0	0	0	0
8	1	2	1	9	0	0	0	0
9	2	4	8	10	11	0	0	0
10	0	4	9	4	0	0	0	0
11	0	2	9	5	0	0	0	0
12	2	2	1	13	0	0	0	0
13	0	5	12	6	0	0	0	0
14	2	4	12	15	16	0	0	0
15	0	4	14	7	0	0	0	0
16	0	3	14	8	0	0	0	0

MATRIX VAN SYNT.ANAL. (2)

1	5	1	0	2	6	0	12	17
2	1	2	1	3	0	0	0	0
3	2	4	2	4	5	0	0	0
4	0	4	3	1	0	0	0	0
5	0	3	3	2	0	0	0	0
6	1	3	1	7	0	0	0	0
7	0	1	6	3	0	0	0	0
8	1	2	1	9	0	0	0	0
9	2	4	8	10	11	0	0	0
10	0	4	9	4	0	0	0	0
11	0	2	9	5	0	0	0	0
12	2	2	1	13	0	0	0	0
13	0	5	12	6	0	0	0	0
14	2	4	12	15	16	0	0	0
15	0	4	14	7	0	0	0	0
16	0	3	14	8	0	0	0	0

STRUKTURELE DESKRIPTIE: (1)

```
(SE   (RA   (NC   (DT DE)
                  (EN FRANKRIJKLEI)))
      (RN   (VE IS))
      (RA   (NC   (DT EEN)
                  (NO ZIJSTRAAT)))
      (RA   (PR VAN)
            (NC   (DT DE)
                  (EN KEIZERLEI)))
      (PT .))
```

STRUKTURELE DESKRIPTIE: (2)

```
(SE   (RA   (NC   (DT DE)
                  (EN KEIZERLEI)))
      (RN   (VE IS))
      (RA   (NC   (DT EEN)
                  (NO ZIJSTRAAT)))
      (RA   (PR VAN)
            (NC   (DT DE)
                  (EN FRANKRIJKLEI)))
      (PT .))
```

De lezer zal wel al ingezien hebben dat transformaties uitvoeren gewoon het veranderen van pointers is. Transformaties gaan daardoor zeer snel in vergelijking met de andere linguïstische operaties. De moeilijkheid is natuurlijk welke transformaties er nodig zijn, doch dit ligt buiten het bestek van deze tekst. We geven nog een laatste voorbeeld, namelijk een verplaatsing (permutatie).

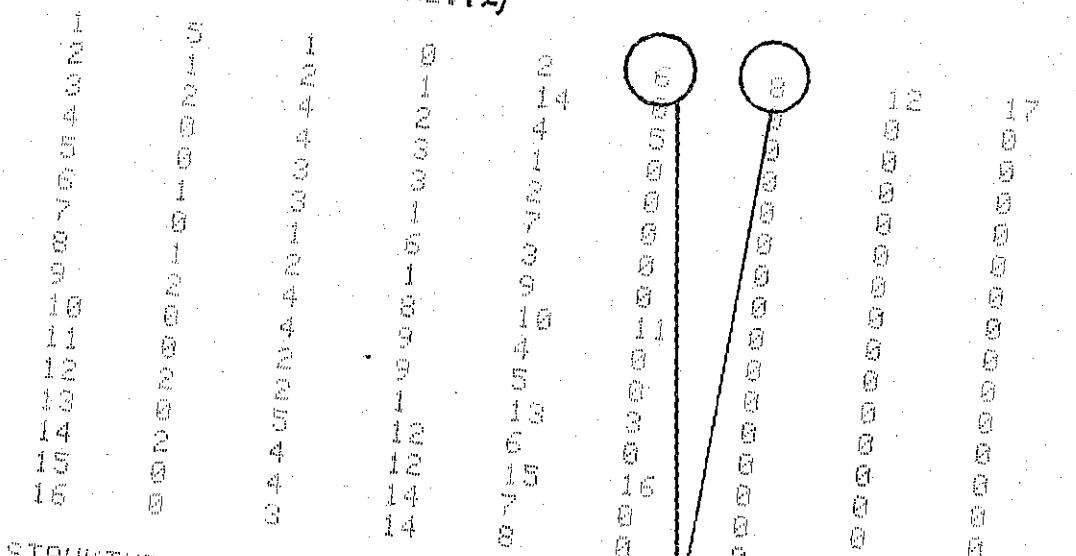
3. Verplaatsing.

Uitgangspunt is opnieuw 'De keizerlei is een zijstraat van de Frankrijklei'. Nu willen we als uiteindelijke output de strukturele deskriptie van 'De Keizerlei een zijstraat is van de Frankrijklei.' Dit gaat als volgt:

Z = B(1,5)
B(1,5) = B(1,6)
B(1,6) = Z

Het omwisselen van pointers dus.

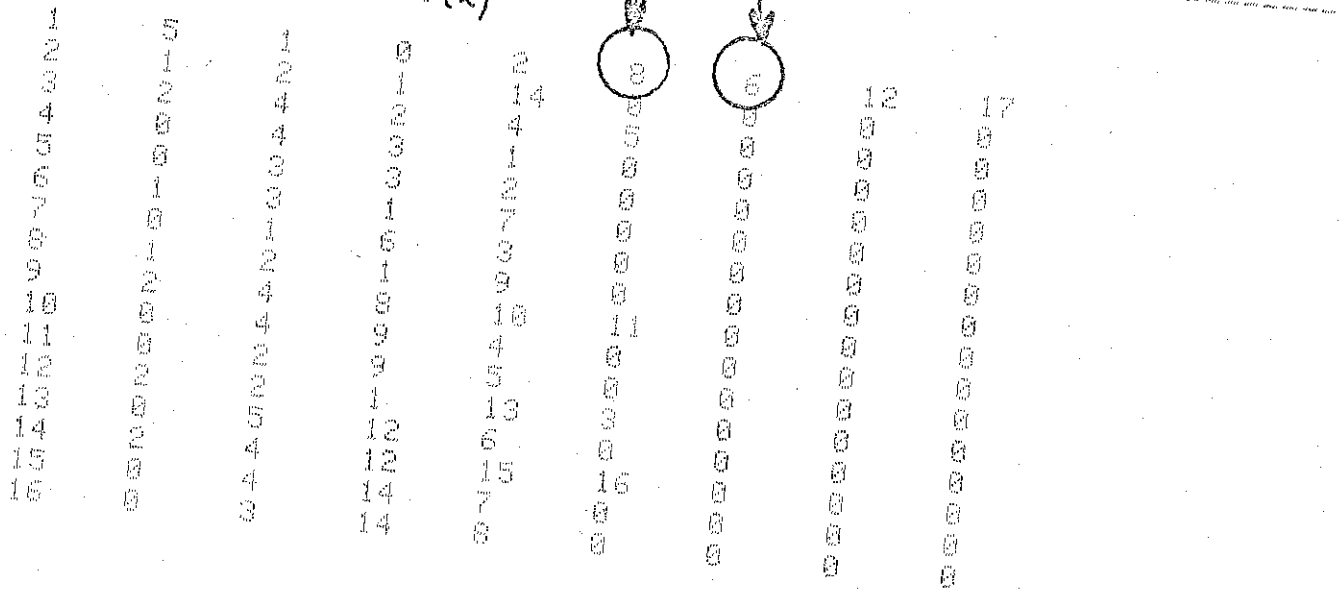
MATRIX VAN SYNT.ANAL. (1)



STRUKTURELE DESKRIPTIE: (1)

```
(SE   (RA   (NC   (DT DE)
                  (EN KEIZERLEI))
      (RN   (VE IS))
      (RA   (NC   (DT EEN)
                  (NO_ZIJSTRAAT))
      (RA   (PR VAN)
            (NC   (DT DE)
                  (EN FRANKRIJKLEI)))
      (PT .))
```

MATRIX VAN SYNT.ANAL. (2)



STRUKTURELE BESKRIPTIE: (2)

```
(SE   (RA   (NC   (DT DE)
                  (EN KEIZERLEI))
      (RA   (NC   (DT EEN)
                  (NO ZIJSTRAAT)))
      (RN   (VE IS))
      (RA   (PR VAN)
            (NC   (DT DE)
                  (EN FRANKRIJKLEI)))
      (PT .))
```

Probleem 4. Het konstrueren van boomdiagrammen via het genereren van zinnen.

Het laatste probleem dat we zullen behandelen is, hoe we gegeven een grammatika zinnen kunnen genereren en wel op een zodanige wijze dat de gegenereerde zin met zijn strukturele deskriptie in een boom-matrix terecht komt.

We konstrueren dus een programma dat de generatieve aspekten van een kontekstvrije grammatika realiseert. We noemen het systeem SYNTAXGENERATION I. Het gaat hier om een prototype dat later zal uitgebouwd worden.

Syntaxgeneration I kent twee fazen:

a) Input van syntaktische informatie:
   Hierin wordt een grammatika ingegeven, dwz voor G =⟨Vn, Vt,P,S⟩ zal er een verzameling terminale en nonterminale symbolen worden ingegeven, een reeks produkties en een axioma.

b) Genereren van zinnen.
   Dit systeem accepteert een generatieve stuurinput die aangeeft welke regel uit de grammatika moet toegepast worden en een interpretatieve stuurinput die aangeeft hoe de terminale symbolen worden geïnterpreteerd.

a. Input van syntaktische informatie

Voor dit probleem kunnen we kort zijn. We nemen met enige wijzigingen de input-eenheid over van SYNTAXTEST I (zie Steels, 1974, b). Hierin wordt eerst gevraagd naar de nonterminale en terminale symbolen. Deze symbolen worden gekodeerd volgens de strategie, dat vroeger binnengekomen symbolen lagere nummers krijgen. Dan kan men de produkties ingeven van waaruit de syntaxmatrix wordt gekonstrueerd. In tegenstelling met SYNTAXTEST I wordt geen kode ("2") toegevoegd voor nonterminale symbolen. Wel komt op de syntax-matrix hoeveel rechterelementen er aanwezig zijn in de herschrijfregel. Deze informatie wordt verkregen door een teller (T) op te stellen telkens een rechterhelftis verwerkt. We geven even voor de volledigheid het programma van de input van de syntaktische informatie. Ook voeren we een 'run' uit. De informatiematrix bevat van waar tot waar de produkties van een bepaald nonterminaal symbool gaan (in dit voorbeeld heeft SE 2 herschrijfregels , nl. 2 en 3, de produkties gaan dus van 2 tot 3.)
De syntaxmatrix geeft in de eerste kolom het aantal elementen rechts van de pijl en in de volgende kolommen die elementen zelf.

```
 90 I=3
 100 N$="AX"
 110 DISP "INPUT NONTERMINALS :";
 120 INPUT C$
 130 PRINT "NONTERMINALS:";C$
 140 PRINT
 150 N$[I,LEN(C$)+I]=C$
 160 I=I+LEN(C$)
 170 N1=((I-1)/2)
 180 DISP "INPUT TERMINALS";
 190 INPUT C$
 200 PRINT "TERMINALS:";C$
 210 PRINT
 220 N$[I,LEN(C$)+I]=C$
 230 PRINT "V(VER2. SYMBOLEN)=";N$
 240 PRINT
 250 PRINT "AANTAL NONTERMINALS:";N1
 260 PRINT
 270 A2=0
 280 PRINT "PRODUKTIES:"
 290 DISP "INPUT PRODUKTIE";
 300 T=0
 310 INPUT C$
 320 PRINT C$
 330 IF C$="↑" THEN 650
 340 C=LEN(C$)
 350 FOR Q=1 TO C STEP 3
 360 IF Q>6 THEN 430
 370 IF C$[Q,Q+1]="->" THEN 590
 380 IF C$[Q,Q+1]#"  " THEN 430
 390 I[2,A1]=I[2,A1]+1
 400 A2=I[2,A1]
 410 A3=1
 420 GOTO 590
 430 FOR J=1 TO LEN(N$)/2
 440 IF C$[Q,Q+1]=N$[(J*2)-1,J*2] THEN 470
 450 NEXT J
 460 GOTO 290
 470 IF Q>2 THEN 550
 480 A2=A2+1
 490 A3=2
 500 A1=J
 510 I[2,A1]=A2
 520 IF I[1,A1]#0 THEN 540
 530 I[1,A1]=A2
 540 GOTO 590
 550 IF J>N1 THEN 560
 560 S[I[2,A1],A3]=J
 570 T=T+1
 580 A3=A3+1
 590 NEXT Q
 600 S[I[2,A1],1]=T
 610 FOR J=A3 TO 5
 620 S[I[2,A1],J]=0
 630 NEXT J
 640 GOTO 290
```

```
1.SYNTAXIS

NONTERMINALS:SENPVP

TERMINALS:DTNOVECJ

V(VERZ. SYMBOLEN)=AXSENPVPDTNOVECJ

AANTAL NONTERMINALS: 4

PRODUKTIES:
AX -> SE
SE -> SE CJ SE
SE -> NP VP
NP -> DT NO
VP -> VE NP
↑

SYNTAXMATRIX:
 2     0     0     0     0
 2     8     2     0     0
 3     4     0     0     0
 5     6     0     0     0
 7     3     0     0     0
INFORMATIEMATRIX:
 1     2     4     5     0     0     0     0
 1     3     4     5     0     0     0     0
```

b. Genereren van zinnen.

Het is een misvatting dat een generatieve grammatika een algoritme zou zijn om zinnen te genereren. Een grammatika is enkel INPUT voor zo'n algoritme. In een grammatika zelf wordt immers geen enkele informatie gegeven hoe de grammatika nu moet leiden tot zinnen. Dit is gegeven in de teorie over die grammatika. Het is nu juist dit onderdeel dat we zullen expliciteren en formaliseren. Met andere woorden wat volgt is een algorimte dat een grammatika als input accepteert en dat zinnen genereert vanuit die grammatika.

Eerste faze: het genereren van een syntaktische struktuur.

a) strategie

We poneren een matrix B die moet dienen om de boomdiagram in op te slaan. Voor de rest maken we gebruik van de syntaxmatrix S.
Eerst wordt het axioma op B gezet, dwz het geordende n-tupel ⟨0,1,0⟩wordt opgeslagen.
Dan geven we een stuurimpuls R, waarin de regel wordt aangegeven die moet toegepast worden (dit zou ook een random mechanisme kunnen zijn). Het algoritme zoekt de linkerhelft van die regel (k). Dan zoekt het op B waar zich ergens dit nonterminaal symbool bevindt. Eens dit gevondeen geven we aan in de $\alpha 1$ van deze rij van B hoeveel elementen rechts erbij komen en duiden op αi aan waar deze elementen recht zullen terecht komen (dwz pointers naar de n-tupels van de verschillende herschrijvingen)
Dan maken we voor alle verschillende rechterelementen van de produktie een nieuwe rij in B en geven in $\alpha 1$, 0, in $\alpha 2$ het symbool zelf, (eigenlijk een pointer) in $\alpha 3$ het anker, dwz de rij waar de linkerhelft staat en de rest in onbepaald

Dit mechanisme loopt tot alle nonterminale symbolen zijn herschreven en tot er enkel terminale elementen overblijven.

b) programma

```
880 I1=1
890 I2=1
900 B[I1,1]=0
910 B[I1,2]=1
920 B[I1,2]=1
930 B[I1,3]=0
940 GOTO 1090
950 FOR J1=1 TO I2
960 IF B[J1,2]#K THEN 980
970 IF B[J1,1]=0 THEN 1000
980 NEXT J1
990 STOP
1000 B[J1,1]=S[R,1]+B[J1,1]
1010 FOR J2=1 TO S[R,1]
1020 I2=I2+1
1030 B[J1,J2+3]=I2
1040 B[I2,1]=0
1050 B[I2,2]=S[R,J2+1]
1060 B[I2,3]=J1
1070 NEXT J2
1080 RETURN
1090 INPUT R,K
1100 PRINT R;
1110 IF R=0 THEN 1140
1120 GOSUB 950
1130 GOTO 1090
```

Tweede faze: de inteppretatie van de terminale symbolen.

a) Strategie

We doorlopen B en er verschijnt voor elk terminaal symbool op het scherm "INTERPRETATIE VOOR A " en A is de naam van het te interpreteren symbool. Dan kan een interpretatie ingegeven worden. Deze komt op een string (C $) en er wordt op de vektoren P en E onthouden van waar tot waar dit woord gaat. In B (4) geven we aan om welk woord het hier gaat.

Dit mechanisme wordt doorlopen tot alle terminale symbolen werden geïnterpreteerd.

b) Programma

```
1140 V=1
1150 C=0
1160 FOR J3=1 TO I2
1170 IF B[J3,1]#0 THEN 1250
1180 DISP "INTERPRETATIE VOOR  ";N$[(B[J3,2]*2)-1,B[J3,2]*2];
1190 C=C+1
1200 P[C]=V+1
1210 INPUT C$[V+1]
1220 V=LEN(C$)
1230 E[C]=V
1240 B[J3,4]=C
1250 NEXT J3
```

Nota: Aan het programma werd ook het systeem gekoppeld dat een output in E.L.B. realiseert.

We beschrijven nu een experiment met SYNTAXGENERATION I. In een eerste faze wordt de grammatika ingegeven van waaruit de syntaxmatrix en de informatiematrix wordt gekonstrueerd. Daarna worden een aantal regels toegepast en worden de terminale symbolen geïnterpreteerd waaruit een boom in matrixvorm volgt. Vervolgens verschijnt de strukturele deskriptie in E.L.B.

Daarna doen we enkele andere experimenten in de transformatie van de bekomen syntaktische struktuur, in het herinterpeteren van dezelfde syntaktische struktuur, in het afleiden van andere strukturele deskripties met interpretaties.

Hiermee wordt een arsenaal van syntaktische technieken gedemonstreerd die gebruikelijk zijn momenteel in de linguïstiek.
De experimenten op hogere niveaus zullen in volgende rapporten worden besproken.

```
L.STEELS
SYNTAXTEST

_________
I.SYNTAXIS
_________

NONTERMINALS:SENPVP

TERMINALS:DTNOVECJ

V(VERZ. SYMBOLEN)=AXSENPVPDTNOVECJ

AANTAL NONTERMINALS: 4

PRODUKTIES:
AX -> SE
SE -> SE CJ SE
SE -> NP VP
NP -> DT NO
VP -> VE NP
↑

SYNTAXMATRIX:
 1      2      0      0      0
 3      2      0      2      0
 2      3      4      0      0
 2      5      6      0      0
 2      7      3      0      0
INFORMATIEMATRIX:
 1      2      4      5      0      0      0      0
 1      3      4      5      0      0      0      0

TOEGEPASTE REGELS
 1
 2
 3
 4
 5
 5
 4
 4
 0
BOOM IN MATRIXVORM
 1      1      1      0      2
 2      3      2      1      3      4      5
 3      2      2      2      6      7
 4      0      0      2      1
 5      2      2      0      8      9
 6      2      2      3      10     11
 7      2      2      4      3      14     15
 8      2      3      5      5      12     13
 9      2      4      5      6      16     17
 10     0      5      6      2
 11     0      6      6      3
 12     0      5      8      4
 13     0      6      8      5
 14     0      7      7      5
 15     2      8      7      18     19
 16     0      7      9      7
 17     2      3      9      20     21
 18     0      5      15     8
 19     0      6      15     9
 20     0      5      17     10
 21     0      6      17     11
```

```
STRUKTURELE DESKRIPTIE IN E.L.B.
 (AX   (SE   (SE   (NP   (DT EEN)
                         (NO OLIFANT))
                   (VP   (VE ZAG)
                         (NP   (DT EEN)
                               (NO MUIS))))
             (CJ MAAR)
             (SE   (NP   (DT DE)
                         (NO MUIS))
                   (VP   (VE BEET)
                         (NP   (DT DE)
                               (NO OLIFANT)))))))
```

We doen nu even een transformatie op de gegeven syntaktische struktuur, met name een permutatie 'een olifant zag een muis...' wordt omgezet in 'zag een olifant een muis...' met de strukturele deskriptie natuurlijk ook gepermuteerd.

Dit kan door de volgende reeks operaties uit te voeren:
Z = B(3,4)
B(3,4) = B (3,5)
B (3,5) = Z

Het resultaat geeft de volgende output

```
BOOM IN MATRIXVORM
 1     1     1     0     2
 2     3     2     1     3     4     5
 3     2     2     2     1     6
 4     0     0     2
 5     2     2     2     8     9
 6     2     3     3     10    11
 7     2     4     3     14    15
 8     2     3     5     12    13
 9     2     4     5     16    17
 10    0     5     6     2
 11    0     6     6     3
 12    0     5     8     4
 13    0     6     8     5
 14    0     7     7     6
 15    2     3     7     18    19
 16    0     7     9     7
 17    2     3     9     20    21
 18    0     5     15    8
 19    0     6     15    9
 20    0     5     17    10
 21    0     6     17    11
```

```
STRUKTURELE DESKRIPTIE IN E.L.B.
 (AX   (SE   (SE   (VP   (VE ZAG)
                         (NP   (DT EEN)
                               (NO MUIS)))
                   (NP   (DT EEN)
                         (NO OLIFANT)))
             (CJ MAAR)
             (SE   (NP   (DT DE)
                         (NO MUIS))
                   (VP   (VE BEET)
                         (NP   (DT DE)
                               (NO OLIFANT)))))
```

—

Nadat we de struktruele deskriptie terug in zijn oorspronkelijke staat hebben hersteld, gaan we nu even een andere interpretatie ingeven.

```
BOOM IN MATRIXVORM
 1     1     1     0     2
 2     0     2     1     3     4     5
 3     0     3     2     6     7
 4     0     4     2     1
 5     2     5     2     8     9
 6     0     6     3     10    11
 7     2     7     3     14    15
 8     0     8     5     12    13
 9     0     9     5     16    17
10     0     5     6     2
11     0     6     6     3
12     0     8     8     4
13     0     8     8     5
14     0     7     7     6
15     2     8     7     18    19
16     0     9     9     7
17     2     9     9     20    21
18     0     5     15    8
19     0     6     15    9
20     0     5     17    10
21     0     6     17    11
```

```
STRUKTURELE DESKRIPTIE IN E.L.B.
 (AX   (SE   (SE   (NP   (DT VELE)
                         (NO GELEERDEN))
                   (VP   (VE BEDENKEN)
                         (NP   (DT VELE)
                               (NO GELEERDHEDEN)))
             (CJ MAAR)
             (SE   (NP   (DT VELE)
                         (NO GELEERDHEDEN))
                   (VP   (VE VERGETEN)
                         (NP   (DT VELE)
                               (NO GELEERDEN)))))
```

Tot slot passen we een andere reeks regels toe, om ook dit aspekt te illustreren,en geven dan een bijkomende interpretatie van de syntaktische stru ktuur.
We moeten ook nog opmerken dat een gans andere syntaxis kan ingegeven worden als dit noodzakelijk mocht zijn.

```
TOEGEPASTE REGELS
 1
 3
 4
 5
 4
 0
BOOM IN MATRIXVORM
 1     1     1     0     2
 2     2     2     1     3     4
 3     2     3     2     5     6
 4     2     4     2     7     8
 5     0     5     3     1
 6     0     6     3     2
 7     0     7     4     3
 8     2     3     4     9     10
 9     0     5     0     4
 10    0     6     0     5

STRUKTURELE DESKRIPTIE IN E.L.B.
 (AX   (SE   (NP   (DT DE)
                   (NO KOEKOEK))
             (VP   (VE ZINGT)
                   (NP   (DT ZIJN)
                         (NO LIEDJE))))
```

Besluit

In deze paper werd het probleem gesteld van de representatie van linguïstische informatie. Nadat kort werd ingegaan op bestaande vormen van representatie, hebben we een konseptuele basis voor de definitie van representaties voorop gesteld.

Daarna werden de mogelijkheden onderzocht om de voorgestelde konsepties ook praktisch te definiëren. Met als uitgangspunt de machine representaties in de programmeertaal LISP, hebben we een matrix-representatie van list-strukturen ontwikkeld. Om de bruikbaarheid voor de linguïstische zoekaktiviteiten te demonstreren hebben we een aantal programma's gekonstrueerd om specifieke linguïstische taken op te knappen. Zo werd naast een automatische syntaktische analyse, ook een genererend systeem opgezet en een mogelijkheid om transformaties uit te voeren op een gegeven struktuur.

Luc Steels
Antwerpen 1974

Bibliografie

BEKIĆ, H. (1973) On the formal definition of programming languages, p. 297, e.v. in Itsfeldt, W. (ed), International Computing Symposium, Bonn.

BERKELEY, E.C. &.D.G. Bobrow, (ed) (1967) The programming language LISP: Its operation and application. The M.I.T. Press. M.I.T. Cambridge, Mass.

BOBROW, O & RAPHAEL B. (1974) New Programming Languages for artificial intelligence research. ACM. computer surveys, vol 6, nr 3. sept 1974.

BRAINERD, B. (1971) Introduction to the mathematics of language study. Prentice Hall, Engledwood Cliffs, New Jersey.

CRESWELL, M.J. (1973) Logics and languages. Methuen & co. Ltd. London.

CHOMSKY, N. (1963) On formal properties of grammarx. in Luce, R. et.al. (ed) Handbook of Mathematical Psychology, Vol II; New York, Wiley. pp 323 - 418.

CHOMSKY, N. (1965) Aspects of the theory of syntax. The M.I.T. Press. M.I.T. Cambridge, mass.

ERNST, G. & A. NEWELL (1969) GPS: A case study in Generality and Problem solving. Academic Press, N.Y. & London.

FEIGENBAUM, E. & J. FELDMAN (1963) (eds.) Computers & Thought. McGraw Hill NewYork.

FINDLER, N.V. & B. MELTZER. (1971) Artificial Intelligence and Heuristic Programming. Edinburgh. University Press.

FOX, L. (ed) L966. Advances in programming and non-numerical computation. Pergamon press. Oxford.

PARTEE, H.B. (1974) Logic and semantics. In: Procedings of the I.S. Summerschool for mathematical and computational linguistics. Pisa (forthcoming).

HEWITT, C. (1973) Procedural Semantics: Models of procedures and the teaching of procedures. in Rustin, R. (ed).

LEE, J.A.N. (1973) Computer semantics. Studies of Algorithms, Processors and Languages. Van Nostrand Reinhold Cy. N.Y.

LINDSAY, R.K. (1963) Inferential Memory at the basis of machines which understand natural language. in Feigenbaum & Feldman (1963), pag 217, e.v. .

McCARTHY, J. et.al. (1961) LISP 1.5. Programmer's mannal. The M.I.T. Press, M.I.T. Cambridge, Massachusetts.

QUILLIAN, R. (1967) Semantic Memory. in: Minsky, M. Semantic Information Processing. M.I.T. Press, M.I.T. Cambridge, Mass.

RUSTIN, R. (1973) Natural language Processing. Algorithmic Press, N.Y.

SANDEWALL, E. (1974) The use of the predicate calculus for semantic data bases. in: Proceedings I.S. for mathematical and computational linguisticss, Pisa. (forthcoming).

STEELS, L. (1974 a) Seminaries in procedurële semantiek. Universiteit Antwerpen (U.I.A.) Afd. Ned. Taalkunde. Interne nota. december 1974.

STEELS, L. (1974 b) SYNTAXTEST I (a deterministc conetext free grammar parser) & SYNTAXTEST II (a nondeterministic transition network parser). Universiteit Antwerpen (U.I.A.) Afd. Nederlandse taalkunde. Technisch rapport. LS/1. December 1974.

Türing, A. (1963) Computing machinery and intelligence in Feigenbaum and Feldman. p. 11.-35.

VELDE, R.G. van de (1972) Inleiding tot de strukturele morfo-syntaxis. Talen en Cultuur (ed. M. De sGrève , F. van Passel), 10, Ontwikkeling, Antwerpen, Labor, Brussel.

WEISSMAN, (1967) LISP 1.5. Primer. Dickenson Pub. Cy. Belmont, California, 1967.

WINOGRAD (1974) Five lectures on artificial intelligence. Tokyo, Electrotechnical laboratory.

WINOGRAD, T. (1974) Current topics in Computational semantics. in Procedings I.S. computational & mathematical linguistics Pisa. (forthcoming).

WOODWARD, P.M. (1966) Listprogramming, p. 29 e.v. in Fox (1966)