Een attribuutgrammatica voor de substantiefgroep

Maarten Pennings
217161

5 januari 1990

Inhoudsopgave

1 Inleiding **2**
- 1.1 Overzicht . . . 2
- 1.2 Notatie . . . 2
- 1.3 Typografie . . . 3
- 1.4 Verantwoording . . . 3

2 De grammatica **4**
- 2.1 De substantiefgroep, de basis . . . 4
- 2.2 Toeters en bellen . . . 6
- 2.3 De substantiefgroep: recursie . . . 7
- 2.4 Het resultaat . . . 8
  - 2.4.1 De domeinen . . . 8
  - 2.4.2 De non-terminals . . . 8
  - 2.4.3 De terminals . . . 9
  - 2.4.4 De pseudo-terminals . . . 9
  - 2.4.5 De produktieregels . . . 9

3 Ambiguïteit **11**
- 3.1 De probleemgevallen . . . 11
- 3.2 Voorbeelden . . . 12
- 3.3 Synoniemen probleem . . . 12
- 3.4 Ambigue constructies ontleden . . . 13
- 3.5 De uitvoer . . . 13
- 3.6 Een extra attribuut . . . 14
  - 3.6.1 De domeinen . . . 14
  - 3.6.2 De non-terminals . . . 15
  - 3.6.3 De (pseudo-)terminals . . . 15
  - 3.6.4 De produktieregels . . . 15
- 3.7 Herstel . . . 16

4 Datastructuren **17**
- 4.1 Rauwe tekst . . . 17
- 4.2 De woordtypes . . . 17
- 4.3 Behandelde tekst . . . 18
- 4.4 Rijtje zinnen . . . 18
- 4.5 De data module . . . 19

5 De lexical scanner **20**
- 5.1 Taakomschrijving . . . 20
- 5.2 De lexicon . . . 20
  - 5.2.1 De invoergrammatica . . . 21
  - 5.2.2 De datatypes . . . 21

5.2.3 Toevoegen . . . 22
5.2.4 Opzoeken . . . 23
5.2.5 De lexicon module . . . 23
5.3 De scanner . . . 24
5.3.1 De parser voor de lexicon . . . 24
5.3.2 Woordtyperen . . . 25

6 De parser **26**
6.1 De datastructuren . . . 26
6.2 Recursive-descent parser . . . 27
6.3 Back-track parser . . . 30

7 Userinterface **33**
7.1 Eenvoud . . . 33
7.2 De LOG-file . . . 33
7.3 Ontleed schema . . . 35
7.3.1 Woorden invullen . . . 35
7.3.2 Specificatie . . . 35
7.3.3 De box . . . 36
7.3.4 Het samenvoegen van boxen . . . 37
7.3.5 Structuur naar box . . . 39
7.3.6 Afdrukken box . . . 40
7.4 Het programma . . . 41

A Module Data **42**

B Module Uitvoer **45**

C Module Lexicon **47**

D Module Scanner **49**

E Module Parser **51**

F Module User **56**

G Het programma **61**

Hoofdstuk 1

Inleiding

In dit verslag behandelen we het ontwerp van een taalherkenner. Dit is een algoritme dat de vraag beantwoordt: is deze rij woorden Nederlands of niet? We volgen het idee van Hugo Brandt Corstius in [Brandt Corstius] pagina 154 - 159. Hij geeft daar een contextvrije grammatica die een onbeperkt aantal Nederlandse zinnen voortbrengt. De grootste tekortkoming van deze grammatica valt snel op: de substantiefgroep is zeer primitief: geen lidwoorden (de of het?) en geen adjectieven (met of zonder e?).

Hebben we, voor deze contextcondities, de veel krachtigere — maar ook complexere — transformationele grammatica's nodig? De contextvrije grammatica's zijn in ieder geval ontoereikend. Met dit project willen we kijken welke resultaten we kunnen bereiken met attribuut grammatica's. We beperken ons hierbij tot de substantiefgroep, zonder beperkende bijzinnen, om niet meteen te verzanden in de overweldigende verscheidenheid van een natuurlijke taal. Tijdens dit project heeft er zich een opvolger gemeld die van plan is de beperkende bijzinnen toe te voegen.

1.1 Overzicht

Na de globale inleiding in dit hoofdstuk, behandelen we in hoofdstuk 2 het taalkundige gedeelte van de taalherkenner. We ontwerpen daar een contextvrije grammatica die (een deelverzameling van) de Nederlandse substantiefgroep voortbrengt. Om de verbuiging van het adjectief en de keuze van het lidwoord in goede banen te leiden voeren we attributen in.

In hoofdstuk 3 nemen we de ontwikkelde grammatica in ogenschouw. Wat te verwachten valt van een grammatica voor een natuurlijke taal wordt bewaarheid: ambiguïteit. En wel op verschillende manieren. Er zijn woorden(-tekens) met meerdere betekenissen, er zijn woord(-tekens) die tot verschillende woordtypes behoren, en de grammatica zelf is al ambigu.

Met hoofdstuk 4 begint de computerkunde. De eerste aanzet vormt de keuze van enkele datastructuren. Vervolgens ontwerpen de lexical scanner en zijn lexicon (hoofdstuk 5) en de parser (hoofdstuk 6). Ten slotte wordt in hoofdstuk 7 nog enige aandacht geschonken aan een — enigszins nette — uitvoer van het programma.

De appendices die daarna volgen bevatten de complete source van alle modules waaruit het programma bestaat gevolgd door een literatuuropgave.

1.2 Notatie

Gewoontegetrouw noteren we de universele kwantificatie op de volgende manier

$$(\mathbf{A}l : D : E)$$

waarbij $\mathbf{A}$ de kwantor is, l een lijst van gebonden variabelen, D een predikaat en E de gekwantificeerde expressie is. Zowel D als E zullen in het algemeen variabelen van l bevatten. Predikaat D beperkt het domein van de gebonden variabelen en E is goed gedefinieerd voor waarde in dat

bereik. Analoog gebruiken we de kwantor **E** voor de existentiële kwantificatie. Deze notatie komt van [Dijkstra]

1.3 Typografie

De gewone (brood-) tekst wordt gezet in het "Roman" font. De terminals van de grammatica's en de programmafragmenten in het "`Typewriter`" font en de attributen in het "Sans serif" font.

1.4 Verantwoording

Dit werkstuk is het resultaat van de opdracht bij het vak computer taalkunde. Het is door mij gemaakt in overleg met Alfo Melisse, mijn "opvolger", en drs. P.C. uit den Bogaart, de docent van het vak computer taalkunde aan de vakgroep toegepaste taalkunde van de Technische Universiteit Eindhoven.

Hoofdstuk 2

De grammatica

Zoals we in de inleiding al gezegd hebben beperken we ons tot de substantiefgroep. Enerzijds is dit taalkundige object te ingewikkeld voor een contextvrije grammatica, anderzijds is het een goed isoleerbaar zinsdeel waarvan de complexiteit naar behoeve kan worden uitgebreid door achtereenvolgens adjectieven, samenstellingen, voorzetselconstituenten en zelfs beperkende bijzinnen toe te laten.

In dit hoofdstuk stellen we een attribuut grammatica op die naar onze smaak voldoende complex is. De produktieregels die we introduceren berusten allemaal op informatie uit [Geerts].

2.1 De substantiefgroep, de basis

Wat is een substantiefgroep? De basis voor dit zinsdeel vormt de kerngroep die intuïtief de volgende produktieregel heeft

KernGroep $\longrightarrow$ Lidwoord Substantief

Slaan we [Geerts] er op na, dan blijkt dit natuurlijk een te simplistische voorstelling van zaken, maar voor ons is het vooralsnog voldoende om een paar attributen te introduceren. Want

`de kikker` is goed, maar `de kusje` niet.
`de kikkers` en `de kusjes` zijn daarentegen wel beide goed.

We zien uit dit voorbeeld al dat we zowel de substantieven als de lidwoorden moeten partitioneren: de substantieven in de-woorden, het-woorden en mv-woorden en, heel voor de hand liggend, de lidwoorden in de-bepalingen, het-bepalingen en een-bepalingen. Raadplegen we [Geerts] dan vinden we achtereenvolgens (de getallen tussen haakjes geven het paginanummer uit [Geerts] aan):

Lidwoord({de-bepaling, het-bepaling, een-bepaling**})** (zie 112)

Lidwoord(de-bepaling)	$\longrightarrow$	`de`
Lidwoord(het-bepaling)	$\longrightarrow$	`het`
Lidwoord(een-bepaling)	$\longrightarrow$	`een`

en

Substantief({de-woord, het-woord,mv-woord**})** (zie 39)

Substantief(de-woord)	$\longrightarrow$	`ridder` | `fee` | `heks` | `pad` | `ezel` | `kikker` | `bos` | `bal` | `stad` | `vesting` | `...`
Substantief(het-woord)	$\longrightarrow$	`meisje` | `zusje` | `poesje` | `kalf` | `kusje` | `bos` | `pad` | `bal` | `kasteel` | `...`
Substantief(mv-woord)	$\longrightarrow$	`ridders` | `feeen` | `heksen` | `padden` | `ezels` | `kikkers` | `bossen` | `ballen` | `steden` | `vestingen` | `...` | `meisjes` | `zusjes` | `poesjes` | `kalveren` | `kusjes` | `bossen` | `paden` | `bals` | `kastelen` | `...`

De context-eis van de KernGroep wordt dan

KernGroep (zie 39-52)

KernGroep $\longrightarrow$ Lidwoord(l) Substantief(s)
(s = mv-woord $\wedge$ l = de-bepaling)$\vee$
(s = de-woord $\wedge$ (l = de-bepaling $\vee$ l = een-bepaling))$\vee$
(s = het-woord $\wedge$ (l = het-bepaling $\vee$ l = een-bepaling))

Alle tot dusver gebruikte attributen, en alle die gaan volgen, zijn van het type "synthesized" ofwel "bottom-up". Dit betekent dat bij een produktieregel $A \longrightarrow B$ de vorm van B restricties oplegt aan die van A. Omgekeerd is ook mogelijk. Dat staat bekend onder "inherrited" of "top-down".

Natuurlijk willen we ook adjectieven tussen het lidwoord en het substantief. Hierbij ontstaan vervoegingsproblemen als in

`een onneembare vesting` en `een onneembaar kasteel`

die in [Geerts] op pagina 322 worden opgelost. Maar op pagina 702 blijkt er zelfs een voorkeursvolgorde te zijn voor adjectieven binnen één groep! Vergelijk

```
gouden mooie ronde koets
gouden ronde mooie koets
mooie gouden ronde koets
mooie ronde gouden koets
ronde gouden mooie koets
ronde mooie gouden koets
```

en merk op dat alleen de vierde zin "lekker loopt". Direct voor het substantief komen namelijk bij voorkeur stof-adjectieven (`gouden`), geografische-adjectieven (`Urker`), of een van de uitzonderingen `rechter`, `linker`, `volbloed`, `halfbloed`, `gratis`, ... Deze adjectieven hebben de hoogste prioriteit (prioriteit 1). Op de tweede plaats komen de vorm- en kleur adjectieven (`ronde`) met prioriteit 2 en op de derde plaats de overige (zoals `mooie`).

Dwars door deze driedeling loopt, zoals reeds opgemerkt, de verbuiging: met-e of zonder-e. Zo krijgen we dus 6 produktieregels:

Adjectief({met-e, zonder-e},{1, 2, 3}) (zie 322,702)

Adjectief(met-e,3)	$\longrightarrow$	`lekkere` | `volslanke` | `toekomstige` | `mooie` | `dwaze` | `boze` | `gave` | `laffe` | ...
Adjectief(zonder-e,3)	$\longrightarrow$	`lekker` | `volslank` | `toekomstig` | `mooi` | `dwaas` | `boos` | `gaaf` | `laf` | ...
Adjectief(met-e,2)	$\longrightarrow$	`vierkante` | `groene` | `ronde` | `rode` | ...
Adjectief(zonder-e,2)	$\longrightarrow$	`vierkant` | `groen` | `rond` | `rood` | ...
Adjectief(met-e,1)	$\longrightarrow$	`Amsterdamse` | `Belgische` | ...
Adjectief(zonder-e,1)	$\longrightarrow$	`Amsterdams` | `Belgisch` | ...

"Helaas" blijken er ook nog adjectieven te bestaan die niet verbogen worden, zoals de stof-adjectieven en de al boven genoemde uitzonderingen. Daarnaast zijn onverbogen de adjectieven op -a, -o, -e, -en, -é, -i, -y. We moeten dus het attribuut van Adjectief wijzigen, en drie nieuwe produktieregels toevoegen:

Adjectief({met-e, zonder-e, onverbogen},{1, 2, 3}) (zie 322,702)

Adjectief(onverbogen,3)	$\longrightarrow$	`albino` | `timide` | `dronken` | `privé` | `sexy` | ...
Adjectief(onverbogen,2)	$\longrightarrow$	`lila` | `beige` | `roze` | `kaki` | ...
Adjectief(onverbogen,1)	$\longrightarrow$	`gouden` | `stenen` | `mica` | ... | `Edammer` | `Groninger` | `Urker` | ... | `rechter` | `linker` | `volbloed` | `gratis` | ...

We vormen een adjectiefgroep door simpelweg adjectieven achter elkaar te plaatsen. Behalve de bovengenoemde prioriteiten van adjectieven binnen zo'n groep, moeten alle adjectieven ook nog gelijk verbogen zijn. Deze contextcondities worden bij de AdjectiefGroep gecontroleerd:

AdjectiefGroep({met-e, zonder-e, onverbogen}) (zie 702)

$$\begin{aligned}
&\text{AdjectiefGroep}(\textsf{met-e}) \longrightarrow \{\ \text{Adjectief}(v_i, n_i)\ \}_{0\le i<N}\\
&\quad (\mathbf{A}i : 1 \le i < N : n_{i-1} \ge n_i) \wedge\\
&\quad (\mathbf{A}i : 0 \le i < N : v_i \in \{\textsf{met-e}, \textsf{onverbogen}\}) \wedge\\
&\quad (\mathbf{E}i : 0 \le i < N : v_i = \textsf{met-e})\\
&\text{AdjectiefGroep}(\textsf{zonder-e}) \longrightarrow \{\ \text{Adjectief}(v_i, n_i)\ \}_{0\le i<N}\\
&\quad (\mathbf{A}i : 1 \le i < N : n_{i-1} \ge n_i) \wedge\\
&\quad (\mathbf{A}i : 0 \le i < N : v_i \in \{\textsf{zonder-e}, \textsf{onverbogen}\}) \wedge\\
&\quad (\mathbf{E}i : 0 \le i < N : v_i = \textsf{zonder-e})\\
&\text{AdjectiefGroep}(\textsf{onverbogen}) \longrightarrow \{\ \text{Adjectief}(v_i, n_i)\ \}_{0\le i<N}\\
&\quad (\mathbf{A}i : 1 \le i < N : n_{i-1} \ge n_i) \wedge\\
&\quad (\mathbf{A}i : 0 \le i < N : v_i = \textsf{onverbogen})
\end{aligned}$$

De KernGroep controleert dan op de juiste verbuiging:

KernGroep (zie 325-326)

$$\begin{aligned}
&\text{KernGroep} \longrightarrow \text{Lidwoord}(l)\ \text{AdjectiefGroep}(a)\ \text{Substantief}(s)\\
&\quad (s = \textsf{mv-woord} \wedge a \in \{\textsf{met-e}, \textsf{onverbogen}\} \wedge l = \textsf{de-bepaling}) \vee\\
&\quad (s = \textsf{de-woord} \wedge a \in \{\textsf{met-e}, \textsf{onverbogen}\} \wedge l = \textsf{de-bepaling}) \vee\\
&\quad (s = \textsf{de-woord} \wedge a \in \{\textsf{met-e}, \textsf{onverbogen}\} \wedge l = \textsf{een-bepaling}) \vee\\
&\quad (s = \textsf{het-woord} \wedge a \in \{\textsf{met-e}, \textsf{onverbogen}\} \wedge l = \textsf{het-bepaling}) \vee\\
&\quad (s = \textsf{het-woord} \wedge a \in \{\textsf{zonder-e}, \textsf{onverbogen}\} \wedge l = \textsf{een-bepaling})
\end{aligned}$$

Merk op dat de adjectiefgroep óók de lege string (ϵ) voortbrengt onder het attribuut onverbogen. En dat past mooi. We willen immers niet altijd een adjectief, en de lege adjectiefgroep legt zo geen beperkingen op aan de te vormen kerngroep.

Deze produktieregels vormen de basis van de substantiefgroep. In de volgende paragrafen zullen we produktieregels toevoegen om de substantiefgroep realistischer te maken.

2.2 Toeters en bellen

In deze paragraaf gaan we de kerngroep wat verder aankleden. Geen wezenlijke, of beter ingewikkelde, constructies, maar wel constructies die de hem een wat realistischer aanzien geven. Zo breiden we de AdjectiefGroep uit met een optioneel bijwoord

Bijwoord (zie 748)

Bijwoord ⟶ `nogal` | `tamelijk` | `betrekkelijk` | `vrij` | `heel` | `zeer` |
`uitermate` | `vreselijk` | `ontzettend` | `ongemeen` | `...`

en de KernGroep met een optioneel telwoord.

Telwoord (zie 289-291)

Telwoord ⟶ `twee` | `drie` | `vier` | `...` |
`beide` | `vele` | `enige` | `...`

Dit genereert dan kerngroepen als

`de ontzettend mooie prinses` en `de zeven dwergen`

Het zou leuk zijn als we Sneeuwwitje gewoon Sneeuwwitje zouden kunnen noemen in plaats van `de ontzettend mooie prinses`. Daarom voegen we de regel

Naam ⟶ `Assepoester` | `Roodkapje` | `Sneeuwwitje` | `...`

toe. Deze syntactische categorie is feitelijk een tweede basis voor de substantiefgroep. Hier is het punt aangekomen dat expliciet te maken:

SubstantiefGroep ⟶ KernGroep
SubstantiefGroep ⟶ Naam

Het chronische gebruik van `de` en `het` gaat op den duur natuurlijk te vervelen. Gelukkig biedt ook hier [Geerts] een oplossing: (bezittelijke) voornaamwoorden kunnen de functie van het lidwoord overnemen.

Voornaamwoord({de-bepaling, het-bepaling, een-bepaling**})** (zie 216)

Voornaamwoord(de-bepaling)	⟶	`deze` | `die`
Voornaamwoord(het-bepaling)	⟶	`dit` | `dat`

Bij de bezittelijke voornaamwoorden treedt er een probleem op: `mijn` kan zowel als de-bepaling als als het-bepaling fungeren. We voeren daarom een extra attribuut de/het-bepaling in:

BezittelijkVNW({de-bepaling, het-bepaling, een-bepaling, de/het-bepaling**})** (zie 197)

BezittelijkVNW(de-bepaling)	⟶	`onze`
BezittelijkVNW(het-bepaling)	⟶	`ons`
BezittelijkVNW(de/het-bepaling)	⟶	`mijn` | `jouw` | `je` | `uw` | `zijn` | `haar` | `jullie` | `hun`

zodat we nu over

`haar zeven dwergen`

kunnen spreken. Merk op dat het bezittelijk voornaamwoord `uw` zowel in het enkel- als in het meervoud voorkomt, maar dat dat verschil voor ons niet van belang is.

2.3 De substantiefgroep: recursie

We hebben nu de basis van de substantiefgroep gedefinieerd. In deze paragraaf wordt de structuur van onze grammatica een klasse interessanter: recursie doet zijn intrede. We komen dat op twee plaatsen tegen, bij nevenschikking van twee substantiefgroepen en bij de voorzetselconstituent.

Allereerst de nevenschikking. Het zou leuk zijn de acht hoofdpersonen uit de vorige paragraaf in één sprookje op te laten treden. We voegen dus toe

SubstantiefGroep	⟶	SubstantiefGroep Voegwoord SubstantiefGroep
Voegwoord	⟶	`en`

zodat voortaan ook

`Sneeuwwitje en de zeven dwergen`

voortgebracht wordt. Helemaal gelukkig zijn we hier nog niet mee. Het volgende voorbeeld blinkt niet uit in stijl.

`Assepoester en haar geniepige stiefmoeder en die lelijke zusjes`

Vandaar de voorzetselconstituenten. Zij staan "stijlvollere" constructies toe als

`het glazen muiltje in de hand van die knappe prins` of
`Assepoester met haar geniepige stiefmoeder en die lelijke zusjes`

Een voorzetselconstituent is gewoon

Voorzetselconstituent	⟶	Voorzetsel SubstantiefGroep

Hier zit echter één addertje onder het gras (pagina 628 [Geerts]): het voorzetsel `tussen` verwacht een meervoudige substantiefgroep.

`het zwaard tussen zijn ribben` of
`de heks tussen Hans en Grietje`

We splitsen dus de voorzetsels in twee groepen:

Voorzetsel({enkelvoud, meervoud**})** (zie 624-630)

Voorzetsel(meervoud)	⟶	`tussen`
Voorzetsel(enkelvoud)	⟶	`aan` | `achter` | `bij` | `boven` | `buiten` | `door` | `in` | `langs` | `met` | `na` | `naar` | `naast` | `nabij` | `om` | `onder` | `op` | `over` | `rond` | `tegen` | `tegenover` | `tijdens` | `tot` | `uit` | `van` | `voor` | `zonder` | `...`

en voegen attributen toe aan de KernGroep

KernGroep({meervoud, enkelvoud}) (zie 694)

KernGroep(meervoud) $\longrightarrow$ Voorbepaling(v) [Telwoord] AdjectiefGroep(a) Substantief(s)
$s = \text{mv-woord} \wedge a \in \{\text{met-e, onverbogen}\} \wedge v \in \{\text{de-bepaling, de/het-bepaling}\}$

KernGroep(enkelvoud) $\longrightarrow$ Voorbepaling(v) AdjectiefGroep(a) Substantief(s)
$(s = \text{de-woord} \wedge a \in \{\text{met-e, onverbogen}\} \wedge v \in \{\text{de-bepaling, de/het-bepaling}\}) \vee$
$(s = \text{het-woord} \wedge a \in \{\text{met-e, onverbogen}\} \wedge v \in \{\text{het-bepaling, de/het-bepaling}\}) \vee$
$(s = \text{de-woord} \wedge a \in \{\text{met-e, onverbogen}\} \wedge v = \text{een-bepaling}) \vee$
$(s = \text{het-woord} \wedge a \in \{\text{zonder-e, onverbogen}\} \wedge v = \text{een-bepaling})$

en vervolgens aan de substantiefgroep

SubstantiefGroep(meervoud) $\longrightarrow$ SubstantiefGroep(s_1) Voegwoord SubstantiefGroep(s_2)
SubstantiefGroep(s) $\longrightarrow$ SubstantiefGroep(s) Voorzetselconstituent
SubstantiefGroep(s) $\longrightarrow$ KernGroep(s)
SubstantiefGroep(enkelvoud) $\longrightarrow$ Naam

zodat ook bij de

Voorzetselconstituent (zie 712)

Voorzetselconstituent $\longrightarrow$ Voorzetsel(v) SubstantiefGroep(s)
$v = \text{meervoud} \Rightarrow s = \text{meervoud}$

de context gecontroleerd kan worden.

2.4 Het resultaat

In deze paragraaf geven we een formele definitie van de ontworpen attribuut grammatica G_s met $G_s = (N_s, T_s, P_s, \text{SubstantiefGroep})$. Deze grammatica is de in de vorige paragraaf ontwikkelde grammatica zij het dat hij op enkele punten nog is bijgeschaafd.

2.4.1 De domeinen

We beginnen met de gebruikte domeinen.

$$
\begin{aligned}
Bepaling &= \{\text{de-bepaling, het-bepaling, de/het-bepaling, een-bepaling}\} \\
Verbuiging &= \{\text{met-e, zonder-e, onverbogen}\} \\
Prioriteit &= \{1, 2, 3\} \\
Woord &= \{\text{de-woord, het-woord, mv-woord}\} \\
Getal &= \{\text{meervoud, enkelvoud}\}
\end{aligned}
$$

2.4.2 De non-terminals

De verzameling N_s van nonterminals wordt dan gedefinieerd als:

$N_s =$ { SubstantiefGroep($Getal$), Voorzetselconstituent
, KernGroep($Getal$), AdjectiefGroep($Verbuiging$)
, Voorzetsel($Getal$), Substantief($Woord$)
, Adjectief($Verbuiging$, $Prioriteit$)
, Voorbepaling($Bepaling$), Bijwoord, Telwoord
, DeWoord, HetWoord, MVwoord, Naam
, EVvoorzetsel, MVvoorzetsel, Voegwoord
, AdjectiefMetE3, AdjectiefZonderE3, AdjectiefOnverbogen3
, AdjectiefMetE2, AdjectiefZonderE2, AdjectiefOnverbogen2
, AdjectiefMetE1, AdjectiefZonderE1, AdjectiefOnverbogen1
, DeBepaling, HetBepaling, DeHetBepaling, EenBepaling
}

2.4.3 De terminals

Een nog volledig onbesproken grootheid: de terminals. Ook nu maken we ons ervan af: we negeren accenten en trema's. De verzameling T_s van terminals definiëren we als:

$T_s =$ { a, b, ..., z
, A, B, ..., Z
}

2.4.4 De pseudo-terminals

Wat zijn pseudo-terminals? Pseudo-terminals zijn non-terminals die gezien hun eenvoudige structuur (zoals bijvoorbeeld identifiers bij programmeertalen) niet van produktieregels worden voorzien. In de praktijk betekent dit vaak dat niet de parser maar de lexical scanner deze nonterminals "parset". Wij gebruiken de volgende pseudo-terminals:

$\mathcal{L}$(Bijwoord)	=	{nogal, tamelijk, betrekkelijk, vrij, ...}
$\mathcal{L}$(Telwoord)	=	{twee, drie, ... , beide, vele, ...}
$\mathcal{L}$(DeWoord)	=	{ridder, fee, heks, pad, ezel, ...}
$\mathcal{L}$(HetWoord)	=	{meisje, zusje, kalf, bos, pad, kasteel, ...}
$\mathcal{L}$(MVwoord)	=	{ridders, zusjes, padden, ezels, kastelen, ...}
$\mathcal{L}$(Naam)	=	{Assepoester, Roodkapje, Sneeuwwitje, ...}
$\mathcal{L}$(EVvoorzetsel)	=	{aan, achter, bij, boven, buiten, door, ...}
$\mathcal{L}$(MVvoorzetsel)	=	{tussen}
$\mathcal{L}$(Voegwoord)	=	{en}
$\mathcal{L}$(AdjectiefMetE3)	=	{lekkere, volslanke, toekomstige, ...}
$\mathcal{L}$(AdjectiefZonderE3)	=	{lekker, volslank, toekomstig, ...}
$\mathcal{L}$(AdjectiefOnverbogen3)	=	{albino, timide, dronken, privé, sexy, ...}
$\mathcal{L}$(AdjectiefMetE2)	=	{vierkante, groene, rode, ...}
$\mathcal{L}$(AdjectiefZonderE2)	=	{vierkant, groen, rood, ...}
$\mathcal{L}$(AdjectiefOnverbogen2)	=	{lila, beige, roze, kaki, ...}
$\mathcal{L}$(AdjectiefMetE1)	=	{Amsterdamse, Belgische, ...}
$\mathcal{L}$(AdjectiefZonderE1)	=	{Amsterdams, Belgisch, ...}
$\mathcal{L}$(AdjectiefOnverbogen1)	=	{gouden, stenen, Urker, linker, gratis, ...}
$\mathcal{L}$(DeBepaling)	=	{de, deze, die, onze}
$\mathcal{L}$(HetBepaling)	=	{het, dit, dat, ons}
$\mathcal{L}$(DeHetBepaling)	=	{mijn, jouw, je, uw, zijn, haar, jullie, hun}
$\mathcal{L}$(EenBepaling)	=	{een}

2.4.5 De produktieregels

De verzameling P_s van produktieregels is alsvolgt gedefinieerd:

$P_s =$

{SubstantiefGroep(meervoud) $\longrightarrow$ SubstantiefGroep(s_1) Voegwoord SubstantiefGroep(s_2)

, SubstantiefGroep(s) $\longrightarrow$ SubstantiefGroep(s) Voorzetselconstituent

, SubstantiefGroep(s) $\longrightarrow$ KernGroep(s)

, SubstantiefGroep(enkelvoud) $\longrightarrow$ Naam

, Voorzetselconstituent $\longrightarrow$ Voorzetsel(v) SubstantiefGroep(s)
$v = \text{meervoud} \Rightarrow s = \text{meervoud}$

, KernGroep(meervoud) $\longrightarrow$ Voorbepaling(v) [Telwoord] AdjectiefGroep(a) Substantief(s)
$s = \text{mv-woord} \wedge a \in \{\text{met-e, onverbogen}\} \wedge v \in \{\text{de-bepaling, de/het-bepaling}\}$

, KernGroep(enkelvoud) $\longrightarrow$ Voorbepaling(v) AdjectiefGroep(a) Substantief(s)
$(s = \text{de-woord} \wedge a \in \{\text{met-e, onverbogen}\} \wedge v \in \{\text{de-bepaling, de/het-bepaling}\}) \vee$
$(s = \text{het-woord} \wedge a \in \{\text{met-e, onverbogen}\} \wedge v \in \{\text{het-bepaling, de/het-bepaling}\}) \vee$
$(s = \text{de-woord}, \wedge a \in \{\text{met-e, onverbogen}\} \wedge v = \text{een-bepaling}) \vee$
$(s = \text{het-woord} \wedge a \in \{\text{zonder-e, onverbogen}\} \wedge v = \text{een-bepaling})$

$$
\begin{array}{lll}
, \text{AdjectiefGroep}(\textsf{met-e}) & \longrightarrow & [\text{ Bijwoord }]\ \{\text{ Adjectief}(v_i, n_i)\ \}_{0 \le i < N} \\
& & (\mathbf{A} i : 1 \le i < N : n_{i-1} \ge n_i) \wedge \\
& & (\mathbf{A} i : 0 \le i < N : v_i \in \{\textsf{met-e}, \textsf{onverbogen}\}) \wedge \\
& & (\mathbf{E} i : 0 \le i < N : v_i = \textsf{met-e}) \\
, \text{AdjectiefGroep}(\textsf{zonder-e}) & \longrightarrow & [\text{ Bijwoord }]\ \{\text{ Adjectief}(v_i, n_i)\ \}_{0 \le i < N} \\
& & (\mathbf{A} i : 1 \le i < N : n_{i-1} \ge n_i) \wedge \\
& & (\mathbf{A} i : 0 \le i < N : v_i \in \{\textsf{zonder-e}, \textsf{onverbogen}\}) \wedge \\
& & (\mathbf{E} i : 0 \le i < N : v_i = \textsf{zonder-e}) \\
, \text{AdjectiefGroep}(\textsf{onverbogen}) & \longrightarrow & \{\text{Bijwoord }\}_{0 \le i < M}\ \{\text{ Adjectief}(v_i, n_i)\ \}_{0 \le i < N} \\
& & (\mathbf{A} i : 1 \le i < N : n_{i-1} \ge n_i) \wedge \\
& & (\mathbf{A} i : 0 \le i < N : v_i \in \{\textsf{onverbogen}\}) \wedge \\
& & M = 0 \vee (M = 1 \wedge N > 0) \\
, \text{Voorzetsel}(\textsf{meervoud}) & \longrightarrow & \text{MVvoorzetsel} \\
, \text{Voorzetsel}(\textsf{enkelvoud}) & \longrightarrow & \text{EVvoorzetsel} \\
, \text{Substantief}(\textsf{de-woord}) & \longrightarrow & \text{DeWoord} \\
, \text{Substantief}(\textsf{het-woord}) & \longrightarrow & \text{HetWoord} \\
, \text{Substantief}(\textsf{mv-woord}) & \longrightarrow & \text{MVwoord} \\
, \text{Adjectief}(\textsf{met-e},3) & \longrightarrow & \text{AdjectiefMetE3} \\
, \text{Adjectief}(\textsf{zonder-e},3) & \longrightarrow & \text{AdjectiefZonderE3} \\
, \text{Adjectief}(\textsf{onverbogen},3) & \longrightarrow & \text{AdjectiefOnverbogen3} \\
, \text{Adjectief}(\textsf{met-e},2) & \longrightarrow & \text{AdjectiefMetE2} \\
, \text{Adjectief}(\textsf{zonder-e},2) & \longrightarrow & \text{AdjectiefZonderE2} \\
, \text{Adjectief}(\textsf{onverbogen},2) & \longrightarrow & \text{AdjectiefOnverbogen2} \\
, \text{Adjectief}(\textsf{met-e},1) & \longrightarrow & \text{AdjectiefMetE1} \\
, \text{Adjectief}(\textsf{zonder-e},1) & \longrightarrow & \text{AdjectiefZonderE1} \\
, \text{Adjectief}(\textsf{onverbogen},1) & \longrightarrow & \text{AdjectiefOnverbogen1} \\
, \text{Voorbepaling}(\textsf{de-bepaling}) & \longrightarrow & \text{DeBepaling} \\
, \text{Voorbepaling}(\textsf{het-bepaling}) & \longrightarrow & \text{HetBepaling} \\
, \text{Voorbepaling}(\textsf{de/het-bepaling}) & \longrightarrow & \text{DeHetBepaling} \\
, \text{Voorbepaling}(\textsf{een-bepaling}) & \longrightarrow & \text{EenBepaling} \\
\}
\end{array}
$$

Hoofdstuk 3

Ambiguïteit

Wat te valt er te verwachten van een grammatica voor een natuurlijke taal? Dubbelzinnigheid! Zoals al in de inleiding vermeldt, treden er op verschillende plaatsen dubbelzinnigheden op. De eenvoudigste problemen — de synoniemen — lost de lexical scanner op, de complexere — ambiguïteit — worden door de parser opgelost.

In de eerste paragraaf schetsen we de verschillende categorieën. De volgende paragraaf behandelt enkele voorbeelden wat uitvoeriger.

3.1 De probleemgevallen

We kunnen twee hoofdgevallen onderscheiden: de synoniemen en ambigue constructies. Het eerste geval valt uiteen in drie categorieën.

- Syntactisch ononderscheidbare synoniemen.

  `de bank in de bank` (het meubel in het gebouw)

  Deze categorie levert voor ons geen problemen op.

- Syntactisch onderscheidbare synoniemen.

  `de pad op het pad` (de amfibie op de weg)

  Het woord `pad` wordt voortgebracht door twee nonterminals: door de Substantief(de-woord) en door de Substantief(het-woord). Voor ons valt deze categorie samen met de volgende.

- Woordtekens die tot verschillende woordtypes behoren.

  `een wild vrij paard en een vrij wild paard`

  `vrij` is zowel een Bijwoord als een AdjectiefZonderE3.

Het tweede hoofdgeval kunnen we onderverdelen in twee categorieën.

- Grammaticale ambiguïteit.

  `Sneeuwwitje en de zeven dwergen en de heks`

  Wat zowel opgevat kan worden als

  `(Sneeuwwitje en de zeven dwergen) en de heks`

  of als

  `Sneeuwwitje en (de zeven dwergen en de heks)`

  Eigenlijk is dit onderscheid niet interessant. Maar omdat onze grammatica toch al — vanwege de volgende categorie — ambigue constructies opspoort kan dit in een moeite meegenomen worden.

- Semantische ambiguïteit

  `de zoon van de koning en de koningin` (heeft/hebben honger.)

  Deze semantische dubbelzinnigheid is wel interessant. Merk bijvoorbeeld op dat

  `de haat tussen de zoon van (de koning en de koningin)`

  geen goede ontleding is.

3.2 Voorbeelden

Twee interessante categorieën zullen we in deze paragraaf toelichten aan de hand van zogeheten ontleed schema's. Dat zijn schema's die de parse-boom van een zin(-sdeel) duidelijk weergeven. Zo vinden we bij het voorbeeld uit de derde categorie de volgende twee schema's.

<table>
<tr><th>een</th><th>vrij</th><th>wild</th><th>paard</th></tr>
<tr><td rowspan="3">Voorbepaling</td><td>Bijwoord</td><td>Adjectief</td><td rowspan="3">Substantief</td></tr>
<tr><td colspan="2">AdjectiefGroep</td></tr>
<tr><td colspan="3">SubstantiefGroep</td></tr>
</table>

en

<table>
<tr><th>een</th><th>vrij</th><th>wild</th><th>paard</th></tr>
<tr><td rowspan="3">Voorbepaling</td><td>Adjectief</td><td>Adjectief</td><td rowspan="3">Substantief</td></tr>
<tr><td colspan="2">AdjectiefGroep</td></tr>
<tr><td colspan="3">SubstantiefGroep</td></tr>
</table>

Bij het voorbeeld van de vijfde categorie zijn ook twee verschillende ontledingen mogelijk.

<table>
<tr><th>de</th><th>zoon</th><th>van</th><th>de</th><th>koning</th><th>en</th><th>de</th><th>koningin</th></tr>
<tr><td rowspan="5">Voorbepaling</td><td rowspan="5">Substantief</td><td rowspan="5">Voorzetsel</td><td>Voorbepaling</td><td>Substantief</td><td rowspan="5"></td><td>Voorbepaling</td><td>Substantief</td></tr>
<tr><td colspan="2">SubstantiefGroep</td><td colspan="2">SubstantiefGroep</td></tr>
<tr><td colspan="4">SubstantiefGroep</td></tr>
<tr><td colspan="3">SubstantiefGroep</td><td colspan="4">Voorzetselconstituent</td></tr>
<tr><td colspan="8">SubstantiefGroep</td></tr>
</table>

en

<table>
<tr><th>de</th><th>zoon</th><th>van</th><th>de</th><th>koning</th><th>en</th><th>de</th><th>koningin</th></tr>
<tr><td rowspan="5">Voorbepaling</td><td rowspan="5">Substantief</td><td rowspan="5">Voorzetsel</td><td>Voorbepaling</td><td>Substantief</td><td rowspan="5"></td><td rowspan="4"></td><td rowspan="4"></td></tr>
<tr><td colspan="2">SubstantiefGroep</td></tr>
<tr><td colspan="3">SubstantiefGroep</td><td colspan="2">Voorzetselconstituent</td></tr>
<tr><td colspan="5">SubstantiefGroep</td><td>Voorbepaling</td><td>Substantief</td></tr>
<tr><td colspan="8">SubstantiefGroep</td></tr>
</table>

3.3 Synoniemen probleem

In [Brandt Corstius] op pagina 171 wordt een suggestie gedaan die wij zullen volgen. Zelfs al besluiten we een Top-Down parser te nemen, dan nog is het verstandig om eerst voor elk woordteken het woordtype te bepalen — dus bij de bodem even opwaarts gaan. Deze "typering" is (tegenwoordig) standaard bij het automatisch ontleden. Hij wordt uitgevoerd door een zogeheten lexical scanner. Dit is een algoritme dat een rij terminals (letters) omzet in een rij pseudo-terminals (woordtypes). Ook wij zullen dus gebruik maken van zo'n lexical scanner.

Maar zoals [Brandt Corstius] ook al opmerkt, synoniemen gooien roet in het eten. Het is namelijk niet mogelijk om een woordteken te vervangen door één woordtype. De suggestie van [Brandt Corstius] is: vervang elk woordteken door een verzameling mogelijke woordtypes. Wij hebben echter een andere oplossing voor ogen.

We laten de lexical scanner uit een rauwe tekst een rij van alle mogelijke zintyperingen genereren. Vervolgens bieden we elke gegenereerde zintypering aan de parser aan. Zo zal de lexical scanner het zinsdeel `een vrij wild paard` omzetten in de zintyperingen

< EenBepaling AdjectiefZonderE3 AdjectiefZonderE3 HetWoord
, EenBepaling Bijwoord AdjectiefZonderE3 HetWoord
>

die beide goedgekeurd zullen worden door de parser. Het zinsdeel `de pad op het pad` zal vier typeringen opleveren (zowel `het pad` als `de pad` bestaat):

< DeBepaling DeWoord EVvoorzetsel HetBepaling DeWoord
, DeBepaling DeWoord EVvoorzetsel HetBepaling HetWoord
, DeBepaling HetWoord EVvoorzetsel HetBepaling DeWoord
, DeBepaling HetWoord EVvoorzetsel HetBepaling HetWoord
>

In dit geval zal de parser echter alleen de tweede accepteren.

3.4 Ambigue constructies ontleden

Ambiguïteit is inherent aan natuurlijke taal. Het is heel goed mogelijk de ontworpen grammatica $LL(1)$ te maken door constructies als

SubstantiefGroep	$\longrightarrow$	KernGroep Rest
Rest	$\longrightarrow$	ϵ
Rest	$\longrightarrow$	Voorzetsel SubstantiefGroep

Dan worden weliswaar dezelfde substantiefgroepen herkent, maar de structuur gaat verloren. Omdat we juist geïnteresseerd zijn in de structuur gaat onze voorkeur uit naar een ambigue grammatica.

Voor de hand ligt het om een back-track grammatica te kiezen. Helaas is onze grammatica links-cyclisch zodat een recht-toe-recht-aan algoritme niet eindigt. De grammatica

SubstantiefGroep	$\longrightarrow$	SubstantiefGroep `en` Substantiefgroep
SubstantiefGroep	$\longrightarrow$	KernGroep

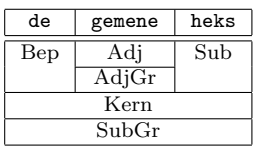
is bijvoorbeeld al links-cyclisch. De parse-procedure die bij de eerste regel hoort mag dus niet eerst naar een substantiefgroep gaan zoeken! Verstandiger is het om de ingevoerde tekst in twee delen te knippen op de terminal `en` en dan beide delen weer aan de parser aan te bieden. Op zo'n manier is de eindiging gegarandeerd: de lengte van de invoer is een goede variante functie.

3.5 De uitvoer

Wat zijn we eigenlijk aan het ontwerpen: een *herkenner* of een *ontleder*? Met andere woorden: is de uitvoer van ons programma op invoer van

```
de gemene heks
```

gewoonweg `Ja` (dat is een substantiefgroep) of zoiets als het ontleed schema



Wij streven naar het laatste, in verband met onze interesse voor ambigue constructies. Vooralsnog zullen we de uitvoer van onze parser echter eenvoudiger houden: we volstaan met het aanbrengen van gelabelde haakjes. Bovenstaand voorbeeld levert dan als uitvoer:

```
[SubGr(Kern(Bep<de>)(AdjGr(Adj<gemene>))(Sub<heks>))]
```

Het is dan altijd nog mogelijk een procedure te ontwerpen die deze haakjes structuur in een of andere boom of blok structuur formatteert. We hebben het type van de haakjes met overleg

gekozen. De vierkante haken [] gebruiken we voor de "zware" groepen, de substantiefgroepen, de ronde () voor de "simpele" groepen en de spitse haken <> voor de woorden. Deze classificering is niet bindend, hij is slechts voor een verhoogde leesbaarheid ingevoerd.

3.6 Een extra attribuut

Nu we vastgelegd hebben wat de uitvoer van ons programma moet zijn, moeten we onze grammatica van een extra attribuut voorzien. In deze paragraaf geven we een formele definitie van de uitgebreide attribuut grammatica G_{so} met $G_{so} = (N_{so}, T_{so}, P_{so}, \text{SubstantiefGroep})$.

3.6.1 De domeinen

We beginnen met de gebruikte domeinen.

$$\begin{aligned}
Bepaling &= \{\text{de-bepaling}, \text{het-bepaling}, \text{de/het-bepaling}, \text{een-bepaling}\} \\
Verbuiging &= \{\text{met-e}, \text{zonder-e}, \text{onverbogen}\} \\
Prioriteit &= \{1, 2, 3\} \\
Woord &= \{\text{de-woord}, \text{het-woord}, \text{mv-woord}\} \\
Getal &= \{\text{meervoud}, \text{enkelvoud}\} \\
Sym &= \{\underline{\text{ a}}, \underline{\text{ b}}, \ldots, \underline{\text{ z}}, \underline{\text{ A}}, \underline{\text{ B}}, \ldots, \underline{\text{ Z}}, \underline{[}, \underline{]}, \underline{(}, \underline{)}, \underline{<}, \underline{>}\} \\
Struc &= Sym^* \\
Strucs &= \mathcal{P}(Struc \times Getal)
\end{aligned}$$

Voor verzamelingen $A \subseteq Sym$ en $B \subseteq Sym$ geldt

$$\begin{aligned}
A \bullet B &= \{a \cdot b \mid a \in A \wedge b \in B\} \\
A^k &= \{a_1 \cdot a_2 \cdot \ldots \cdot a_k \mid (\mathbf{A}i : 1 \leq i \leq k : a_i \in A)\}
\end{aligned}$$

Opmerking: van nu af aan zullen we de infixoperator voor de concatenatie van verzamelingen ($\bullet$) meestal weg laten , net zoals we dat meestal doen voor de concatenatie van de rijtjes zelf ($\cdot$).

Voor de volledigheid geven we de definities van de operaties P, $\times$ en *

$$\begin{aligned}
\mathcal{P}(A) &= \{X \mid X \subseteq A\} \\
A \times B &= \{(a, b) \mid a \in A \wedge b \in B\} \\
A^* &= \bigcup_{k=0}^{\infty} A^k
\end{aligned}$$

We kunnen nu, zoals gebruikelijk het definitie gebied van $\bullet$ en ^{k} uitbreiden. Voor verzamelingen $A \subseteq Struc$ en $B \subseteq Struc$ geldt

$$\begin{aligned}
A \bullet B &= \{a \cdot b \mid a \in A \wedge b \in B\} \\
A^0 &= \emptyset \\
A^{k+1} &= A \bullet A^k
\end{aligned}$$

Tenslotte geven we nog de definitie van de ongebruikelijke genoteerde unaire postfix operatoren $\Downarrow$ en $\Uparrow$:

$$\begin{aligned}
A \Downarrow &= \{x \mid (x, y) \in A\} \\
A \Uparrow &= \{y \mid (x, y) \in A\}
\end{aligned}$$

Om haakjes te besparen kennen we aan de unaire operaties ($\Downarrow$, $\Uparrow$, ^{k}) de hoogste prioriteit toe gevolgd door $\bullet$ en tenslotte $\times$.

3.6.2 De non-terminals

De verzameling N_{so} van nonterminals wordt dan gedefinieerd als:

$N_{so} =$ { SubstantiefGroep($Strucs$)
, Voorzetselconstituent($Strucs$), KernGroep($Getal$,$Struc$)
, AdjectiefGroep($Verbuiging$,$Struc$)
, Voorzetsel($Getal$), Substantief($Woord$)
, Adjectief($Verbuiging$, $Prioriteit$)
, Voorbepaling($Bepaling$), Bijwoord, Telwoord
, DeWoord, HetWoord, MVwoord, Naam
, EVvoorzetsel, MVvoorzetsel, Voegwoord
, AdjectiefMetE3, AdjectiefZonderE3, AdjectiefOnverbogen3
, AdjectiefMetE2, AdjectiefZonderE2, AdjectiefOnverbogen2
, AdjectiefMetE1, AdjectiefZonderE1, AdjectiefOnverbogen1
, DeBepaling, HetBepaling, DeHetBepaling, EenBepaling
}

3.6.3 De (pseudo-)terminals

De verzameling T_{so} van terminals is gedefinieerd als $T_{so} = T_s$. Ook de pseudoterminals van de grammatica G_{so} zijn gelijk aan die van G_s.

3.6.4 De produktieregels

We gebruiken de afkorting

$$\begin{aligned} \text{Sign}(x) &= \text{-1},\ x < 0 \\ &= 0,\ x = 0 \\ &= +1,\ x > 0 \end{aligned}$$

De verzameling P_{so} van produktieregels is alsvolgt gedefinieerd:

$P_{so} =$
{SubstantiefGroep(o) $\longrightarrow$ SubstantiefGroep(o_1) Voegwoord SubstantiefGroep(o_2)
$o:\ o = \{\ \underline{[\text{SubGr}\ }\}o_1 \Downarrow \{\ \underline{(\text{Vw}<>)\ }\}o_2 \Downarrow \{\ \underline{]\ }\} \times \{\mathsf{meervoud}\}$
, SubstantiefGroep(o) $\longrightarrow$ SubstantiefGroep(o_1) Voorzetselconstituent(o_2)
$o:\ o = \{(\ \underline{[\text{SubGr}\ } \cdot s \cdot v \cdot\ \underline{]\ }, g) \mid (s,g) \in o_1 \wedge v \in o_2 \Downarrow\}$
, SubstantiefGroep(o) $\longrightarrow$ KernGroep(s,oo)
$o:\ o = \{(\ \underline{[\text{SubGr}\ } \cdot oo \cdot\ \underline{]\ }, s)\}$
, SubstantiefGroep(o) $\longrightarrow$ Naam
$o:\ o = \{(\ \underline{[\text{SubGr(Naam}<>)]\ }, \mathsf{enkelvoud})\}$
, Voorzetselconstituent(o) $\longrightarrow$ Voorzetsel(v) SubstantiefGroep(o')
$o:\ o = \{(\ \underline{(\text{VzCon(Vz}<>)\ } \cdot s \cdot\ \underline{)\ }, ?) \mid (s,g) \in o' \wedge v = \mathsf{meervoud} \Rightarrow g = \mathsf{meervoud}\}$
, KernGroep(meervoud,oo) $\longrightarrow$ Voorbepaling(v) { Telwoord }$_{0 \le i < M}$
AdjectiefGroep(a,oo') Substantief(s)
$s = \mathsf{mv\text{-}woord} \wedge a \in \{\mathsf{met\text{-}e}, \mathsf{onverbogen}\} \wedge v \in \{\mathsf{de\text{-}bepaling}, \mathsf{de/het\text{-}bepaling}\} \wedge$
$M = 0 \vee M = 1$
$oo:\ oo = \underline{(\text{Kern(Bep}<>)} \cdot \underline{(\text{Telw}<>)}^{\,M} \cdot oo' \cdot \underline{(\text{Sub}<>))}$
, KernGroep(enkelvoud,oo) $\longrightarrow$ Voorbepaling(v) AdjectiefGroep(a,oo') Substantief(s)
$(s = \mathsf{de\text{-}woord} \wedge a \in \{\mathsf{met\text{-}e}, \mathsf{onverbogen}\} \wedge v \in \{\mathsf{de\text{-}bepaling}, \mathsf{de/het\text{-}bepaling}\}) \vee$
$(s = \mathsf{het\text{-}woord} \wedge a \in \{\mathsf{met\text{-}e}, \mathsf{onverbogen}\} \wedge v \in \{\mathsf{het\text{-}bepaling}, \mathsf{de/het\text{-}bepaling}\}) \vee$
$(s = \mathsf{de\text{-}woord}, \wedge a \in \{\mathsf{met\text{-}e}, \mathsf{onverbogen}\} \wedge v = \mathsf{een\text{-}bepaling}) \vee$
$(s = \mathsf{het\text{-}woord} \wedge a \in \{\mathsf{zonder\text{-}e}, \mathsf{onverbogen}\} \wedge v = \mathsf{een\text{-}bepaling})$
$oo:\ oo = \underline{(\text{Kern(Bep}<>)} \cdot oo' \cdot \underline{(\text{Sub}<>))}$

$$
\begin{array}{lll}
, \text{AdjectiefGroep}(\textsf{met-e},oo) & \longrightarrow & \{\ \text{Bijwoord}\ \}_{0\le i<M}\ \{\ \text{Adjectief}(v_i,n_i)\ \}_{0\le i<N} \\
& & (\mathbf{A}i : 1 \le i < N : n_{i-1} \ge n_i) \wedge \\
& & (\mathbf{A}i : 0 \le i < N : v_i \in \{\textsf{met-e}, \textsf{onverbogen}\}) \wedge \\
& & (\mathbf{E}i : 0 \le i < N : v_i = \textsf{met-e}) \wedge \\
& & M = 0 \vee M = 1 \\
& & oo : \ oo = \underline{(\text{AdjGr}} \cdot \underline{(\text{Bijw<>})}^{\,M} \cdot \underline{(\text{Adj<>})}^{\,N} \cdot \underline{)} \\
, \text{AdjectiefGroep}(\textsf{zonder-e},oo) & \longrightarrow & \{\ \text{Bijwoord}\ \}_{0\le i<M}\ \{\ \text{Adjectief}(v_i,n_i)\ \}_{0\le i<N} \\
& & (\mathbf{A}i : 1 \le i < N : n_{i-1} \ge n_i) \wedge \\
& & (\mathbf{A}i : 0 \le i < N : v_i \in \{\textsf{zonder-e}, \textsf{onverbogen}\}) \wedge \\
& & (\mathbf{E}i : 0 \le i < N : v_i = \textsf{zonder-e}) \wedge \\
& & M = 0 \vee M = 1 \\
& & oo : \ oo = \underline{(\text{AdjGr}} \cdot \underline{(\text{Bijw<>})}^{\,M} \cdot \underline{(\text{Adj<>})}^{\,N} \cdot \underline{)} \\
, \text{AdjectiefGroep}(\textsf{onverbogen},oo) & \longrightarrow & \{\ \text{Bijwoord}\ \}_{0\le i<M}\ \{\ \text{Adjectief}(v_i,n_i)\ \}_{0\le i<N} \\
& & (\mathbf{A}i : 1 \le i < N : n_{i-1} \ge n_i) \wedge \\
& & (\mathbf{A}i : 0 \le i < N : v_i \in \{\textsf{onverbogen}\}) \wedge \\
& & M = 0 \vee (M = 1 \wedge N > 0) \wedge \\
& & oo : \ oo = \underline{(\text{AdjGr}}^{\,\text{Sign}(M+N)} \cdot \underline{(\text{Bijw<>})}^{\,M} \cdot \underline{(\text{Adj<>})}^{\,N} \cdot \underline{)}^{\,\text{Sign}(M+N)} \\
, \text{Voorzetsel}(\textsf{meervoud}) & \longrightarrow & \text{MVvoorzetsel} \\
, \text{Voorzetsel}(\textsf{enkelvoud}) & \longrightarrow & \text{EVvoorzetsel} \\
, \text{Substantief}(\textsf{de-woord}) & \longrightarrow & \text{DeWoord} \\
, \text{Substantief}(\textsf{het-woord}) & \longrightarrow & \text{HetWoord} \\
, \text{Substantief}(\textsf{mv-woord}) & \longrightarrow & \text{MVwoord} \\
, \text{Adjectief}(\textsf{met-e},3) & \longrightarrow & \text{AdjectiefMetE3} \\
, \text{Adjectief}(\textsf{zonder-e},3) & \longrightarrow & \text{AdjectiefZonderE3} \\
, \text{Adjectief}(\textsf{onverbogen},3) & \longrightarrow & \text{AdjectiefOnverbogen3} \\
, \text{Adjectief}(\textsf{met-e},2) & \longrightarrow & \text{AdjectiefMetE2} \\
, \text{Adjectief}(\textsf{zonder-e},2) & \longrightarrow & \text{AdjectiefZonderE2} \\
, \text{Adjectief}(\textsf{onverbogen},2) & \longrightarrow & \text{AdjectiefOnverbogen2} \\
, \text{Adjectief}(\textsf{met-e},1) & \longrightarrow & \text{AdjectiefMetE1} \\
, \text{Adjectief}(\textsf{zonder-e},1) & \longrightarrow & \text{AdjectiefZonderE1} \\
, \text{Adjectief}(\textsf{onverbogen},1) & \longrightarrow & \text{AdjectiefOnverbogen1} \\
, \text{Voorbepaling}(\textsf{de-bepaling}) & \longrightarrow & \text{DeBepaling} \\
, \text{Voorbepaling}(\textsf{het-bepaling}) & \longrightarrow & \text{HetBepaling} \\
, \text{Voorbepaling}(\textsf{de/het-bepaling}) & \longrightarrow & \text{DeHetBepaling} \\
, \text{Voorbepaling}(\textsf{een-bepaling}) & \longrightarrow & \text{EenBepaling} \\
\}
\end{array}
$$

3.7 Herstel

Het zal de oplettende lezer niet ontgaan zijn dat we tussen de spitse haken <> bij het samenstellen we de structuur nergens het woord invullen dat op die plaats hoort. Dat is ook niet mogelijk omdat we op die plaats alleen nog maar de categorie kennen waartoe het woord behoort, en niet het woord zelf. Door echter nauwkeurig de speciale haakjes te zetten op de plaatsen waar de woorden horen te staan is het achteraf mogelijk op eenvoudige wijze de losse woorden weer tussen te voegen.

Hoofdstuk 4

Datastructuren

De lexical scanner, die we in het volgende hoofdstuk ontwerpen zet een rijtje woordtekens om in een (of meerdere) rijtje(s) woordtypes. Het is dus niet onverstandig om eerst de datastructuren te ontwerpen die de rauwe invoertekst en de daaruit te destilleren woordtypes representeren vóór we de scanner (of de parser) ontwerpen. Dat is wat we in dit hoofdstuk doen.

4.1 Rauwe tekst

De gebruiker voert een rijtje terminals in. Voor deze onbehandelde invoer definiëren we een synoniem van het standaard type `string` dat een willekeurig rijtje karakters kan bevatten.

```
Type RauweTekst = String;
```

Helemaal willekeurig zijn die rijtjes overigens niet. Ze mogen niet langer dan 255 karakters zijn. Omdat ze slechts één substantiefgroep hoeven te bevatten lijkt ons dit geen bezwaar.

Een van de weinige operaties op het type `RauweTekst` zal het isoleren van de losse woorden zijn. Daarvoor ontwerpen we dus een procedure:

```
Procedure Isoleer(Var zin:RauweTekst; Var Woord:RauweTekst);
  Var p:Byte;
  Begin {Isoleer}
    While (Length(zin)>0) and (zin[1]=' ') Do zin:=Copy(zin,2,Length(zin)-1);
    p:=Pos(' ',zin+' ')-1; Woord:=Copy(zin,1,p); Delete(zin,1,p)
  End;  {Isoleer}
```

Het geïsoleerde woord wordt verwijdert uit de zin. Was de zin leeg, dan blijft hij dat en `Woord` wordt de lege string.

4.2 De woordtypes

De lexical scanner moet deze rauwe invoer omzetten in een rijtje woordtypes. De pseudoterminals uit 2.4.4 zijn precies de woordtypes die wij onderscheiden. We definiëren dus

```
Type
  Categorie   = (Bijwoord,Telwoord,DeWoord,HetWoord,MVwoord,Naam
                ,EVvoorzetsel,MVvoorzetsel,Voegwoord
                ,AdjectiefMetE3,AdjectiefZonderE3,AdjectiefOnverbogen3
                ,AdjectiefMetE2,AdjectiefZonderE2,AdjectiefOnverbogen2
                ,AdjectiefMetE1,AdjectiefZonderE1,AdjectiefOnverbogen1
                ,DeBepaling,HetBepaling,DeHetBepaling,EenBepaling
                );
```

4.3 Behandelde tekst

De zinnen zoals die aan de parser aangeboden worden zijn in feite rijtjes van de in de vorige paragraaf gedefinieerde categorieën, in plaats van rijtjes van ASCII karakters. En dat is jammer want voor rijtjes ASCII karakters is het zeer goed manipuleerbare standaardtype `String` aanwezig.

Onze parser zal straks zinnen in stukjes moeten hakken, woorden uit zinnen verwijderen en moeten zoeken naar woorden. Al deze operaties zijn standaard voor het type `String` maar zullen voor een eigen type `BehandeldeTekst` zelf gedefinieerd moeten worden.

Tenzij . . . , we een bijectieve functie — en zijn inverse — definiëren die categorieën op karakters afbeeldt. Ook hier komt Turbo-Pascal ons tegemoet. Zowel `Categorie` als `Char` zijn namelijk ordinal-types zodat conversie eenvoudig kan plaatsvinden:

```
Function CatToChar(Cat:Categorie):Char;
  Begin {CatToChar}
    CatToChar:=Char(Cat)
  End;  {CatToChar}

Function CharToCat(Ch:Char):Categorie;
  Begin {CharToCat}
    CharToCat:=Categorie(Ch)
  End;  {CharToCat}
```

Als we dan definiëren

```
Type BehandeldeTekst = String;
```

Dan hebben we nu rijtjes woordtypes mét een aantal handige operaties als

- concatenatie
  infix genoteerd met `+`.
- splitsen
  De functie `Copy(s,i,a)` levert van string `s` de substring die begint op positie `i` met lengte `a`.
- verwijderen
  De procedure `Delete(s,n,p)` verwijdert uit string `s` de substring ter lengte `n` die begint op positie `p`.
- invoegen
  De procedure `Insert(s,t,p)` voegt direct ná positie `p` in string `s` de substring `t` in.
- zoeken
  De functie `Pos(s,t)` levert de eerste positie van de string `s` in de string `t`. Komt `s` niet voor in `t` dan levert de functie de waarde 0 op.

Hiermee zijn we op eenvoudige wijze in staat zelf procedures samen te stellen

```
Procedure Rest(Var zin:BehandeldeTekst);
  Begin {Rest}
    zin:=Copy(zin,2,Length(zin)-1)
  End;  {Rest}
```

4.4 Rijtje zinnen

Zoals we al zagen in 3.3 zal de lexical scanner over het algemeen meer dan één `BehandeldeTekst` bij een ingevoerde `RauweTekst` opleveren. We hebben dus een type nodig dat een onbekend aantal zin-typeringen moet kunnen herbergen. Wat ligt meer voor de hand dan een lijst.

```
Type
  Row     =^RowNode;
  RowNode = Record
              z     : BehandeldeTekst;
              next  : Row
            End;
```

Enerzijds hebben we functies nodig om rijtjes te construeren: EmptyRow, SingleRow en de concatenatie van twee rijen. Anderzijds moeten we rijtjes kunnen raadplegen: Head en Tail. Het lege rijtje representeren we door

```
Const EmptyRow : Row = Nil;
```

een "singlerow" maken we met behulp van

```
Function SingleRow(zin:BehandeldeTekst):Row;
  Var r:Row;
  Begin {SingleRow}
    New(r); r^.z:=zin; r^.next:=EmptyRow; SingleRow:=r
  End;  {SingleRow}
```

en de concatenatie van twee rijen gaat met behulp van

```
Function Concat(r1,r2:Row):Row;
  Var r:Row;
  Begin {Concat}
    If r1=EmptyRow
    Then Concat:=r2
    Else Begin
      r:=r1; While r^.next <> EmptyRow Do r:=r^.next;
      r^.next:=r2; Concat:=r1
    End
  End;  {Concat}
```

Het raadplegen gaat met

```
Function Head(r:Row):BehandeldeTekst;
  {Pre: r<>EmptyRow}
  Begin {Head}
    Head:=r^.z
  End;  {Head}

Function Tail(r:Row):Row;
  {Pre: r<>EmptyRow}
  Begin {Tail}
    Tail:=r^.next; Dispose(r)
  End;  {Tail}
```

4.5 De data module

In appendix A vindt U de volledige sourcecode van de module die deze datatypes met hun operatoren definieert en exporteert naar andere modules die hen wensen te gebruiken (`Uses`). De modules die we nog gaan ontwikkelen definiëren soms ook nog datatypes die bij weer andere modules gebruikt worden. Zulke definities komen ook in de Data-module.

Hoofdstuk 5

De lexical scanner

In dit hoofdstuk ontwerpen we de lexical scanner. In de eerste paragraaf geven we een functionele omschrijving. De tweede paragraaf behandelt de implementatie van een dynamische woordenlijst. In de laatste paragraaf ontwerpen we dat deel dat een aangeboden `RauweTekst` omzet in een `BehandeldeTekst`.

5.1 Taakomschrijving

Wat moet onze lexical scanner doen? Strikt genomen heeft hij als taak de afzonderlijke woordtekens van de aangeboden rij letters (terminals) te isoleren, om deze vervolgens om te zetten in rijen woordtypes die aan de parser kunnen worden aangeboden. De lexical scanner moet dus voorzien zijn van een lexicon zodat hij woordtekens op woordtypes kan afbeelden. Gelukkig gebruiken wij slechts een zeer beperkt aantal woordtypes. De pseudoterminals uit paragraaf 2.4.4 vormen de woordtypes voor onze scanner, de terminals die zij voortbrengen de bijbehorende woordtekens.

De lexical scanner kunnen we op twee manieren implementeren: star en dynamisch. We noemen een scanner star als de lexicon als het ware in het programma is opgenomen, en dus niet (eenvoudig) gewijzigd kan worden. Is de lexicon in een apart en wijzigbaar bestand aanwezig dan spreken wij van een dynamische lexical scanner. Uiteraard geniet de dynamisch oplossing de voorkeur.

Een dynamische lexical scanner zal eerst zijn lexicon moeten inlezen om er mee aan de slag te kunnen. Van elk ingevoerd zinsdeel zullen de woorden geïsoleerd moeten worden. Elk woord moet vervolgens getypeerd worden; het zal opgezocht moeten worden in de lexicon om zijn woordtype te kunnen bepalen. Verschillende woordtyperingen leiden dan tot een rijtje (zin-)typeringen. Daarop wordt de parser losgelaten.

5.2 De lexicon

Wat is een geschikte opslagmethode voor een externe lexicon zodat aanmaak en onderhoud eenvoudig is? Een tekstbestand lijkt ons adequaat:

```
DeWoord
ridder
fee
pad
***
HetWoord
bos
pad
***
```

. . .

Achtereenvolgens definiëren we de grammatica G_l voor dit bestand, geven de benodigde datatypes en ontwerpen de bijbehorende procedures en functies. In de volgende paragraaf zullen we dan zien hoe de lexical scanner de lexicon vult en op zijn beurt zijn diensten aanbiedt aan de buitenwereld.

5.2.1 De invoergrammatica

In deze paragraaf geven we een formele definitie van de grammatica G_l met $G_l = (N_l, T_l, P_l, \text{Lijst})$ voor de lexicon.
De verzameling N_l van nonterminals definiëren we als:
N_l = { Letter, Mark, Woord, WoordenLijst, Lijst }
De verzameling T_l van terminals definiëren we als:

T_l = { `a`, `b`, . . . , `z`
, `A`, `B`, . . . , `Z`
, `*`, Eoln
}

Het symbool "Eoln" (EndOfLiNe) staat voor het, door het operating system gereserveerde, einde-regel symbool dat, volgens de afspraken, in tekst-bestanden het einde van een regel markeert.
P_l =

{Letter	⟶	`a` | `b` | . . . | `z` |
		`A` | `B` | . . . | `Z`
, Mark	⟶	`***`
, Woord	⟶	{ Letter } Eoln
, WoordenLijst	⟶	{ Woord } Mark
, Lijst	⟶	`Bijwoord` Woordenlijst
		`Telwoord` Woordenlijst
		`DeWoord` Woordenlijst
		`HetWoord` Woordenlijst
		`MVwoord` Woordenlijst
		`Naam` Woordenlijst
		`EVvoorzetsel` Woordenlijst
		`MVvoorzetsel` Woordenlijst
		`Voegwoord` Woordenlijst
		`AdjectiefMetE3` Woordenlijst
		`AdjectiefZonderE3` Woordenlijst
		`AdjectiefOnverbogen3` Woordenlijst
		`AdjectiefMetE2` Woordenlijst
		`AdjectiefZonderE2` Woordenlijst
		`AdjectiefOnverbogen2` Woordenlijst
		`AdjectiefMetE1` Woordenlijst
		`AdjectiefZonderE1` Woordenlijst
		`AdjectiefOnverbogen1` Woordenlijst
		`DeBepaling` Woordenlijst
		`HetBepaling` Woordenlijst
		`DeHetBepaling` Woordenlijst
		`EenBepaling` Woordenlijst
}		

5.2.2 De datatypes

Alle ingelezen woorden moeten zodanig worden opgeslagen dat ze tijdens het woordtyperen snel kunnen worden teruggevonden. Met snel bedoelen we een tijdscomplexiteit $\mathcal{O}(^2\log n)$ als n het aantal woorden is. Hoewel een binairy searchtree wellicht voor de hand ligt hebben we gekozen voor een gewoon array.

Tijdens het inlezen is het array eenvoudig gesorteerd te houden en het opzoeken gaat vervolgens inderdaad $\mathcal{O}(^2\log n)$ met de binairy search. Een nadeel is natuurlijk de vooraf te bepalen grootte, die misschien te klein zal blijken. Wij vinden dit echter niet zo bezwaarlijk: het aantal woorden is naar onze smaak meer dan voldoende.

We zullen de lexicon echter in een aparte module, `Unit` geheten, plaatsen zodat een wijziging van deze ontwerpbeslissing geen al te onoverzichtelijke gevolgen heeft. Mocht er later behoefte zijn aan een efficiënter geheugen gebruik dan kan deze `Unit` herschreven worden met binairy search trees met de bijbehorende lrots, rdrots etc.

Hoeveel woorden mogen er in onze lexicon? Dat hangt nauw samen met de lengte van de woorden. Wederom zijn we lui. We reserveren voor elk woord een vaste ruimte van twintig letter:

```
Const
  MaxWoordLen    = 20;

Type
  WoordType      = String[MaxWoordLen];
```

Daar de maximale geheugenruimte ongeveer 64k bytes groot is, kunnen we ruim 2500 woorden opslaan:

```
Const
  MaxAantWoorden = 2500;

Type
  WoordIndex     = 0..MaxAantWoorden-1;
  WoordRange     = 0..MaxAantWoorden;
  WoordLijst     = Array[WoordIndex] Of Record
                     woord : WoordType;
                     cat   : Categorie
                   End;

Var
  Lijst          : WoordLijst;
  AantalWoorden  : WoordRange;
```

5.2.3 Toevoegen

De woorden die de parser uit het lexicon-bestand haalt moeten worden toegevoegd aan de interne lexicon, en wel zo, dat deze gesorteerd blijft. Het lijkt wel verstandig eerst te controleren of het woord wel aan de gestelde eisen voldoet:

```
Procedure ControleerWoord(woord:RauweTekst);
  Var i:1..MaxWoordLen;
  Begin {Controleerwoord}
    If Length(woord)>MaxWoordLen Then Error('Woord te lang: '+woord);
    If Length(woord)=0 Then Error('Een woord ter lengte nul.');
    For i:=1 To Length(woord) Do If Not(woord[i] in ['a'..'z','A'..'Z'])
      Then Error('Illegaal karakter in '+woord)
  End;  {Controleerwoord}
```

Dan pas mag het aan de lexicon worden toegevoegd, mits deze niet vol is:

```
Procedure AddToLexicon(woord:RauweTekst; cat:Categorie);
  Begin {AddToLexicon}
    ControleerWoord(woord);
    If AantalWoorden=MaxAantWoorden
```

```
      Then Error('Lexicon is vol')
      Else Insert(woord,cat)
  End;  {AddToLexicon}
```

Dat toevoegen gebeurt feitelijk met

```
Procedure Insert(woord:WoordType; cat:Categorie);
  Var i:WoordRange;
  Begin {Insert}
    i:=AantalWoorden;
    While (i>0) and (woord<Lijst[i-1].woord) Do
      Begin Lijst[i]:=Lijst[i-1]; i:=i-1 End;
    Lijst[i].woord:=woord; Lijst[i].cat:=cat;
    AantalWoorden:=AantalWoorden+1
  End;  {Insert}
```

De nog niet gedefinieerde `Procedure Error` drukt een foutmelding af en aborteert het programma (gebruikmakend van de standaard-opdracht `Halt`):

```
Procedure Error(Fout:String);
  Begin {Error}
    Meld('Er is een fatale fout opgetreden;'); Meld(Fout); Meld(''); Halt
  End;  {Error}
```

Deze procedure stoppen we samen met de standaard uitvoer routine `Meld` in een module uitvoer. De source vindt U in appendix B.

5.2.4 Opzoeken

Naast het opslaan van aangeboden woorden heeft de lexicon module nog één taak: het opzoeken van een woord in de lexicon. Dat kan en dat doen we dus met een binairy search ([Dijkstra]). Hier treedt het probleem op dat één woord(teken) meerdere woordtypes kan hebben. Daarom geeft onze functie een rijtje woordtypes af. Staat het woord niet in de lexicon dan is dit rijtje dus leeg.

```
Function WoordTypes(woord:RauweTekst):BehandeldeTekst;
  Var i,j:WoordRange; Types:BehandeldeTekst; h:Integer;
  Begin {WoordTypes}
    i:=0; j:=AantalWoorden;
    While i+1 <> j Do
    Begin
      h:=(i+j) div 2;
      If Lijst[h].woord<=woord Then i:=h Else j:=h
    End;
    Types:=''; h:=i;
    While (h>=0) and (Lijst[h].woord=woord) Do
      Begin Types:=Types+CatToChar(Lijst[h].cat); h:=h-1 End;
    WoordTypes:=Types;
  End;  {WoordTypes}
```

5.2.5 De lexicon module

We hebben nu de datastructuren en de bijbehorende procedures en functies (`AddToLexicon` en `WoordTypes` in dit geval) ontwikkeld die de lexicon implementeren. Zoals reeds opgemerkt wensen we de gebruiker af te schermen van de feitelijke implementatie; alleen de kennis van het gedrag van de procedures en functies is voor hem belangrijk.

Dat is de reden voor ons om een aparte `Unit` te schrijven die deze lexicon implementeert. De volledige source code vindt U in appendix C. Merk nog op dat deze module zich automatisch initialiseert. Dit gebeurt in de body van de module die aan het einde van de listing staat.

5.3 De scanner

We beschikken nu over een module die een lexicon implementeert. De lexical scanner waarvan we nu de module gaan ontwerpen zal allereerst de lexicon moeten vullen (paragraaf 5.3.1) voor hij hem kan raadplegen (paragraaf 5.3.2). De complete source vindt U in appendix D.

5.3.1 De parser voor de lexicon

Met opzet is de structuur van het lexicon-bestand (zie 5.2.1) eenvoudig gehouden. Elke regel bevat precies één woord, hetzij een categorie woord (als de vorige categorie is afgesloten door `***`), hetzij een woord voor de lexicon.

We hebben dus een procedure nodig die een regel `RauweTekst` van het lexicon-bestand kan inlezen:

```
Procedure Leesregel(Var regel:RauweTekst);
  Begin {LeesRegel}
    If Eof
      Then Error('Onverwacht einde van het bestand. "***" vergeten?');
    ReadLn(regel);
    If Length(regel)>254
      Then Error('Regel langer dan 254 karakters.')
  End;  {LeesRegel}
```

Mocht het tekst-bestand regels bevatten die langer zijn dan 255 karakters, dan plaatst `ReadLn` slechts de eerste 255 in de variabele `regel` terwijl de overige karakters genegeerd worden. Onze procedure `Leesregel` geeft een foutmelding, die berust op dit principe, als te lange regels gebruikt worden.

Bevat een regel `***`, dan wordt een nieuwe categorie opgevoerd. Het zou handig zijn een lijst van alle categorieën ter beschikking te hebben (merk op dat de langste categorienaam 20 karakters lang is):

```
Const
  CategorieNaam : Array[Categorie] Of String[20]=
                  ('Bijwoord','Telwoord','DeWoord','HetWoord','MVwoord','Naam'
                  ,'EVvoorzetsel','MVvoorzetsel','Voegwoord'
                  ,'AdjectiefMetE3','AdjectiefZonderE3','AdjectiefOnverbogen3'
                  ,'AdjectiefMetE2','AdjectiefZonderE2','AdjectiefOnverbogen2'
                  ,'AdjectiefMetE1','AdjectiefZonderE1','AdjectiefOnverbogen1'
                  ,'DeBepaling','HetBepaling','DeHetBepaling','EenBepaling'
                  );
```

zodat we eenvoudig kunnen controleren of het de juiste is:

```
Procedure ControleerCategorie(regel:RauweTekst; Cat:Categorie);
  Begin {ControleerCategorie}
    If regel=CategorieNaam[Cat]
      Then Meld('Inlezen van '+CategorieNaam[Cat])
      Else Error('"'+CategorieNaam[Cat]+'" verwacht.')
  End;  {ControleerCategorie}
```

De parser zelf is dan kinderlijk eenvoudig.

```
Var
  Cat   : Categorie;
  regel : RauweTekst;
Begin {InitLexicon}
  For Cat:=BijWoord To EenBepaling Do
```

```
  Begin
    Leesregel(regel); ControleerCategorie(regel,cat);
    Repeat
      Leesregel(regel);
      If regel<>'***' Then AddToLexicon(regel,cat)
    Until regel='***'
  End;
  If Not Eof Then Error('Einde bestand verwacht.')
End;  {InitLexicon}
```

5.3.2 Woordtyperen

Als we de lexicon gevuld hebben dan kunnen we hem raadplegen. We ontwerpen een recursieve procedure die een rij van alle (zin-)typeringen genereert.

```
Function Scan(kop:BehandeldeTekst; staart:RauweTekst):Row;
  Var woord:RauweTekst; types:BehandeldeTekst; r:Row; p:Byte;
  Begin {Scan}
    Isoleer(staart,woord);
    If woord=''
    Then Scan:=SingleRow(kop)
    Else Begin
      types:=WoordTypes(woord); r:=EmptyRow;
      If types='' Then Meld('"'+woord+'" is niet opgenomen in de woordenlijst');
      For p:=1 To Length(types) Do r:=ConCat(Scan(kop+types[p],staart),r);
      Scan:=r
    End
  End;  {Scan}
```

Hoofdstuk 6

De parser

In dit hoofdstuk ontwikkelen we de parser. Zoals we al eerder zagen voldoen de standaard parsers niet voor ons probleem. Wij zullen gebruik maken van een gecombineerde (aangepaste-)back-track recursive-descent parser. Allereerst bepalen we de benodigde datastructuren voor de attributen. Dan ontwerpen we het recursive descent gedeelte en daarna het back-track gedeelte van onze parser.

6.1 De datastructuren

We beginnen met de implementatie van de domeinen zoals ze gedefinieerd zijn in 3.6.1. Voor *Bepaling*, *Verbuiging*, *Woord* en *Getal* ligt het meest voor de hand om zogeheten enumerated types te gebruiken:

```
Type
  Bepaling   = (de_bepaling, het_bepaling, dehet_bepaling, een_bepaling);
  Verbuiging = (met_e, zonder_e, onverbogen);
  Woord      = (de_woord, het_woord, mv_woord);
  Getal      = (meervoud, enkelvoud);
```

Voor het domein *Prioriteit* nemen we natuurlijk een subrange van `Integer`:

```
Type
  Prioriteit = 1..3;
```

Het domein *Sym* wordt niet gebruikt als attribuut-domein en hoeft daarom niet geïmplementeerd te worden. Het type *Struc* implementeren we door middel van het voorgedefinieerde type `String`. Merk op dat we nu geen bijectieve representatie functie hebben. Dat is echter ook zeer onwaarschijnlijk daar *Struc* oneindig groot is. Bovendien is het onderliggende type niet *Sym* maar de verzameling van alle ASCII karakters. Voor ons is dat geen bezwaar.

```
Type
  Struc      = String;
```

De implementatie van *Strucs* heeft wat meer voeten in aarde. Een verzameling van non-ordinal types “moet” geïmplementeerd worden met behulp van een lijst.

```
Type
  Strucs     =^StrucNode;
  StrucNode  = Record
                 s    : Struc;
                 g    : Getal;
                 next : Strucs
               End;
```

Voor de constructie van *Strucs* definiëren we de constante ∅, de functie `AddElm` die één element aan een verzameling toevoegt en de vereniging (`Union`) van twee verzamelingen. Om een verzameling te kunnen raadplegen definiëren we een functie `PickElm`.

```
Const EmptySet : Strucs = Nil;


Procedure AddElm(Var o:Strucs; s:Struc; g:Getal);
  Var h:Strucs;
  Begin {AddElm}
    New(h); h^.s:=s; h^.g:=g; h^.next:=o; o:=h
  End;  {AddElm}


Function Union(o1,o2:Strucs):Strucs;
  Var o:Strucs;
  Begin {Union}
    If o1=EmptySet
    Then Union:=o2
    Else Begin
      o:=o1; While o^.next <> EmptySet Do o:=o^.next;
      o^.next:=o2; Union:=o1
    End
  End;  {Union}


Procedure PickElm(Var o:Strucs; Var oo:Struc);
  {Pre: o<>EmptySet}
  Var h:Strucs;
  Begin {PickElm}
    oo:=o^.s; h:=o^.next; Dispose(o); o:=h
  End;  {PickElm}
```

6.2 Recursive-descent parser

Wellicht vraagt U zich af welke nonterminals door het recursive descent gedeelte geparsed moeten worden en welke door het back-track gedeelte. Het antwoord op deze vraag is eenvoudig en eenduidig. Die nonterminals die het synthesized attribuut *Strucs* hebben, hebben mogelijk meer dan één structuur en worden dus door het back-track gedeelte geparsed. De overige parsen we met de recursive descent parser.

Volgens de regels der kunst, volgens [Hemerik] dus, ontwerpen we de recursive descent parser procedures. Merk op dat de KernGroep in feite de wortel van de parseboom is. Hij initialiseert de recursive descent parser. Merk bovendien op dat een recursive descent parser altijd gebruik maakt van een sentinel of eindmarkering. Dat is een "vers" symbool dat aan de taal wordt toegevoegd (er ontstaat wat [Hemerik] noemt een "augmented grammer") zodat de lookahead-set nooit leeg is. Was hij dat bij de oude grammatica wel, dan bevat hij nu het nieuwe symbool: de sentinel.

Hiertoe moeten we het type categorie uitbreiden met een extra waarde: de `EndMarker`. Een ander verschil met de parser zoals [Hemerik] hem bouwt is de eindiging bij het optreden van een fout. [Hemerik] maakt gebruik van een `Goto` opdracht, wij voeren een extra parameter `Ok:Boolean` in die aangeeft of er al dan niet een fout is opgetreden. Is `Ok=False` dan is er een fout opgetreden en is de waarde van de attributen ongedefinieerd!

```
Procedure ParseKernGroep(zin:BehandeldeTekst; Var g:Getal;
                         Var oo:Struc; Var Ok:Boolean);
```

```
Var M:Integer; ss:Struc; v:Bepaling; a:Verbuiging; s:Woord;
    BepOk, AGrOk, SubOk : Boolean;
Begin {ParseKernGroep}
  zin:=zin+CatToChar(EndMarker);
  ParseVoorBepaling(zin,v,BepOk); oo:='(Bep<>)';
  M:=0; While CharToCat(zin[1])=Telwoord Do
    Begin Rest(zin); oo:=oo+'(Telw<>)'; M:=M+1 End;
  ParseAdjectiefGroep(zin,a,ss,AGrOk); oo:=oo+ss;
  ParseSubstantief(zin,s,SubOk); oo:='(Kern'+oo+'(Sub<>))';
  Ok:=BepOk and AGrOk and SubOk and (zin=CatToChar(EndMarker)) and
      (
      ( (s=mv_woord) and
        ((a=met_e) or (a=onverbogen)) and
        ((v=de_bepaling) or (v=dehet_bepaling)) and
        ((M=0) or (M=1))
      ) or
      ( (s=de_woord) and
        ((a=met_e) or (a=onverbogen)) and
        ((v=de_bepaling) or (v=dehet_bepaling)) and
        (M=0)
      ) or
      ( (s=het_woord) and
        ((a=met_e) or (a=onverbogen)) and
        ((v=het_bepaling) or (v=dehet_bepaling)) and
        (M=0)
      ) or
      ( (s=de_woord) and
        ((a=met_e) or (a=onverbogen)) and
        (v=een_bepaling) and
        (M=0)
      ) or
      ( (s=het_woord) and
        ((a=zonder_e) or (a=onverbogen)) and
        (v=een_bepaling) and
        (M=0)
      )
      );
  If s=mv_woord Then g:=meervoud Else g:=enkelvoud
End;  {ParseKernGroep}
```

We hoeven nu alleen nog maar de parser voor de Voorbepaling, Substantief en AdjectiefGroep te schrijven. Laten we beginnen met de eerste (eenvoudige) twee:

```
Procedure ParseVoorBepaling(Var zin:BehandeldeTekst; Var b:Bepaling; Var Ok:Boolean);
  Begin {ParseVoorBepaling}
    Ok:=True;
    Case CharToCat(zin[1]) Of
      DeBepaling    : Begin Rest(zin); b:=de_bepaling    End;
      HetBepaling   : Begin Rest(zin); b:=het_bepaling   End;
      DeHetBepaling : Begin Rest(zin); b:=dehet_bepaling End;
      EenBepaling   : Begin Rest(zin); b:=een_bepaling   End
      Else            Ok:=False
    End
  End;  {ParseVoorBepaling}
```

```
Procedure ParseSubstantief(Var zin:BehandeldeTekst; Var w:woord; Var Ok:Boolean);
  Begin {ParseSubstantief}
    Ok:=True;
    Case CharToCat(zin[1]) Of
      DeWoord  : Begin Rest(zin); w:=de_woord  End;
      HetWoord : Begin Rest(zin); w:=het_woord End;
      MVwoord  : Begin Rest(zin); w:=mv_woord  End
      Else       Ok:=False
    End
  End;  {ParseSubstantief}
```

Nu rest ons nog de parser voor de Adjectiefgroep. Waarbij we willen opmerken dat `3` het maximum is van *Prioriteit* zodat we eenvoudig invariant kunnen houden dat alle al geparsete adjectieven een prioriteit hadden die groter was dan `Prio` (of dat `Ok` False is): `Prio=3` initialiseert dit.

Tevens houden we invariant dat `MetFound` aangeeft of er al een adjectief geparsed is dat met-e verbogen was en dat `ZonderFound` aangeeft of er al een geparsed is die zonder-e verbogen was. Beide initialiseren we natuurlijk met het eenheidselement van de disjunctie: `False`.

Merk tevens op dat we bij de eerste twee produktieregels voor de AdjectiefGroep $M = 0 \vee M = 1$ mogen versterken tot $M = 0 \vee (M = 1 \wedge N > 0)$ daar de conjunct $N > 0$ geïmpliceerd wordt door de existentiële kwantificaties. Van deze eigenschap maken we in onze procedure gebruik.

```
Procedure ParseAdjectiefGroep(Var zin:BehandeldeTekst; Var v:Verbuiging;
                              Var s:Struc; Var Ok:Boolean);
  Var
    M, N                       : Integer;
    DezeV                      : Verbuiging;
    DezePrio, Prio             : Prioriteit;
    MetFound,ZonderFound,AdjOk : Boolean;
  Begin {ParseAdjectiefGroep}
    s:=''; M:=0;
    While CharToCat(zin[1])=BijWoord Do
      Begin Rest(Zin); M:=M+1; s:=s+'(Bijw<>)' End;
    N:=0; Prio:=3; MetFound:=False; ZonderFound:=False; Ok:=True;
    While CharToCat(zin[1]) in
      [AdjectiefMetE3,AdjectiefZonderE3,AdjectiefOnverbogen3
      ,AdjectiefMetE2,AdjectiefZonderE2,AdjectiefOnverbogen2
      ,AdjectiefMetE1,AdjectiefZonderE1,AdjectiefOnverbogen1] Do
    Begin
      ParseAdjectief(zin,DezeV,DezePrio,AdjOk); s:=s+'(Adj<>)'; N:=N+1;
      Ok:=Ok and AdjOk and (Prio>=DezePrio); Prio:=DezePrio;
      MetFound:=MetFound or (DezeV=met_e);
      ZonderFound:=ZonderFound or (DezeV=zonder_e)
    End;
    If s<>'' Then s:='(AdjGr'+s+')';
    Ok:=Ok and ((M=0) or ((M=1) and (N>0))) and Not(MetFound and ZonderFound);
    If MetFound
      Then v:=met_e
      Else If ZonderFound
        Then v:=zonder_e
        Else v:=onverbogen
  End;  {ParseAdjectiefGroep}
```

Wel hebben we nog een parser nodig voor het adjectief:

```
Procedure ParseAdjectief(Var zin:BehandeldeTekst; Var v:Verbuiging;
                         Var p:Prioriteit; Var Ok:Boolean);
  Begin {ParseAdjectief}
    Ok:=True;
    Case CharToCat(zin[1]) Of
      AdjectiefMetE3       : Begin Rest(zin); v:=met_e; p:=3 End;
      AdjectiefZonderE3    : Begin Rest(zin); v:=zonder_e; p:=3 End;
      AdjectiefOnverbogen3 : Begin Rest(zin); v:=onverbogen; p:=3 End;
      AdjectiefMetE2       : Begin Rest(zin); v:=met_e; p:=2 End;
      AdjectiefZonderE2    : Begin Rest(zin); v:=zonder_e; p:=2 End;
      AdjectiefOnverbogen2 : Begin Rest(zin); v:=onverbogen; p:=2 End;
      AdjectiefMetE1       : Begin Rest(zin); v:=met_e; p:=1 End;
      AdjectiefZonderE1    : Begin Rest(zin); v:=zonder_e; p:=1 End;
      AdjectiefOnverbogen1 : Begin Rest(zin); v:=onverbogen; p:=1 End
      Else                   Ok:=False
    End
  End;  {ParseAdjectief}
```

6.3 Back-track parser

In deze paragraaf ontwerpen we het back-track gedeelte van de parser. Dit beslaat alleen de non-terminals SubstantiefGroep en VoorzetselConstituent. De invoer van deze parseprocedures bestaat uit een zin die geparsed moet worden, de uitvoer bestaat uit de verzameling van alle mogelijke ontleedstructuren.

Zoals al opgemerkt is de grammatica links-cyclisch. Wat moeten dus speciale aandacht aan de terminatie van de back-track parser schenken. Om eindiging van een recursieve procedure te bewijzen volstaat het een variante functie op te noemen. Een zeer voor de handliggende is in ons geval de lengte van de te ontleden zin. Willen we de ontleding

SubStantiefGroep $\longrightarrow$ SubstantiefGroep VoegWoord SubstantiefGroep

proberen, dan moet er op zijn minst één voegwoord in de zin staan! Willen we de ontleding

SubStantiefGroep $\longrightarrow$ SubstantiefGroep Voorzetselconstituent

proberen, dan moet er een voorzetsel zijn. Vanwege de analogie tussen deze regels, zullen we proberen een algemener raamwerk op te zetten waar beide inpassen:

$$
\begin{array}{l}
\text{SubstantiefGroep}(o) \longrightarrow \text{SubstantiefGroep}(o_1)\ \text{Voegwoord}\ \text{SubstantiefGroep}(o_2) \\
\quad o:\ o = \{(\alpha x_1 \beta x_2 \gamma, f(y_1,y_2)) \mid (x_1,y_1) \in o_1 \wedge (x_2,y_2) \in o_2 \wedge P(y_1,y_2)\} \\
\quad\quad \textbf{where}\ f(a,b) = \mathsf{meervoud} \wedge P(a,b) = True \wedge \\
\quad\quad\quad \alpha = \underline{[\text{SubGr}} \wedge \beta = \underline{(\text{Vw}<>)} \wedge \gamma = \underline{]} \\
\text{SubstantiefGroep}(o) \longrightarrow \text{SubstantiefGroep}(o_1)\ \text{EVvoorzetsel}\ \text{SubstantiefGroep}(o_2) \\
\quad o:\ o = \{(\alpha x_1 \beta x_2 \gamma, f(y_1,y_2)) \mid (x_1,y_1) \in o_1 \wedge (x_2,y_2) \in o_2 \wedge P(y_1,y_2)\} \\
\quad\quad \textbf{where}\ f(a,b) = a \wedge P(a,b) = True \wedge \\
\quad\quad\quad \alpha = \underline{[\text{SubGr}} \wedge \beta = \underline{(\text{VzCon}(\text{Vz}<>)} \wedge \gamma = \underline{)]} \\
\text{SubstantiefGroep}(o) \longrightarrow \text{SubstantiefGroep}(o_1)\ \text{MVvoorzetsel}\ \text{SubstantiefGroep}(o_2) \\
\quad o:\ o = \{(\alpha x_1 \beta x_2 \gamma, f(y_1,y_2)) \mid (x_1,y_1) \in o_1 \wedge (x_2,y_2) \in o_2 \wedge P(y_1,y_2)\} \\
\quad\quad \textbf{where}\ f(a,b) = a \wedge P(a,b) = (b = \mathsf{meervoud}) \wedge \\
\quad\quad\quad \alpha = \underline{[\text{SubGr}} \wedge \beta = \underline{(\text{VzCon}(\text{Vz}<>)} \wedge \gamma = \underline{)]}
\end{array}
$$

Naast deze drie recursieve stappen voor de SubstantiefGroep is er nog de dubbele basis. Voor de produktie vanuit de non-terminal SubstantiefGroep zijn er dus vijf mogelijkheden. De vereniging van alle mogelijke ontledingen van elk der regels bevat precies alle mogelijke ontledingen van de SubstantiefGroep.

```
Procedure ParseSubstantiefGroep(zin:BehandeldeTekst; Var o:Strucs);
  Var o1,o2,o3,o4,o5:Strucs;
  Begin {ParseSubstantiefGroep}
    ParseSubstantiefGroep1(zin,o1);
    ParseSubstantiefGroep2(zin,o2);
    ParseSubstantiefGroep3(zin,o3);
    ParseSubstantiefGroep4(zin,o4);
    ParseSubstantiefGroep5(zin,o5);
    o:=Union(o1,Union(o2,Union(o3,Union(o4,o5))))
  End;  {ParseSubstantiefGroep}
```

Laten we beginnen met de eenvoudige procedures; de beide basissen van de substantiefgroep:

```
Procedure ParseSubstantiefGroep1(zin:BehandeldeTekst; Var o:Strucs);
  Var g:Getal; oo:Struc; Ok:Boolean;
  Begin {ParseSubstantiefGroep1}
    o:=EmptySet;
    ParseKernGroep(zin,g,oo,Ok);
    If Ok Then AddElm(o,'[SubGr'+oo+']',g)
  End;  {ParseSubstantiefGroep1}

Procedure ParseSubstantiefGroep2(zin:BehandeldeTekst; Var o:Strucs);
  Begin {ParseSubstantiefGroep2}
    o:=EmptySet;
    If Zin=CatToChar(Naam) Then AddElm(o,'[SubGr(Naam<>)]',enkelvoud)
  End;  {ParseSubstantiefGroep2}
```

Dan komen we nu bij het lastige gedeelte van de parser: het parsen van een nevenschikking:

```
Procedure ParseSubstantiefGroep3(zin:BehandeldeTekst; Var o:Strucs);

  Function P(a,b:Getal):Boolean; Begin P:=True End;
  Function f(a,b:Getal):Getal;   Begin f:=meervoud End;

  Var i:Word; deel1,deel2:BehandeldeTekst; o1,o2:Strucs;
  Begin {ParseSubstantiefGroep3}
    deel1:=''; deel2:=zin; i:=Pos(CatToChar(Voegwoord),deel2); o:=EmptySet;
    While p>0
    Do Begin
      deel1:=deel1+Copy(deel2,1,i-1); deel2:=Copy(deel2,i+1,Length(deel2)-i);
      ParseSubstantiefGroep(deel1,o1); ParseSubstantiefGroep(deel2,o2);
      deel1:=deel1+CatToChar(Voegwoord);
      o:=Union(o,Prod(o1,o2,f,P,'[SubGr','(Vw<>)',']'));
      i:=Pos(CatToChar(Voegwoord),deel2)
    End
  End;  {ParseSubstantiefGroep3}
```

De twee procedures die de voorzetselconstituent behandelen zijn vrijwel identiek aan de bovenstaande procedure. We behandelen ze hier daarom niet. Wel rest ons nog de functie `Prod` die het *gefilterde* cartesische produkt van twee verzamelingen bepaalt.

Deze functie krijgt als parameter onder andere een filter (een functie van $\omega \times \omega \rightarrow Bool$) en een mapper (een functie van $\omega \times \omega \rightarrow \omega$) mee. In Turbo-pascal gaan de bijbehorende typedeclaratie als volgt:

```
Type
  Filter = Function (a,b:Getal):Boolean;
  Mapper = Function (a,b:Getal):Getal;
```

De functie `Prod` heeft dan de volgende specificatie:

```
Function Prod(o1,o2:Strucs; f:Mapper; P:Filter; alfa,beta,gamma:Struc):Strucs;
  {Pre conditie : True
         Return : { (alfa x1 beta x2 gamma, f(y1,y2))
                  | (x1,y1) in o1   and   (x2,y2) in o2   and   P(y1,y2)
                  }
   }
```

Waarbij we als extra (implementatie) eis hebben dat er geen aliassen gecreëerd mogen worden — uit twee lijsten van, zeg, lengte n en m wordt één lijst geproduceerd van lengte (ten hoogste) nm. Bovendien moeten we ervoor waken "rotzooi" op de heap achter te laten. (Turbo-Pascal schijnt een van de weinige Pascal-implementaties te zijn die een "fatsoenlijke" *heapmanager* heeft.)

De body van de functie genereert natuurlijk alle produkten waarvan de gefilterde bewaard mogen worden:

```
h3:=EmptySet;
h1:=o1;
While h1 <> EmptySet Do
Begin
  h2:=o2;
  While h2 <> EmptySet Do
  Begin
    If P(h1^.g,h2^.g)
      Then AddElm(h3,alfa+h1^.s+beta+h2^.s+gamma,f(h1^.g,h2^.g));
    h2:=h2^.next
  End;
  h1:=h1^.next
End;
DisposeSet(o1); DisposeSet(o2);
Prod:=h3
```

De procedures `DisposeSet` verwijderen een gehele verzameling van de heap:

```
Procedure DisposeSet(o:Strucs);
  Var h:Strucs;
  Begin {DisposeSet}
    While o<>EmptySet Do Begin h:=o^.next; Dispose(o); o:=h End
  End;  {DisposeSet}
```

Daar de twee procedures die de voorzetselconstituent behandelen vrijwel identiek zijn aan bovenstaande procedure hebben we besloten één gegeneraliseerde procedure te schrijven die als extra parameters de mapper, de filter, de categorie waarop gesplitst moet worden en de *Struc* die die categorie omschrijft meekrijgt. Voor een volledige listing zie appendix E.

Hoofdstuk 7

Userinterface

Hoewel wij het niet tot onze taak rekenen het ontwikkelde programma van een uitgebreide *user-interface* te voorzien, ontkomen we er natuurlijk niet aan toch enkele voorzieningen te treffen. Welke dat zijn bespreken we in dit hoofdstuk.

7.1 Eenvoud

Omdat ons programma slechts een onderzoeksobject is, zien we af van een geavanceerde userinterface. De praktijk leert weliswaar dat juist de userinterface voor een groot deel de waardering van de gebruikers voor het programma bepaalt, maar de ervaring leert ook dat dit onderdeel van een programma zeer arbeidsintensief en (dus) zeer foutengevoelig is.

We doen twee concessies. De eerste is dat we de ontleed-structuur in de vorm van een ontleedschema willen weergeven. Daar op die manier (zeker als er meerdere ontleedschema's bij één zin passen) het aantal regels uitvoer nogal groot kan worden, zullen we alle uitvoer óók naar een LOG-file sturen; onze tweede concessie.

Een LOG-file is een gewoon tekst-bestand dat een letterlijke copie bevat van de gehele uitvoer van één programma-sessie. Zo'n bestand kan later eenvoudig bekeken (`TYPE`), gewijzigd (`EDIT`) of geprint (`PRINT`) worden.

7.2 De LOG-file

Met een vooruitziende blik hadden we alle uitvoer al via de routine `Meld` uit de module `Uitvoer` laten lopen. Als we deze routine zo wijzigen dat de boodschap niet alleen naar het scherm maar ook naar de LOG-file wordt geschreven hebben we ons doel bereikt.

```
Var
  LogFile : Text;

Procedure Meld(Tekst:String);
  Begin {Meld}
    Writeln(Tekst); Writeln(LogFile,Tekst)
  End;  {Meld}
```

De LogFile moet wel "geopend" worden. Daar leent de initialisatie-body van de module zich goed voor. Een klein probleem is de naam die de LOG-file moet krijgen. De meest flexibele oplossing laat de gebruiker toe deze te specificeren. Bijvoorbeeld als (command line) parameter bij het opstarten van het programma. We kunnen ons dan nog flexibeler opstellen en geen LOG-file aanmaken als de gebruiker geen naam specificeert. We zullen daarvoor introduceren:

```
Var WriteLog : Boolean;
```

Zodat de al eerder gedefinieerde procedure `Meld` nu luidt:

```
Procedure Meld(Tekst:String);
  Begin {Meld}
    Writeln(Tekst); If WriteLog Then Writeln(LogFile,Tekst)
  End;  {Meld}
```

Rest nog de initialisatie-body van de procedure die `WriteLog` de goede waarde moet geven én eventueel de LOG-file moet openen. De functies `ParamCount` en `ParamStr` die respectievelijk het aantal command line parameters en de parameters zelf opleveren, komen hierbij goed van pas.

```
If ParamCount=1
Then Begin
  {$I-} Assign(LogFile,ParamStr(1)); Rewrite(LogFile); {$I+}
  WriteLog:=IoResult=0
End
Else WriteLog:=False
```

De compiler-directives `I-` en `I+` schakelen de automatische Input/Output foutendetectie uit respectievelijk aan. Treedt er tijdens de "uit" fase een fout op dan wordt dat gesignaleerd in de systeemvariabele `IoResult`. In de praktijk betekent dit dat een volle disk of een illegale file naam tot gevolg heeft dat er geen LOG-file wordt aangemaakt (`WriteLog:=False`).

Een probleem waarbij we nog verder moeten afdalen naar het machine-niveau is het sluiten van de file, dat de module `Uitvoer` natuurlijk zelf voor zijn rekening moet nemen. Hiervoor biedt Turbo-Pascal de zogeheten Exit-Chain: door de systeem pointer `ExitProc` naar een eigen (far call) procedure te laten wijzen, zal deze procedure, bij beïndiging van het programma worden uitgevoerd. Wel moet men erop letten de Exit-Chain niet te verbreken. Een en ander werkt uit tot het volgende fragment:

```
Var
  OldExitProc : Pointer;
  WriteLog    : Boolean;
  LogFile     : Text;

{$F+} {Zet de FAR CALL mode aan}
Procedure CloseLogFile;
  Begin {CloseLogFile}
    If WriteLog Then Close(LogFile); ExitProc:=OldExitProc
  End;  {CloseLogFile}
{$F-} {en weer uit}

Procedure Meld(Tekst:String);
  Begin {Meld}
    Writeln(Tekst);
    If WriteLog
    Then Begin
      {$I-} Writeln(LogFile,Tekst); {$I+}
      If IoResult<>0 Then Writeln('Disk(ette) is vol of andere disk-fout.')
    End
  End;  {Meld}

Begin {UitVoer}
  If ParamCount=1
  Then Begin
    {$I-} Assign(LogFile,ParamStr(1)); Rewrite(LogFile); {$I+}
    WriteLog:=IoResult=0
```

```
    End
    Else WriteLog:=False;
    OldExitProc:=ExitProc; ExitProc:=@CloseLogFile
  End.  {UitVoer}
```

7.3 Ontleed schema

In deze paragraaf gaan we een procedure ontwikkelen die een *Struc* omzet in een ontleedschema; die dus van de structuur

```
[SubGr[SubGr(Kern(Bep<>)(Sub<>))](VzCon(Vz<>)[SubGr(kern(Bep<>)(Sub<>))])]
```

die hoort bij de zin

```
een attribuutgrammatica voor de substantiefgroep
```

het volgende ontleedschema maakt.

<table>
<tr><td>een</td><td>attribuutgrammatica</td><td>voor</td><td>de</td><td>substantiefgroep</td></tr>
<tr><td>Bep</td><td>Sub</td><td rowspan="3">Vz</td><td>Bep</td><td>Sub</td></tr>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Kern</td></tr>
<tr><td colspan="2">Kern</td><td colspan="2">SubGr</td></tr>
<tr><td colspan="3">SubGr</td><td colspan="2">VzCon</td></tr>
<tr><td colspan="5">SubGr</td></tr>
</table>

7.3.1 Woorden invullen

De eerste stap is natuurlijk het al besproken proces van het invullen van de woorden. Output genereren betekent randgevallen detecteren. In ons geval betekent dat, dat we rekening moeten houden met het feit dat een structuur wel eens te lang zou kunnen gaan worden, als we de woorden toevoegen.

```
Function WoordenInvullen(zin:RauweTekst; oo:Struc):Struc;
  Var woord:RauweTekst; Overflow:Boolean;
  Begin {WoordenInvullen}
    Overflow:=False;
    Repeat
      Isoleer(zin,woord);
      Overflow:=Length(oo)+Length(woord)>255;
      If Overflow Then Writeln('Sorry, ik kan niet alle woorden invullen.');
      If (woord<>'') and Not Overflow Then Insert(woord,oo,Pos('<>',oo)+1)
    Until Overflow or (zin='');
    WoordenInvullen:=oo
  End;  {WoordenInvullen}
```

7.3.2 Specificatie

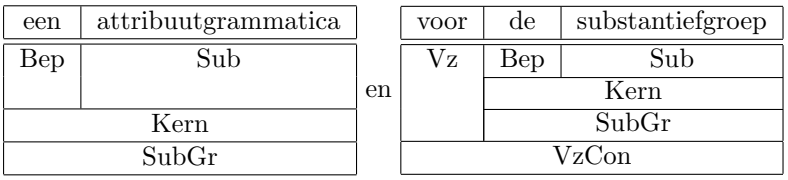
We kunnen de structuur van de zin ook iets gewijzigd weergeven. We schrijven de karakters één voor één op maar vóór we een openingshaakje schrijven gaan we eerst een regel omlaag, en nadat we een sluithaakje hebben geschreven gaan we weer een regel omhoog.

```
[SubGr                                                          ]
      [SubGr                    ](VzCon                        )
            (Kern              )       (Vz  )[SubGr            ]
                 (Bep  )(Sub  )           <>       (kern       )
                     <>     <>                          (Bep  )(Sub  )
                                                           <>     <>
```

Zo zien we de *boomstructuur* wat duidelijker: de wortel bevat de tekst `SubGr` terwijl de (twee) takken de teksten

`[SubGr(Kern(Bep<>)(Sub<>))]` en `(VzCon(Vz<>)[SubGr(kern(Bep<>)(Sub<>))])`

bevatten. De ontleedstructuren van de twee takken luidt:



en U ziet de recursieve bui al hangen: we plaatsen gewoon de wortel er onder:

De procedure die we moeten ontwerpen heeft als invoer een goedhaakse gelabelde haakjesexpressie en als uitvoer een *box* (zoals dat in LaTeX heet) die de ontleed structuur bevat.

7.3.3 De box

Hoe representeren we een box? Wat is een box? Een box is een "rechthoek" die tekst bevat. Bij ons komt dat neer op een rijtje regels van *gelijke* lengte. De regels van een box representeren we, voor de afwisseling, met strings. Om niet al te snel een "Stack overflow" te krijgen, zullen we de lengte beperken:

```
Const
  MaxRegelLength = 150;
Type
  RegelType      = String[MaxRegelLength];
```

Een box bevat een aantal van deze regels. Gemakshalve zullen we een box als een `array` van regels implementeren. We verwachten niet dat de nest-diepte van de haakjes de 10 zal overschrijden, omdat de lengte van een *Struc* ten hoogste 255 is en een "basis" toch al snel de lengte van 25 overschrijdt: `[SubGr(Kern(Bep<>)(Sub<>))]`. Omdat we ook nog horizontale strepen plaatsen voor elke wortel kiezen we voor het maximale aantal regels 20.

```
Const
  MaxLine   = 20;

Type
  LineRange = 0..MaxLine-1;
  LineAant  = 0..MaxLine;
  BoxType   = Record
                line : Array[LineRange] Of RegelType;
                aant : LineAant
              End;
```

Voor we operaties op boxen definiëren zullen we eerst, informeel, een typeinvariant opstellen voor box.

- De regels 0 *tot* `Box.aant` zijn de regels die gedefinieerd zin voor `Box`.
- De lege box stellen we (dus) voor door een box `Box` met `Box.aant=0`.
- Alle regels van één box zijn even lang.

- Een box bevat mogelijk wel een interne structuur van lijnen maar hij is *nooit* omlijnd.

De lijnen waar we het hierboven over hebben worden samengesteld uit de standaard extended ASCII karakters:

```
Const
  hor   = Chr(196); {zoiets als -}
  ver   = Chr(179); {zoiets als |}
  cross = Chr(197); {zoiets als +}
  hordn = Chr(194); {zoiets als T}
  horup = Chr(193); {zoiets als een omgekeerde T}
  verlt = Chr(180); {zoiets als -|}
  verrt = Chr(195); {zoiets als |-}
  uplt  = Chr(218); {voor de linker bovenhoek}
  uprt  = Chr(191); {voor de rechter bovenhoek}
  dnlt  = Chr(192); {voor de linker onderhoek}
  dnrt  = Chr(217); {voor de rechter onderhoek}
```

7.3.4 Het samenvoegen van boxen

De twee belangrijkste operaties op boxen zijn: twee boxen (horizontaal) aan elkaar plakken en een box voorzien van een onderste regel (vertikaal plakken). Hoe dat gaat bespreken we in de volgende twee subparagraaf.

Horizontaal plakken

De regels van de ene box zijn allemaal evenlang en de regels van de andere box zijn op hun beurt ook allemaal evenlang. Plakken we de regels paarsgewijs achter elkaar, dan onstaat een box waarvan alle regels even lang zijn.

Gewoon achter elkaar plakken, dat kan helaas niet. Plakken we twee boxen tegen elkaar, dan moet er een vertikale streep tussen. Hoewel, als een van de boxen leeg is, hoeft zelfs dat niet. Maar zelfs als ze geen van beide leeg zijn is een vertikale streep ook niet altijd adequaat. Een horizontale streep in een van de boxen moet namelijk mooi aansluiten op de vertikale streep. Mooi betekent hier: plaats een T-stukje of zelfs een "cross" in plaats van een vertikaal lijntje.

Dit geavanceerde plakken van twee regels voert de volgende functie uit:

```
Function Connect(s,t:RegelType):RegelType;
  Var a,b:Char;
  Begin {Connect}
    If (s='') or (t='')
    Then Connect:=s+t
    Else Begin
      a:=s[Length(s)]; b:=t[1];
      If (a=hor)  and (b=hor)  Then Connect:=s+cross+t;
      If (a<>hor) and (b=hor)  Then Connect:=s+verrt+t;
      If (a=hor)  and (b<>hor) Then Connect:=s+verlt+t;
      If (a<>hor) and (b<>hor) Then Connect:=s+ver+t
    End
  End;  {Connect}
```

Kunnen we de regels dan gewoon "geavanceerd" achter elkaar plakken en zo de nieuwe box vormen? Was het maar zo simpel: boxen hoeven niet een gelijk aantal regels te bevatten! De kortste box moet aangevuld worden tot hij dezelfde hoogte heeft als de langste box. De regels waarmee hij aangevuld wordt mogen de verticale structuur niet verstoren. Wat deze structuur is bepalen we met

```
Function VerPattern(Box:BoxType):RegelType;
  Var i:Byte; Line:RegelType;
  Begin {VerPattern}
    If Box.aant=0 Then Line:='' Else Line:=Box.line[Box.aant-1];
    For i:=1 To Length(Line) Do If Line[i]<>ver Then Line[i]:=' ';
    VerPattern:=Line
  End;  {VerPattern}
```

Dan kunnen we nu eindelijk de procedure schrijven die de ene box aan de ander plakt:

```
Procedure HorPlak(Var Box1:BoxType; Box2:BoxType);
  Var i:LineAant; p1,p2:RegelType;
  Begin {HorPlak}
    i:=0; p1:=VerPattern(Box1); p2:=VerPattern(Box2);
    While (i<Box1.Aant) and (i<Box2.Aant) Do
      Begin Box1.Line[i]:=Connect(Box1.Line[i],Box2.line[i]); i:=i+1 End;
    While i<Box2.Aant Do Begin Box1.Line[i]:=Connect(p1,Box2.line[i]); i:=i+1 End;
    While i<Box1.Aant Do Begin Box1.Line[i]:=Connect(Box1.line[i],p2); i:=i+1 End;
    Box1.aant:=i
  End;  {HorPlak}
```

Vertikaal plakken

Vertikaal plakken heet bij ons het proces waarbij één regel onder aan een box moet worden toegevoegd, uiteraard gescheiden van die box met een horizontale lijn. Deze regel is de wortel van de boom, de box bevat de takken die uit die wortel ontspringen.

Het grootste probleem dat we hier tegenkomen is dat de "worteltekst" mogelijk breder is dan de box die erboven komt. In dat geval moeten we de box oprekken. We eisen als preconditie dat de box niet leeg is.

```
Procedure RekOp(Var Box:BoxType; l:Byte);
  {Pre: Box.aant>0}
  Var x,y:Byte;
  Begin {RekOp}
    With Box Do For y:=0 To aant-1 Do
    Begin
      For x:=1 To l div 2 Do If line[y][1]=hor
        Then line[y]:=hor+line[y]
        Else line[y]:=' '+line[y];
      For x:=1 To l div 2 + l mod 2 Do If line[y][Length(line[y])]=hor
        Then line[y]:=line[y]+hor
        Else line[y]:=line[y]+' '
    End
  End;  {RekOp}
```

Daarna moet de (niet lege) box onderstreept worden. Gewoon een horizontale lijn trekken kan niet, de lijn moet mooi aansluiten op vertikale lijnen:

```
Procedure Underline(Var Box:BoxType);
  {Pre: Box.aant>0}
  Var i:Byte;
  Begin {Underline}
    Box.line[Box.aant]:=Box.line[Box.aant-1];
    With Box Do For i:=1 To Length(line[0]) Do If line[aant][i]=ver
      Then line[aant][i]:=horup
      Else line[aant][i]:=hor;
```

```
    Box.aant:=Box.aant+1
  End;  {Underline}
```

Tenslotte moet de wortel zelf nog onder de box worden gevoegd. Gecentreerd uiteraard.

```
Function Center(s:RegelType; l:Byte) :RegelType;
  {l>=Length(s)}
  Var i:Byte;
  Begin {Center}
    For i:=1 To (l-Length(s)) div 2 Do s:=' '+s;
    For i:=Length(s)+1 To l Do s:=s+' '; Center:=s
  End;  {Center}
```

Wie denkt dat de procedure `VerPlak` gewoon de concatenatie van bovenstaande drie procedures is, ziet één klein probleem over het hoofd. De box wordt zomaar twee regels langer! Misschien treedt er wel een "box overflow" op.

```
Procedure VerPlak(Var Box:BoxType; Wortel:RegelType);
  { Box.Aant>0 }
  Begin {VerPlak}
    If Length(Box.line[0]) < Length(Wortel)
      Then RekOp(Box,Length(Wortel)-Length(Box.line[0]));
    If Box.aant=MaxLine
      Then Writeln('Sorry, maximale diepte overschreden')
      Else UnderLine(Box);
    With Box Do If aant=MaxLine
      Then Writeln('Sorry, maximale diepte overschreden')
      Else Begin line[aant]:=Center(Wortel,Length(line[0])); aant:=aant+1 End
   End;  {VerPlak}
```

Treedt er een box overflow op, dan gaat het samenstellen van de boxen gewoon door, in horizontale zin tenminste. De uitvoer is dus niet echt incorrect, hoogstens incorrect door onvolledigheid.

Waarom controleren we niet op box overflow bij het horizontaal plakken? Om de eenvoudige reden dat strings automatisch worden afgekapt op hun maximale lengte, *zonder* een run-time error te genereren (dit in tegenstelling tot het indexeren van een array buiten zijn domein). En het resultaat bij zo'n ongecontroleerde horizontale overflow is hetzelfde als bij de gecontroleerde vertikale overflow: een stuk box wordt onttrokken aan het oog.

7.3.5 Structuur naar box

Dan zijn we nu in staat de recursieve procedure te ontwerpen die een structuur op zijn ontleedschema afbeeldt. (Helaas moet het een procedure zijn in plaats van een (veel meer voor de hand liggend) functie daar het resultaat type geen basis type is). We maken gebruik van de ontleed truc uit 7.3.2. Twee hulp functies bepalen of een karakter een open of sluithaak is:

```
Function OpenHaak(Ch:Char):Boolean;
  Begin {OpenHaak}
    OpenHaak:=(Ch='[') or (Ch='(') or (Ch='<') or (Ch='{')
  End;  {OpenHaak}

Function SluitHaak(Ch:Char):Boolean;
  Begin {SluitHaak}
    SluitHaak:=(Ch=']') or (Ch=')') or (Ch='>') or (Ch='}')
  End;  {SluitHaak}
```

Door te tellen hoeveel haakjes geopend en gesloten zijn weten we op welk haakniveau we zijn. Is dit niveau 1, dan zijn we de label van de wortel aan het lezen, is dit niveau 2 of hoger, dan zijn we een tak aan het lezen.

```
Procedure StrucToBox(zin:Struc; Var Box:BoxType);
  (* Pre : zin is goedhaaks,
           zin is dus van de vorm (E), [E], <E> of {E} en E is goedhaaks.
     Post: box bevat de blokstructuur van zin,
           box is dus niet leeg (want zin is niet leeg).
  *)
  Var i,h:Word; root,branch:Struc; NewBox:BoxType;
  Begin {StrucToBox}
    h:=0; root:=''; branch:=''; Box.Aant:=0; {Box is dus leeg}
    For i:=1 To Length(zin) Do
    Begin
      If OpenHaak(zin[i]) Then h:=h+1;
      If h=1 Then root:=root+zin[i] Else branch:=branch+zin[i];
      If SluitHaak(zin[i])
      Then Begin
        h:=h-1; If h=1 Then
          Begin StrucToBox(branch,NewBox); HorPlak(Box,NewBox); branch:='' End
      End
    End;
    root:=Copy(root,2,Length(root)-2);
    If Box.Aant=0
      Then Begin Box.line[Box.aant]:=root; Box.aant:=1 End
      Else VerPlak(Box,root)
  End;  {StrucToBox}
```

7.3.6 Afdrukken box

We hebben nu een `Struc` naar een box getransformeerd, we moeten de box nog wel printen. En, met een kader eromheen. Dus een horizontale lijn erboven en eronder en een vertikale lijn links en rechts. We beginnen met het plaatsen van de vertikale lijnen. Hierbij moeten we natuurlijk rekening houden met de in de box aanwezige horizontale lijnen. Per boxregel voegen we lijnstukjes toe met behulp van

```
Function Sides(line:String):String;
 Begin {Sides}
   If line=''
   Then Sides:=ver+ver
   Else Begin
     If line[1]=hor Then line:=verrt+line Else line:=ver+line;
     If line[Length(line)]=hor Then line:=line+verlt Else line:=line+ver;
     Sides:=line
   End
 End;  {Sides}
```

Voor de boven- en onderkant hebben we ook het aansluit probleem, nu met de vertikale lijnen. We geven als parameter mee de bovenste respectievelijk de onderste box regel plus een parameter die aangeeft of er een T-stukje naar boven of naar beneden geplaatst moet worden:

```
Function HorPattern(line:RegelType; Ch:Char):String;
  Var i:Byte;
  Begin {HorPattern}
    For i:=1 To Length(Line) Do
      If Line[i]=ver Then Line[i]:=Ch Else Line[i]:=hor;
    HorPattern:=Line
  End;  {HorPattern}
```

De procedure die het ontleedschema van een `Struc` print is dan

```
Procedure PrintBox(zin1:RauweTekst; zin2:Struc);
  Var Box:BoxType; i:Byte;
  Begin {PrintBox}
    StrucToBox(WoordenInvullen(zin1,zin2),Box);
    With Box Do
    Begin
      Meld(uplt+HorPattern(line[0],hordn)+uprt);
      For i:=0 To Box.aant-1 Do Meld(Sides(Box.line[i]));
      Meld(dnlt+HorPattern(line[aant-1],horup)+dnrt)
    End
  End;  {PrintBox}
```

7.4 Het programma

Het hoofdprogramma leest een "rauwe zin" in en voert de behandelde zin zelf, diens structuur en het bijbehorende ontleedschema weer uit. We hebben dus nog een procedure nodig die een zin inleest (`LeesRauwe`), een die de behandelde zin uitvoert (`PrintBehandelde`) en een die de structuur uitvoert (`PrintStruc`).

```
Procedure LeesRauwe(Var zin:RauweTekst);
  Begin {LeesRauwe}
    Meld('================================================================================');
    Meld('Voer een substantiefgroep in. (alleen RETURN om te stoppen)');
    ReadLn(zin); Meld(zin) {Voor in de LOG-file}
  End;  {LeesRauwe}

Procedure PrintBehandelde(n:Word; zin:BehandeldeTekst);
  Var nummer, tekst:String; i:Byte;
  Begin {PrintBehandelde}
    Str(n:6,nummer);
    Meld(nummer+'. met woordtypes');
    tekst:='       '; i:=1;
    While i<=Length(zin) Do
      If Length(tekst+' '+CategorieNaam[CharToCat(zin[i])]) < 80
      Then Begin tekst:=tekst+' '+CategorieNaam[CharToCat(zin[i])]; i:=i+1 End
      Else Begin Meld(Tekst); tekst:='       ' End;
    Meld(tekst)
  End;  {PrintBehandelde}

Procedure PrintStruc(n0,n1:Word; zin:BehandeldeTekst);
  Var nummer0,nummer1:String;
  Begin {PrintStruc}
    Str(n0:2,nummer0); Str(n1:3,nummer1);
    Meld(nummer0+'.'+nummer1+'. met structuur');
    Meld(zin)
  End;  {PrintStruc}
```

De complete source van de userinterface vindt u in appendix F. De sourcecode van het hoofdprogramma in appendix G.

Bijlage A

Module Data

```
Unit Data;

Interface

  Type
    RauweTekst     = String;
    BehandeldeTekst= String;
    Categorie      =(Bijwoord,Telwoord,DeWoord,HetWoord,MVwoord,Naam
                    ,EVvoorzetsel,MVvoorzetsel,Voegwoord
                    ,AdjectiefMetE3,AdjectiefZonderE3,AdjectiefOnverbogen3
                    ,AdjectiefMetE2,AdjectiefZonderE2,AdjectiefOnverbogen2
                    ,AdjectiefMetE1,AdjectiefZonderE1,AdjectiefOnverbogen1
                    ,DeBepaling,HetBepaling,DeHetBepaling,EenBepaling
                    ,EndMarker {Speciaal symbool voor recursive descent parser}
                    );
    Row            =^RowNode;
    RowNode        = Record
                       z     : BehandeldeTekst;
                       next  : Row
                     End;
    Bepaling       = (de_bepaling, het_bepaling, dehet_bepaling, een_bepaling);
    Verbuiging     = (met_e, zonder_e, onverbogen);
    Woord          = (de_woord, het_woord, mv_woord);
    Getal          = (meervoud, enkelvoud);
    Prioriteit     = 1..3;
    Struc          = String;
    Strucs         =^StrucNode;
    StrucNode      = Record
                       s    : Struc;
                       g    : Getal;
                       next : Strucs
                     End;


  Const
    EmptyRow       : Row = Nil;
    CategorieNaam  : Array[Categorie] Of String[20]=
                     ('Bijwoord','Telwoord','DeWoord','HetWoord','MVwoord','Naam'
                     ,'EVvoorzetsel','MVvoorzetsel','Voegwoord'
```

```
                    ,'AdjectiefMetE3','AdjectiefZonderE3','AdjectiefOnverbogen3'
                    ,'AdjectiefMetE2','AdjectiefZonderE2','AdjectiefOnverbogen2'
                    ,'AdjectiefMetE1','AdjectiefZonderE1','AdjectiefOnverbogen1'
                    ,'DeBepaling','HetBepaling','DeHetBepaling','EenBepaling'
                    ,'DUMMY' {dummy waarde, zie Type Categorie}
                    );
    EmptySet       : Strucs = Nil;

  Function  CatToChar(Cat:Categorie):Char;
  Function  CharToCat(Ch:Char):Categorie;
  Procedure Isoleer(Var zin:RauweTekst; Var Woord:RauweTekst);
  Procedure Rest(Var zin:BehandeldeTekst);

  Function  SingleRow(zin:BehandeldeTekst):Row;
  Function  Concat(r1,r2:Row):Row;
  Function  Head(r:Row):BehandeldeTekst;
  Function  Tail(r:Row):Row;

  Procedure AddElm(Var o:Strucs; s:Struc; g:Getal);
  Function  Union(o1,o2:Strucs):Strucs;
  Procedure PickElm(Var o:Strucs; Var oo:Struc);

Implementation

  Function CatToChar(Cat:Categorie):Char;
    Begin {CatToChar}
      CatToChar:=Char(Cat)
    End;  {CatToChar}

  Function CharToCat(Ch:Char):Categorie;
    Begin {CharToCat}
      CharToCat:=Categorie(Ch)
    End;  {CharToCat}

  Procedure Isoleer(Var zin:RauweTekst; Var Woord:RauweTekst);
    Var p:Byte;
    Begin {Isoleer}
      While (Length(zin)>0) and (zin[1]=' ') Do zin:=Copy(zin,2,Length(zin)-1);
      p:=Pos(' ',zin+' ')-1; Woord:=Copy(zin,1,p); Delete(zin,1,p)
    End;  {Isoleer}

  Procedure Rest(Var zin:BehandeldeTekst);
    Begin {Rest}
      zin:=Copy(zin,2,Length(zin)-1)
    End;  {Rest}

  Function SingleRow(zin:BehandeldeTekst):Row;
    Var r:Row;
    Begin {SingleRow}
      New(r); r^.z:=zin; r^.next:=EmptyRow; SingleRow:=r
    End;  {SingleRow}

  Function Concat(r1,r2:Row):Row;
    Var r:Row;
```

```
    Begin {Concat}
      If r1=EmptyRow
      Then Concat:=r2
      Else Begin
        r:=r1; While r^.next <> EmptyRow Do r:=r^.next;
        r^.next:=r2; Concat:=r1
      End
    End;  {Concat}

  Function Head(r:Row):BehandeldeTekst;
    {Pre: r<>EmptyRow}
    Begin {Head}
      Head:=r^.z
    End;  {Head}

  Function Tail(r:Row):Row;
    {Pre: r<>EmptyRow}
    Begin {Tail}
      Tail:=r^.next; Dispose(r)
    End;  {Tail}

  Procedure AddElm(Var o:Strucs; s:Struc; g:Getal);
    Var h:Strucs;
    Begin {AddElm}
      New(h); h^.s:=s; h^.g:=g; h^.next:=o; o:=h
    End;  {AddElm}

  Function Union(o1,o2:Strucs):Strucs;
    Var o:Strucs;
    Begin {Union}
      If o1=EmptySet
      Then Union:=o2
      Else Begin
        o:=o1; While o^.next <> EmptySet Do o:=o^.next;
        o^.next:=o2; Union:=o1
      End
    End;  {Union}

  Procedure PickElm(Var o:Strucs; Var oo:Struc);
    {Pre: o<>EmptySet}
    Var h:Strucs;
    Begin {PickElm}
      oo:=o^.s; h:=o^.next; Dispose(o); o:=h
    End;  {PickElm}

End.  {Data}
```

Bijlage B

Module Uitvoer

```
Unit Uitvoer;

Interface

  Procedure Meld(Tekst:String);
  Procedure Error(Fout:String);

Implementation

  Var
    OldExitProc : Pointer;
    WriteLog    : Boolean;
    LogFile     : Text;

  Procedure Error(Fout:String);
    Begin {Error}
      Meld('Er is een fatale fout opgetreden;'); Meld(Fout); Meld(''); Halt
    End;  {Error}

  {$F+} {Zet de FAR CALL mode aan}
  Procedure CloseLogFile;
    Begin {CloseLogFile}
      If WriteLog Then Close(LogFile); ExitProc:=OldExitProc
    End;  {CloseLogFile}
  {$F-} {en weer uit}

  Procedure Meld(Tekst:String);
    Begin {Meld}
      Writeln(Tekst);
      If WriteLog
      Then Begin
        {$I-} Writeln(LogFile,Tekst); {$I+}
        If IoResult<>0 Then Writeln('Disk(ette) is vol of andere disk-fout.')
      End
    End;  {Meld}

Begin {UitVoer}
  If ParamCount=1
  Then Begin
```

```
    {$I-} Assign(LogFile,ParamStr(1)); Rewrite(LogFile); {$I+}
    WriteLog:=IoResult=0
  End
  Else WriteLog:=False;
  OldExitProc:=ExitProc; ExitProc:=@CloseLogFile
End.  {UitVoer}
```

Bijlage C

Module Lexicon

```
Unit Lexicon;

Interface

  Uses
    Data,
    Uitvoer;

  Procedure AddToLexicon(woord:RauweTekst; cat:Categorie);
  Function  WoordTypes(woord:RauweTekst):BehandeldeTekst;

Implementation

  Const
    MaxWoordLen    = 20;
    MaxAantWoorden = 2500;

  Type
    WoordType      = String[MaxWoordLen];
    WoordIndex     = 0..MaxAantWoorden-1;
    WoordRange     = 0..MaxAantWoorden;
    WoordLijst     = Array[WoordIndex] Of Record
                       woord : WoordType;
                       cat   : Categorie
                     End;

  Var
    Lijst          : WoordLijst;
    AantalWoorden  : WoordRange;

  Procedure AddToLexicon(woord:RauweTekst; cat:Categorie);

    Procedure ControleerWoord(woord:RauweTekst);
      Var i:1..MaxWoordLen;
      Begin {Controleerwoord}
        If Length(woord)>MaxWoordLen Then Error('Woord te lang: '+woord);
        If Length(woord)=0 Then Error('Een woord ter lengte nul.');
        For i:=1 To Length(woord) Do If Not(woord[i] in ['a'..'z','A'..'Z'])
          Then Error('Illegaal karakter in '+woord)
```

```
      End;  {Controleerwoord}

    Procedure Insert(woord:WoordType; cat:Categorie);
      Var i:WoordRange;
      Begin {Insert}
        i:=AantalWoorden;
        While (i>0) and (woord<Lijst[i-1].woord) Do
          Begin Lijst[i]:=Lijst[i-1]; i:=i-1 End;
        Lijst[i].woord:=woord; Lijst[i].cat:=cat;
        AantalWoorden:=AantalWoorden+1
      End;  {Insert}

    Begin {AddToLexicon}
      ControleerWoord(woord);
      If AantalWoorden=MaxAantWoorden
        Then Error('Lexicon is vol')
        Else Insert(woord,cat)
    End;  {AddToLexicon}

  Function WoordTypes(woord:RauweTekst):BehandeldeTekst;
    Var i,j:WoordRange; Types:BehandeldeTekst; h:Integer;
    Begin {WoordTypes}
      i:=0; j:=AantalWoorden;
      While i+1 <> j Do
      Begin
        h:=(i+j) div 2;
        If Lijst[h].woord<=woord Then i:=h Else j:=h
      End;
      Types:=''; h:=i;
      While (h>=0) and (Lijst[h].woord=woord) Do
        Begin Types:=Types+CatToChar(Lijst[h].cat); h:=h-1 End;
      WoordTypes:=Types;
    End;  {WoordTypes}

Begin {Lexicon}
  AantalWoorden:=0;
End.  {Lexicon}
```

Bijlage D

Module Scanner

```
Unit Scanner;

Interface

  Uses
    Data,
    Uitvoer;

  Procedure InitLexicon;
  Function  LexicalScanner(zin:RauweTekst):Row;

Implementation

  Uses
    Lexicon;

  Procedure InitLexicon;

    Var Bestand : Text;

    Procedure ReadError(msg:String);
      Begin {ReadError}
        Close(Bestand); Error(msg)
      End;  {ReadError}

    Procedure Leesregel(Var regel:RauweTekst);
      Begin {LeesRegel}
        If Eof(Bestand)
          Then ReadError('Onverwacht einde van het bestand. "***" vergeten?');
        ReadLn(Bestand,regel);
        If Length(regel)>254
          Then ReadError('Regel langer dan 254 karakters.')
      End;  {LeesRegel}

    Procedure ControleerCategorie(regel:RauweTekst; Cat:Categorie);
      Begin {ControleerCategorie}
        If regel=CategorieNaam[Cat]
          Then Meld('Inlezen van '+CategorieNaam[Cat])
          Else ReadError('"'+CategorieNaam[Cat]+'" verwacht.')
```

```
      End;  {ControleerCategorie}

    Var
      Cat   : Categorie;
      regel : RauweTekst;
    Begin {InitLexicon}
      {$I-} Assign(Bestand,'LEXICON.TXT'); Reset(Bestand); {$I+}
      If IoResult<>0 Then Error('lexicon bestand LEXICON.TXT niet gevonden');
      For Cat:=BijWoord To EenBepaling Do
      Begin
        Leesregel(regel); ControleerCategorie(regel,cat);
        Repeat
          Leesregel(regel);
          If regel<>'***' Then AddToLexicon(regel,cat)
        Until regel='***'
      End;
      If Not Eof(Bestand) Then ReadError('Einde bestand verwacht.');
      Close(Bestand)
    End;  {InitLexicon}

  Function LexicalScanner(zin:RauweTekst):Row;
    Function Scan(kop:BehandeldeTekst; staart:RauweTekst):Row;
      Var woord:RauweTekst; types:BehandeldeTekst; r:Row; p:Byte;
      Begin {Scan}
        Isoleer(staart,woord);
        If woord=''
        Then Scan:=SingleRow(kop)
        Else Begin
          types:=WoordTypes(woord); r:=EmptyRow;
          If types='' Then Meld('"'+woord+'" is niet opgenomen in de woordenlijst');
          For p:=1 To Length(types) Do r:=ConCat(Scan(kop+types[p],staart),r);
          Scan:=r
        End
      End;  {Scan}
    Begin {LexicalScanner}
      LexicalScanner:=Scan('',zin)
    End;  {LexicalScanner}

Begin {Scanner}
  InitLexicon
End.  {Scanner}
```

Bijlage E

Module Parser

```
Unit Parser;

Interface

  Uses
    Data;

  Procedure ParseSubstantiefGroep(zin:BehandeldeTekst; Var o:Strucs);

Implementation

  Type
    Filter = Function (a,b:Getal):Boolean;
    Mapper = Function (a,b:Getal):Getal;

  {$F+} {Dwing FAR CALLS af. Moet voor functies als parameter}
  Function P1(a,b:Getal):Boolean; Begin P1:=True End;
  Function f1(a,b:Getal):Getal;   Begin f1:=meervoud End;
  Function P2(a,b:Getal):Boolean; Begin P2:=True End;
  Function f2(a,b:Getal):Getal;   Begin f2:=a End;
  Function P3(a,b:Getal):Boolean; Begin P3:=b=meervoud End;
  Function f3(a,b:Getal):Getal;   Begin f3:=a End;
  {$F-}

  Procedure AddElm(Var o:Strucs; s:Struc; g:Getal);
    Var h:Strucs;
    Begin {AddElm}
      New(h); h^.s:=s; h^.g:=g; h^.next:=o; o:=h
    End;  {AddElm}

  Function Union(o1,o2:Strucs):Strucs;
    Var o:Strucs;
    Begin {Union}
      If o1=EmptySet
      Then Union:=o2
      Else Begin
        o:=o1; While o^.next <> EmptySet Do o:=o^.next;
        o^.next:=o2; Union:=o1
      End
```

```
  End;  {Union}

Function Prod(o1,o2:Strucs; f:Mapper; P:Filter; alfa,beta,gamma:Struc):Strucs;
   (* Preconditie : True
          Return : { (alfa x1 beta x2 gamma, f(y1,y2))
                   | (x1,y1) in o1   and   (x2,y2) in o2   and   P(y1,y2)
                   }
   *)
  Procedure DisposeSet(o:Strucs);
    Var h:Strucs;
    Begin {DisposeSet}
      While o<>EmptySet Do Begin h:=o^.next; Dispose(o); o:=h End
    End;  {DisposeSet}

  Var h1,h2,h3:Strucs;
  Begin {Prod}
    h3:=EmptySet;
    h1:=o1;
    While h1 <> EmptySet Do
    Begin
      h2:=o2;
      While h2 <> EmptySet Do
      Begin
        If P(h1^.g,h2^.g)
          Then AddElm(h3,alfa+h1^.s+beta+h2^.s+gamma,f(h1^.g,h2^.g));
        h2:=h2^.next
      End;
      h1:=h1^.next
    End;
    DisposeSet(o1); DisposeSet(o2);
    Prod:=h3
  End;  {Prod}

Procedure ParseSubstantiefGroep(zin:BehandeldeTekst; Var o:Strucs);

  Procedure ParseSubstantiefGroep1(zin:BehandeldeTekst; Var o:Strucs);

    Procedure ParseKernGroep(zin:BehandeldeTekst; Var g:Getal;
                             Var oo:Struc; Var Ok:Boolean);

      Procedure ParseVoorBepaling(Var zin:BehandeldeTekst; Var b:Bepaling;
                                  Var Ok:Boolean);
        Begin {ParseVoorBepaling}
          Ok:=True;
          Case CharToCat(zin[1]) Of
            DeBepaling    : Begin Rest(zin); b:=de_bepaling    End;
            HetBepaling   : Begin Rest(zin); b:=het_bepaling   End;
            DeHetBepaling : Begin Rest(zin); b:=dehet_bepaling End;
            EenBepaling   : Begin Rest(zin); b:=een_bepaling   End
            Else            Ok:=False
          End
        End;  {ParseVoorBepaling}

      Procedure ParseAdjectiefGroep(Var zin:BehandeldeTekst; Var v:Verbuiging;
```

```
                          Var s:Struc; Var Ok:Boolean);

  Procedure ParseAdjectief(Var zin:BehandeldeTekst; Var v:Verbuiging;
                           Var p:Prioriteit; Var Ok:Boolean);
    Begin {ParseAdjectief}
      Ok:=True;
      Case CharToCat(zin[1]) Of
        AdjectiefMetE3       : Begin Rest(zin); v:=met_e; p:=3 End;
        AdjectiefZonderE3    : Begin Rest(zin); v:=zonder_e; p:=3 End;
        AdjectiefOnverbogen3 : Begin Rest(zin); v:=onverbogen; p:=3 End;
        AdjectiefMetE2       : Begin Rest(zin); v:=met_e; p:=2 End;
        AdjectiefZonderE2    : Begin Rest(zin); v:=zonder_e; p:=2 End;
        AdjectiefOnverbogen2 : Begin Rest(zin); v:=onverbogen; p:=2 End;
        AdjectiefMetE1       : Begin Rest(zin); v:=met_e; p:=1 End;
        AdjectiefZonderE1    : Begin Rest(zin); v:=zonder_e; p:=1 End;
        AdjectiefOnverbogen1 : Begin Rest(zin); v:=onverbogen; p:=1 End
        Else                   Ok:=False
      End
    End;  {ParseAdjectief}

  Var
    M, N                       : Integer;
    DezeV                      : Verbuiging;
    DezePrio, Prio             : Prioriteit;
    MetFound,ZonderFound,AdjOk : Boolean;
  Begin {ParseAdjectiefGroep}
    s:=''; M:=0;
    While CharToCat(zin[1])=BijWoord Do
      Begin Rest(Zin); M:=M+1; s:=s+'(Bijw<>)' End;
    N:=0; Prio:=3; MetFound:=False; ZonderFound:=False; Ok:=True;
    While CharToCat(zin[1]) in
      [AdjectiefMetE3,AdjectiefZonderE3,AdjectiefOnverbogen3
      ,AdjectiefMetE2,AdjectiefZonderE2,AdjectiefOnverbogen2
      ,AdjectiefMetE1,AdjectiefZonderE1,AdjectiefOnverbogen1] Do
    Begin
      ParseAdjectief(zin,DezeV,DezePrio,AdjOk); s:=s+'(Adj<>)'; N:=N+1;
      Ok:=Ok and AdjOk and (Prio>=DezePrio); Prio:=DezePrio;
      MetFound:=MetFound or (DezeV=met_e);
      ZonderFound:=ZonderFound or (DezeV=zonder_e)
    End;
    If s<>'' Then s:='(AdjGr'+s+')';
    Ok:=Ok and ((M=0) or ((M=1) and (N>0))) and
        Not(MetFound and ZonderFound);
    If MetFound
      Then v:=met_e
      Else If ZonderFound
        Then v:=zonder_e
        Else v:=onverbogen
  End;  {ParseAdjectiefGroep}

Procedure ParseSubstantief(Var zin:BehandeldeTekst; Var w:woord; Var Ok:Boolean);
  Begin {ParseSubstantief}
    Ok:=True;
    Case CharToCat(zin[1]) Of
```

```
        DeWoord  : Begin Rest(zin); w:=de_woord  End;
        HetWoord : Begin Rest(zin); w:=het_woord End;
        MVwoord  : Begin Rest(zin); w:=mv_woord  End
        Else       Ok:=False
      End
    End;  {ParseSubstantief}

  Var M:Integer; ss:Struc; v:Bepaling; a:Verbuiging; s:Woord;
      BepOk, AGrOk, SubOk : Boolean;
  Begin {ParseKernGroep}
    zin:=zin+CatToChar(EndMarker);
    ParseVoorBepaling(zin,v,BepOk); oo:='(Bep<>)';
    M:=0; While CharToCat(zin[1])=Telwoord Do
      Begin Rest(zin); oo:=oo+'(Telw<>)'; M:=M+1 End;
    ParseAdjectiefGroep(zin,a,ss,AGrOk); oo:=oo+ss;
    ParseSubstantief(zin,s,SubOk); oo:='(Kern'+oo+'(Sub<>))';
    Ok:=BepOk and AGrOk and SubOk and (zin=CatToChar(EndMarker)) and
        (
        ( (s=mv_woord) and
          ((a=met_e) or (a=onverbogen)) and
          ((v=de_bepaling) or (v=dehet_bepaling)) and
          ((M=0) or (M=1))
        ) or
        ( (s=de_woord) and
          ((a=met_e) or (a=onverbogen)) and
          ((v=de_bepaling) or (v=dehet_bepaling)) and
          (M=0)
        ) or
        ( (s=het_woord) and
          ((a=met_e) or (a=onverbogen)) and
          ((v=het_bepaling) or (v=dehet_bepaling)) and
          (M=0)
        ) or
        ( (s=de_woord) and
          ((a=met_e) or (a=onverbogen)) and
          (v=een_bepaling) and
          (M=0)
        ) or
        ( (s=het_woord) and
          ((a=zonder_e) or (a=onverbogen)) and
          (v=een_bepaling) and
          (M=0)
        )
        );
    If s=mv_woord Then g:=meervoud Else g:=enkelvoud
  End;  {ParseKernGroep}

Var g:Getal; oo:Struc; Ok:Boolean;
Begin {ParseSubstantiefGroep1}
  o:=EmptySet;
  ParseKernGroep(zin,g,oo,Ok);
  If Ok Then AddElm(o,'[SubGr'+oo+']',g)
End;  {ParseSubstantiefGroep1}
```

```
    Procedure ParseSubstantiefGroep2(zin:BehandeldeTekst; Var o:Strucs);
      Begin {ParseSubstantiefGroep2}
        o:=EmptySet;
        If Zin=CatToChar(Naam) Then AddElm(o,'[SubGr(Naam<>)]',enkelvoud)
      End;  {ParseSubstantiefGroep2}

    Procedure ParseSubstantiefGroep3(zin:BehandeldeTekst; f:Mapper; P:Filter;
                                     Cat:Categorie; alfa,beta,gamma:Struc;
                                     Var o:Strucs);
      Var i:Word; deel1,deel2:BehandeldeTekst; o1,o2:Strucs;
      Begin {ParseSubstantiefGroep3}
        deel1:=''; deel2:=zin; i:=Pos(CatToChar(Cat),deel2); o:=EmptySet;
        While i>0
        Do Begin
          deel1:=deel1+Copy(deel2,1,i-1); deel2:=Copy(deel2,i+1,Length(deel2)-i);
          ParseSubstantiefGroep(deel1,o1); ParseSubstantiefGroep(deel2,o2);
          deel1:=deel1+CatToChar(Cat);
          o:=Union(o,Prod(o1,o2,f,P,alfa,beta,gamma));
          i:=Pos(CatToChar(Cat),deel2)
        End
      End;  {ParseSubstantiefGroep3}

    Var o1,o2,o3,o4,o5:Strucs;
    Begin {ParseSubstantiefGroep}
      ParseSubstantiefGroep1(zin,o1);
      ParseSubstantiefGroep2(zin,o2);
      ParseSubstantiefGroep3(zin,f1,P1,Voegwoord,'[SubGr','(Vw<>)',']',o3);
      ParseSubstantiefGroep3(zin,f2,P2,EVvoorzetsel,'[SubGr','(VzCon(Vz<>)',')]',o4);
      ParseSubstantiefGroep3(zin,f3,P3,MVvoorzetsel,'[SubGr','(VzCon(Vz<>)',')]',o5);
      o:=Union(o1,Union(o2,Union(o3,Union(o4,o5))))
    End;  {ParseSubstantiefGroep}

End.  {Parser}
```

Bijlage F

Module User

```
Unit User;

Interface

  Uses
    Data,
    Uitvoer;

  Procedure LeesRauwe(Var zin:RauweTekst);
  Procedure PrintBehandelde(n:Word; zin:BehandeldeTekst);
  Procedure PrintStruc(n0,n1:Word; zin:BehandeldeTekst);
  Procedure PrintBox(zin1:RauweTekst; zin2:Struc);

Implementation

  Const
    MaxRegelLength = 150;
    MaxLine        = 20;
    hor            = Chr(196); {zoiets als -}
    ver            = Chr(179); {zoiets als |}
    cross          = Chr(197); {zoiets als +}
    hordn          = Chr(194); {zoiets als T}
    horup          = Chr(193); {zoiets als een omgekeerde T}
    verlt          = Chr(180); {zoiets als -|}
    verrt          = Chr(195); {zoiets als |-}
    uplt           = Chr(218); {voor de linker bovenhoek}
    uprt           = Chr(191); {voor de rechter bovenhoek}
    dnlt           = Chr(192); {voor de linker onderhoek}
    dnrt           = Chr(217); {voor de rechter onderhoek}

  Type
    RegelType      = String[MaxRegelLength];
    LineRange      = 0..MaxLine-1;
    LineAant       = 0..MaxLine;
    BoxType        = Record
                       line : Array[LineRange] Of RegelType;
                       aant : LineAant
                     End;
```

```
Procedure LeesRauwe(Var zin:RauweTekst);
  Begin {LeesRauwe}
    Meld('================================================================================');
    Meld('Voer een substantiefgroep in. (alleen RETURN om te stoppen)');
    ReadLn(zin); Meld(zin) {Voor in de LOG-file}
  End;  {LeesRauwe}

Procedure PrintBehandelde(n:Word; zin:BehandeldeTekst);
  Var nummer, tekst:String; i:Byte;
  Begin {PrintBehandelde}
    Str(n:6,nummer);
    Meld(nummer+'. met woordtypes');
    tekst:='       '; i:=1;
    While i<=Length(zin) Do
      If Length(tekst+' '+CategorieNaam[CharToCat(zin[i])]) < 80
      Then Begin tekst:=tekst+' '+CategorieNaam[CharToCat(zin[i])]; i:=i+1 End
      Else Begin Meld(Tekst); tekst:='       ' End;
    Meld(tekst)
  End;  {PrintBehandelde}

Procedure PrintStruc(n0,n1:Word; zin:BehandeldeTekst);
  Var nummer0,nummer1:String;
  Begin {PrintStruc}
    Str(n0:2,nummer0); Str(n1:3,nummer1);
    Meld(nummer0+'.'+nummer1+'. met structuur');
    Meld(zin)
  End;  {PrintStruc}

Procedure PrintStruc(n0,n1:Word; zin:BehandeldeTekst);
  Var nummer0,nummer1:String;
  Begin {PrintStruc}
    Str(n0:3,nummer0); Str(n1,nummer1);
    Meld(nummer0+'.'+nummer1+'. met structuur');
    Meld(zin)
  End;  {PrintStruc}

Function WoordenInvullen(zin:RauweTekst; oo:Struc):Struc;
  Var woord:RauweTekst; Overflow:Boolean;
  Begin {WoordenInvullen}
    Overflow:=False;
    Repeat
      Isoleer(zin,woord);
      Overflow:=Length(oo)+Length(woord)>255;
      If Overflow Then Writeln('Sorry, ik kan niet alle woorden invullen.');
      If (woord<>'') and Not Overflow Then Insert(woord,oo,Pos('<>',oo)+1)
    Until Overflow or (zin='');
    WoordenInvullen:=oo
  End;  {WoordenInvullen}

Procedure HorPlak(Var Box1:BoxType; Box2:BoxType);

  Function Connect(s,t:RegelType):RegelType;
    Var a,b:Char;
    Begin {Connect}
```

```
      If (s='') or (t='')
      Then Connect:=s+t
      Else Begin
        a:=s[Length(s)]; b:=t[1];
        If (a=hor)  and (b=hor)  Then Connect:=s+cross+t;
        If (a<>hor) and (b=hor)  Then Connect:=s+verrt+t;
        If (a=hor)  and (b<>hor) Then Connect:=s+verlt+t;
        If (a<>hor) and (b<>hor) Then Connect:=s+ver+t
      End
    End;  {Connect}

  Function VerPattern(Box:BoxType):RegelType;
    Var i:Byte; Line:RegelType;
    Begin {VerPattern}
      If Box.aant=0 Then Line:='' Else Line:=Box.line[Box.aant-1];
      For i:=1 To Length(Line) Do If Line[i]<>ver Then Line[i]:=' ';
      VerPattern:=Line
    End;  {VerPattern}

  Var i:LineAant; p1,p2:RegelType;
  Begin {HorPlak}
    i:=0; p1:=VerPattern(Box1); p2:=VerPattern(Box2);
    While (i<Box1.Aant) and (i<Box2.Aant) Do
      Begin Box1.Line[i]:=Connect(Box1.Line[i],Box2.line[i]); i:=i+1 End;
    While i<Box2.Aant Do Begin Box1.Line[i]:=Connect(p1,Box2.line[i]); i:=i+1 End;
    While i<Box1.Aant Do Begin Box1.Line[i]:=Connect(Box1.line[i],p2); i:=i+1 End;
    Box1.aant:=i
  End;  {HorPlak}

Procedure VerPlak(Var Box:BoxType; Wortel:RegelType);
  {Pre: Box.aant>0 }
  Procedure RekOp(Var Box:BoxType; l:Byte);
    {Pre: Box.aant>0 }
    Var x,y:Byte;
    Begin {RekOp}
      With Box Do For y:=0 To aant-1 Do
      Begin
        For x:=1 To l div 2 Do If line[y][1]=hor
          Then line[y]:=hor+line[y]
          Else line[y]:=' '+line[y];
        For x:=1 To l div 2 + l mod 2 Do If line[y][Length(line[y])]=hor
          Then line[y]:=line[y]+hor
          Else line[y]:=line[y]+' '
      End
    End;  {RekOp}

  Procedure Underline(Var Box:BoxType);
    {Box.aant>0}
    Var i:Byte;
    Begin {Underline}
      Box.line[Box.aant]:=Box.line[Box.aant-1];
      With Box Do For i:=1 To Length(line[0]) Do If line[aant][i]=ver
        Then line[aant][i]:=horup
        Else line[aant][i]:=hor;
```

```
      Box.aant:=Box.aant+1
    End;  {Underline}

  Function Center(s:RegelType; l:Byte) :RegelType;
    {l>=Length(s)}
    Var i:Byte;
    Begin {Center}
      For i:=1 To (l-Length(s)) div 2 Do s:=' '+s;
      For i:=Length(s)+1 To l Do s:=s+' '; Center:=s
    End;  {Center}

  Begin {VerPlak}
    If Length(Box.line[0]) < Length(Wortel)
      Then RekOp(Box,Length(Wortel)-Length(Box.line[0]));
    If Box.aant=MaxLine
      Then Writeln('Sorry, maximale diepte overschreden')
      Else UnderLine(Box);
    With Box Do If aant=MaxLine
      Then Writeln('Sorry, maximale diepte overschreden')
      Else Begin line[aant]:=Center(Wortel,Length(line[0])); aant:=aant+1 End
  End;  {VerPlak}

Function OpenHaak(Ch:Char):Boolean;
  Begin {OpenHaak}
    OpenHaak:=(Ch='[') or (Ch='(') or (Ch='<') or (Ch='{')
  End;  {OpenHaak}

Function SluitHaak(Ch:Char):Boolean;
  Begin {SluitHaak}
    SluitHaak:=(Ch=']') or (Ch=')') or (Ch='>') or (Ch='}')
  End;  {SluitHaak}

Procedure StrucToBox(zin:Struc; Var Box:BoxType);
  (* Pre : zin is goedhaaks,
           zin is dus van de vorm (E), [E], <E> of {E} en E is goedhaaks.
     Post: box bevat de blokstructuur van zin,
           box is dus niet leeg (want zin is niet leeg).
  *)
  Var i,h:Word; root,branch:Struc; NewBox:BoxType;
  Begin {StrucToBox}
    h:=0; root:=''; branch:=''; Box.Aant:=0; {Box is dus leeg}
    For i:=1 To Length(zin) Do
    Begin
      If OpenHaak(zin[i]) Then h:=h+1;
      If h=1 Then root:=root+zin[i] Else branch:=branch+zin[i];
      If SluitHaak(zin[i])
      Then Begin
        h:=h-1; If h=1 Then
          Begin StrucToBox(branch,NewBox); HorPlak(Box,NewBox); branch:='' End
      End
    End;
    root:=Copy(root,2,Length(root)-2);
    If Box.Aant=0
      Then Begin Box.line[Box.aant]:=root; Box.aant:=1 End
```

```
        Else VerPlak(Box,root)
    End;  {StrucToBox}

  Function Sides(line:String):String;
    Begin {Sides}
      If line=''
      Then Sides:=ver+ver
      Else Begin
        If line[1]=hor Then line:=verrt+line Else line:=ver+line;
        If line[Length(line)]=hor Then line:=line+verlt Else line:=line+ver;
        Sides:=line
      End
    End;  {Sides}

  Function HorPattern(line:RegelType; Ch:Char):String;
    Var i:Byte;
    Begin {HorPattern}
      For i:=1 To Length(Line) Do
        If Line[i]=ver Then Line[i]:=Ch Else Line[i]:=hor;
      HorPattern:=Line
    End;  {HorPattern}

  Procedure PrintBox(zin1:RauweTekst; zin2:Struc);
    Var Box:BoxType; i:Byte;
    Begin {PrintBox}
      StrucToBox(WoordenInvullen(zin1,zin2),Box);
      With Box Do
      Begin
        Meld(uplt+HorPattern(line[0],hordn)+uprt);
        For i:=0 To Box.aant-1 Do Meld(Sides(Box.line[i]));
        Meld(dnlt+HorPattern(line[aant-1],horup)+dnrt)
      End
    End;  {PrintBox}

End.  {User}
```

Bijlage G

Het programma

```
Program SubGr;
{$M 65000,0,650000} {Zwaar Stack en Heap gebruik}

Uses
  Data,
  Uitvoer,
  Scanner,
  Parser,
  User;

Var
  zin    : RauweTekst;
  rij    : Row;
  o      : Strucs;
  oo     : Struc;
  n0, n1 : Integer;

Begin {SubGr}
  Repeat
    LeesRauwe(zin); rij:=LexicalScanner(zin); n0:=0;
    While rij<>EmptyRow Do
    Begin
      n0:=n0+1; n1:=0;
      PrintBehandelde(n0,Head(rij));
      ParseSubstantiefGroep(Head(rij),o);
      While o<>EmptySet Do
      Begin
        n1:=n1+1; PickElm(o,oo); PrintStruc(n0,n1,oo);
        If Length(oo)=255 Then Meld('Vermoeden: structuur te groot');
        PrintBox(zin,oo)
      End;
      rij:=Tail(rij)
    End
  Until zin=''
End.  {SubGr}
```

Bibliografie

[Brandt Corstius] Brandt Corstius, H.,
Computer-taalkunde,
Muiderberg, Coutinho, 1978.

[Dijkstra] Dijkstra, Edsger W. en Feijen, W.H.J.,
Een methode van programmeren,
Den Haag, Academic service, 1984.

[Geerts] Geerts, G., en andere,
Algemene Nederlandse Spraakkunst,
Groningen: Wolters-Noordhoff; Leuven: Wolters, 1984.

[Hemerik] *Syllabus bij het college COMPILERS I*,
Eindhoven, Technische Universiteit Eindhoven, 1985.