Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

S. van Egmond, F.C. Heeman

Inform: prototype van een interactieve formuleverwerker

Informatica/Afdeling Programmatuur Notitie CS-N8604 April

Inform: prototype of an interactive system for formula processing

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Inform: Prototype van een Interactieve Formuleverwerker

Sylvia van Egmond, Frans C. Heeman
Vrije Universiteit Amsterdam

Op het CWI is een project gaande over interactieve tekstverwerking. Doel van dit project is uiteindelijk één systeem te maken, waarmee interactief documenten kunnen worden gemanipuleerd die tekst, tabellen, wiskundige formules en plaatjes kunnen bevatten. Voor onze afstudeeropdracht hebben we een prototype ontworpen en geïmplementeerd, dat wiskundige formules kan zetten.

1. Inleiding

1.1. Beschrijving van de opdracht

Het maken van een document, dat tabellen, wiskundige formules, plaatjes en tekst kan bevatten, is geen eenvoudige zaak. Op het UNIX-systeem gaat dit als volgt:

Voor plaatjes is er een programma dat 'pic' heet. Een ander programma, 'tbl', tekent tabellen. Voor wiskundige formules wordt het programma 'eqn' gebruikt. Deze drie programma's zijn hulpmiddelen van weer een ander programma, 'troff' genaamd: dit laatste programma wordt gebruikt om tekst te verwerken.

De vier programma's werken op een zelfde manier. De gebruiker van zo'n programma moet een soort beschrijving van het gewenste resultaat geven: deze beschrijving moet door een editor als 'vi' (wéér een ander programma) in de computer opgeslagen worden. Het benodigde tekstverwerkings-programma ('pic', 'tbl', 'eqn' of 'troff') gaat met deze beschrijving aan de slag om een getekend resultaat op papier af te leveren (een 'batch'-methode). Wanneer de gebruiker dit resultaat bekijkt, blijkt vaak dat het tekstverwerkings-programma de beschrijving anders heeft opgevat dan de gebruiker dácht dat het programma het zou opvatten. Het resultaat is niet naar haar zin, en de gebruiker moet de beschrijving (ergens) aanpassen. Vervolgens wordt weer een nieuw (beter?) resultaat op papier afgeleverd door het tekstverwerkings-programma.

Deze methode kost veel tijd en papier. Bovendien is het niet eenvoudig om met deze methode te leren werken: de gebruiker moet vijf programma's (vier tekstverwerkings-programma's en één editor) leren gebruiken.

Een alternatief is de volgende methode. Er is één programma dat plaatjes, tabellen, wiskundige formules en tekst kan verwerken. Het programma is zó opgezet, dat ook niet-ervaren gebruikers er snel en gemakkelijk mee kunnen werken. De gebruiker geeft aanwijzingen aan het programma om een bepaald resultaat te bereiken, en het programma laat op ieder moment op het beeldscherm het effect van de aanwijzingen zien (een 'interactieve' methode). Als de gebruiker uiteindelijk tevreden is over het resultaat dat het programma laat zien, kan de gebruiker het resultaat op papier laten afdrukken.

Voordelen van deze aanpak zijn: de gebruiker hoeft maar één programma te leren gebruiken (en dit

Notitie CS-N8604
Centrum voor Wiskunde en Informatica
Kruislaan 413, 1098 SJ Amsterdam

wordt geacht makkelijk te zijn), het resultaat is op ieder moment te zien, het gewenste resultaat wordt sneller bereikt en het verspilt geen papier.

Er is maar één nadeel. Dit nadeel is niet van belang voor de gebruiker maar wél voor degene die het programma moeten maken (wij dus, b.v.): het is best moeilijk om zo'n programma te maken!

We zijn begonnen met eerst het één en ander te lezen over allerlei tekstverwerkers en andere verwante onderwerpen. Toen hebben we een aantal ideëen uitgewerkt en aan de hand hiervan delen van het programma ontworpen, in de computer gevoerd, uitgetest en gedocumenteerd. Zo groeide gaandeweg het programma: in het begin krabbelde het programma wat vreemde symbolen op het scherm, maar nu kan het indrukwekkende formules tekenen. Van deze groei hebben we een werkverslag bijgehouden, en aan de hand hiervan hebben we deze scriptie geschreven.

1.2. Structuur van de scriptie

De scriptie bestaat uit drie delen: het verslag, de documentatie en het programma zelf. Hoofdstuk 2 van het verslag beschrijft de uitgangspunten van het te maken systeem. In hoofdstuk 3 noemen we de belangrijkste problemen, en beschrijven we de oplossingen en het bereiken van deze oplossingen. Hieruit afgeleid wordt een globaal overzicht van het programma gegeven in hoofdstuk 4. In hoofdstuk 5 vertellen we hoe het programma, geschreven in de programmeertaal 'C' en werkend op een SUN-1 werkstation, er nu voorstaat, en wat er verbeterd en uitgebreid kan worden. Tenslotte staat in hoofdstuk 6 een samenvatting van onze resultaten.

In de documentatie wordt uitvoerig ieder programma-deel in zijn huidige vorm beschreven.

In de hele scriptie worden de volgende notaties gebruikt. Met het woord 'formule' bedoelen we altijd wiskundige formules. Als algemeen voorbeeld van zo'n formule gebruiken we

$$\int_0^1 2x^2 dx = \frac{2}{3}$$

Namen van variabelen in het programma staan tussen enkele aanhalingstekens (b.v. 'active'). Namen van procedures worden ook zo genoteerd en eindigen bovendien met '()' (b.v. 'process()'). Constanten staan in hoofdletters (b.v. INT). Alle namen in het programma zijn in het engels.

Omdat er geen beter systeem tot onze beschikkking was, hebben we deze scriptie gedrukt m.b.v. de programma's 'pic', 'tbl', 'eqn' en 'troff': alle genoemde nadelen van deze systemen hebben we dan ook ondervonden.

2. Doelstellingen

In dit hoofdstuk formuleren we de belangrijkste doelstellingen van het systeem. Eerst beschrijven we de uitgangspunten van het systeem. Met deze uitgangspunten in gedachten vergelijken we dan bestaande tekstverwerkers, die ook wiskundige formules kunnen zetten, met elkaar. We bekijken wat deze tekstverwerkers te bieden hebben en wat ze volgens ons missen. Door deze vergelijking kunnen we de oorspronkelijke uitgangspunten omzetten in een aantal toepasbare doelstellingen.

Vervolgens noemen we nog enkele onderwerpen die met onze opdracht verwant zijn. Deze onderwerpen zijn: syntax-gestuurde editors, attribuut-grammatica's, en het idee van een box.

2.1. Uitgangspunten van het systeem

Het doel van de opdracht is, een systeem te ontwerpen dat interactief wiskundige formules kan zetten. Het systeem moet te gebruiken zijn door zowel mensen met wiskundige kennis (b.v. de auteur van een artikel), als door mensen zonder speciale wiskundige kennis (b.v. een typist).

De auteur schrijft met de hand een artikel. Formules heeft hij ook met de hand getekend. Vervolgens gaat de auteur, of een typist, het artikel in de computer invoeren om het te laten zetten.

Het systeem moet de invoer verwerken en op ieder moment op het scherm het artikel laten zien, zoals het later gezet wordt. Dit is een interactieve werkwijze. Het voordeel van deze werkwijze is, dat de ingevoerde versie op ieder moment vergeleken kan worden met de orginele handgeschreven versie.

Aangezien ook een typist het systeem moet kunnen gebruiken, moet de in- en uitvoer niet in

technische termen gebeuren, maar m.b.v. symbolen. Het systeem kan b.v. als volgt werken:

Het systeem heeft een 'muis'. Met deze muis kan een aanwijs-cursor op het scherm bewogen worden. Op het scherm staat een lijst (een menu) met wiskundige symbolen (b.v. een $\int$). Stel dat een typist een handgeschreven artikel gaat invoeren, en dat hij de volgende symbolen tegenkomt:

$$\int_0^1 2x^2 dx = \frac{2}{3} \tag{2.1.1}$$

De typist zet m.b.v. de muis de aanwijs-cursor op het symbool '$\int$' in het menu, en drukt op een knop op de muis. Op het scherm verschijnt nu het symbool '$\int$', en een andere cursor, de positie-cursor, verschijnt onder het symbool. Volgens de handgeschreven versie moet op deze positie een '0' staan, dus de typist tikt een '0' in. Verder is er geen invoer meer voor deze positie en de typist drukt op de END-toets. Nu komt de positie-cursor boven het symbool '$\int$' te staan, en de typist tikt een '1' in, enz.

De typist ziet dus in de handgeschreven versie symbolen staan, die ook op het scherm te zien zijn. Als de typist zo'n symbool gaat gebruiken, helpt het systeem hem verder. Het systeem moet dus een bepaalde kennis hebben om de gebruiker te helpen. Waar moet deze kennis uit bestaan?

Neem formule (2.1.1). De auteur zal van deze formule een wiskundige beschrijving geven in de trant van: een integraal bestaat uit een integraal-teken ('$\int$'), een ondergrens ('0'), een bovengrens ('1') en een integrand ('$2x^2dx$'). Zo'n beschrijving geeft de opbouw, de syntax, van een formule weer. De typist zal wellicht een andere syntax geven, b.v.: het symbool '$\int$' heeft onder en boven zichzelf weer formules staan. De typist ziet de formule '$2x^2dx$' dan niet als een onderdeel van de integraal, maar als een formule die naast het symbool '$\int$' staat. Blijkbaar is er dus niet precies één syntax.

Behalve een syntax heeft een formule ook een uiterlijke vorm, of lay-out. Deze lay-out noemen we de semantiek van de formule. Een formule heeft niet precies één semantiek. Wanneer formule (2.1.1) in de regel wordt gezet, zoals $\int_0^1 2x^2 dx = \frac{2}{3}$, dan kunnen de grenzen misschien beter rechts naast het integraal-teken staan, om de formule minder hoog te maken.

Er is nu een methode nodig om het systeem kennis te geven over syntax en semantiek van formules. Het idee is, om dit m.b.v. een grammatica te doen: deze grammatica beschrijft een syntax en een semantiek van formules. Het systeem wordt dan gestuurd door deze grammatica. Aangezien syntax en semantiek veranderlijk zijn, zal de grammatica zo onafhankelijk mogelijk van de rest van het systeem moeten zijn. Verder is er een grammatica-mechanisme nodig dat als notatie dient om de grammatica te beschrijven, zoals de BNF-vorm.

In de volgende paragraaf worden enkele bestaande tekstverwerkers bekeken, en formuleren we enkele doelstellingen.

2.2. Bestaande tekstverwerkers

Bekende tekstverwerkers, die ook formules verwerken, zijn o.a. TEX [5] en EQN [3] , de laatste als preprocessor van nroff/troff. Beide systemen hebben een beschrijving van een formule, die gezet moet worden, als invoer. Voor EQN is de beschrijving van de volgende vorm:

```
int from 0 to 1 {2 x sup 2 dx}~=~2 over 3
```

Dit levert de formule:

$$\int_0^1 2x^2 dx = \frac{2}{3}$$

De gebruiker geeft deze beschrijving aan de tekstverwerker. De tekstverwerker produceert het resultaat op papier. Dit soort tekstverwerkers zijn batch-systemen.

Nadeel van deze systemen is, dat de gebruiker een soort taaltje moet kennen om de formule te beschrijven. Het is niet altijd even simpel om van een bepaalde formule de juiste beschrijving te geven. Nog lastiger wordt het voor de gebruiker, wanneer zij een tekst met formules, tabellen, plaatjes, etc. door een batch-systeem moet laten zetten. Vaak gaat de gebruiker in zo'n geval als volgt te werk:

De gebruiker geeft een tekst met lay-out commando's aan de tekstverwerker. Vervolgens bekijkt zij het resultaat op papier, en vergelijkt dit met haar opzet. In het algemeen is het resultaat de eerste keer niet goed. De gebruiker past de beschrijving hier en daar aan, en het proces herhaalt zich. Dit proces is vrij omslachtig en papier-verspillend.

Een ander nadeel van TEX en EQN is, dat ze niet makkelijk uitbreidbaar zijn. De gebruiker kan niet op een makkelijke manier nieuwe wiskundige symbolen of constructies toevoegen.

Een voordeel van deze twee systemen is het hoge niveau van de commando's. De gebruiker hoeft niet te specificeren hoe de formule precies gezet moet worden: beide systemen weten b.v. dat een integraal een ondergrens kan hebben, en waar deze moet worden geplaatst. Met andere woorden, beide systemen hebben een syntax en semantiek van formules. Deze syntax is echter vrij oppervlakkig. Hier volgen een paar voorbeelden van EQN-invoer en de geproduceerde formule:

int from 0 to 1 {2 x sup 2 dx}~=~2 over 3

$$\int_0^1 2x^2 dx = \frac{2}{3}$$ correct

int to 0 from 1 {2 x sup 2 dx}~=~2 over 3

$$\int_1^0 2x^2 dx = \frac{2}{3}$$

int to 0 to 1 {2 x sup 2 dx}~=~2 over 3

$$\int_0^1 2x^2 dx = \frac{2}{3}$$

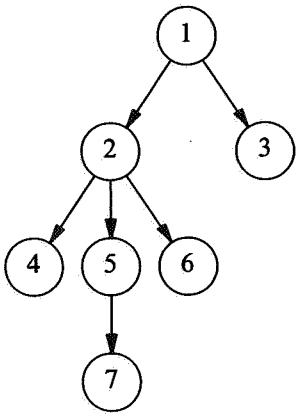
int from 0 from 1 {2 x sup 2 dx}~=~2 over 3

$$\int_1^0 2x^2 dx = \frac{2}{3}$$

Scribe [10] is een mooi voorbeeld van een systeem dat met een syntax van tekst werkt. Scribe is een batch-systeem dat alleen tekst kan verwerken. Het systeem werkt met een databank. In die databank staat de syntax-definitie van tekst voor een tijdschrift X, een intern rapport, enz. Deze syntax-definitie beschrijft de structuur waaraan de ingevoerde tekst moet voldoen. B.v.: de tekst moet beginnen met een titel, en deze wordt gecentreerd, in italic, puntgrootte 20; vervolgens moet een abstract beginnen enz. De gebruiker geeft in de tekst aan: de volgende tekst is een titel; de tekst na de titel is de abstract, enz. De syntax-definitie wordt gebruikt om te controleren of de ingevoerde tekst in een bepaalde vorm staat, en verzorgt een lay-out. De syntax-definitie's zijn geen onderdeel van het programma. Het toevoegen van nieuwe, of het veranderen van bestaande, definitie's kan eenvoudig gebeuren. Bijvoorbeeld iemand met typografische kennis kan dit doen.

Er zijn ook systemen met een syntax van tekst, die interactief werken (b.v. Etude [1]): de gebruiker ziet op ieder moment het document, zoals het later gezet wordt, op het beeldscherm. Als de gebruiker b.v. in een tekst met een nette rechterkantlijn een karakter toevoegt, dan wordt de tekst opnieuw uitgelijnd en door het systeem gezet; deze nieuwe versie wordt weer op het scherm geschreven.

We hebben twee systemen gevonden die interactief werken en een syntax van formules gebruiken. Een editor van Levison [6] en Edimath van Quint [9] zijn zulke systemen. In beide systemen is de syntax en de semantiek echter een vast onderdeel van het systeem: om de syntax of de semantiek te veranderen moet het programma veranderd worden.



FIGUUR 2.1

De syntax bepaalt de structuur van een formule. De schrijvers representeren deze structuur d.m.v. een boom. Een boom bestaat uit knopen en pijlen (zie fig. 2.1). Knoop 1 en knoop 2 zijn verbonden door een pijl. De pijl wijst van knoop 1 naar knoop 2. Knoop 1 is de ouder van knoop 2, en knoop 2 is een kind van knoop 1. Een ouder kan meerdere kinderen hebben (deze zijn broers van elkaar); een kind heeft echter hoogstens één ouder. Heeft een knoop geen kinderen, dan is zo'n knoop een blad (knoop 3 in fig. 2.1). Heeft een knoop geen ouder, dan is zo'n knoop de wortel van de boom (knoop 1 in fig. 2.1).

Beide schrijvers zijn ervan uitgegaan, dat de syntax van een formule meer moet aansluiten bij de 2-dimensionale vorm van een formule, dan bij de wiskundige betekenis. De formule '$y = ax + b$' wordt dus gezien als een string van karakters en niet als een wiskundige vergelijking. De reden hiervoor is, dat ook een niet-wiskundige met het systeem moet kunnen werken. Bovendien was een andere eis, dat het systeem niet onnodig ingewikkeld moest worden.

Een syntax van formules wordt bepaald door de manier waarop een formule wordt opgedeeld in deelformules. Zo'n opdeling wordt gerepresenteerd door de deelformules als kinderen van een ouderknoop te representeren: deze ouderknoop representeert de hele formule. Deelformules worden weer verder opgedeeld, enz.

De schrijvers hebben echter formules op een verschillende manier opgedeeld. We laten dit aan de hand van een voorbeeld zien (fig. 2.2).

Een eerste opvallend verschil is, dat de boom van Quint omvangrijker is dan die van Levison. In de boom van Levison is de formule makkelijker te herkennen.

Een integraal ziet er bij Quint uit als een formule met twee deelformules: de onder- en bovengrens. De integrand is géén deel van de integraal. Machtsverheffen bestaat uit één deelformule: de macht. De grondexpressie hoort níet bij het machtsverheffen. Integraal en machtsverheffen worden bij Quint inderdaad gerepresenteerd in een 2-dimensionale vorm.

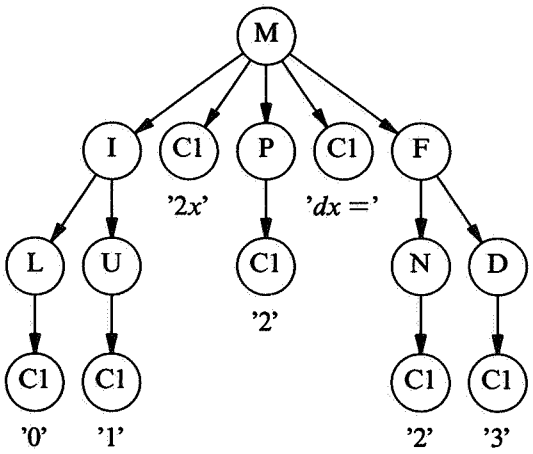
Bij Levison heeft de integraal 3 deelformules (ondergrens, bovengrens en integrand) en machtsverheffen 2 (de grondexpressie en de macht). Dit is dus een meer wiskundige representatie.

Als de gebruiker in de editor van Levison een formule van de vorm 'x^y' wil invoeren, met een expressie op de plaats van 'x', dan gaat dit als volgt: eerst moet de gebruiker het commando voor machtsverheffen geven, dan de expressie 'x', en vervolgens de macht 'y'. Vaak zal de gebruiker het commando voor machtsverheffen "te laat" (d.w.z. ná de grondexpressie) geven, vooral als de expressie 'x' redelijk ingewikkeld is. De gebruiker moet dan met enig kunst en vliegwerk de schade herstellen.

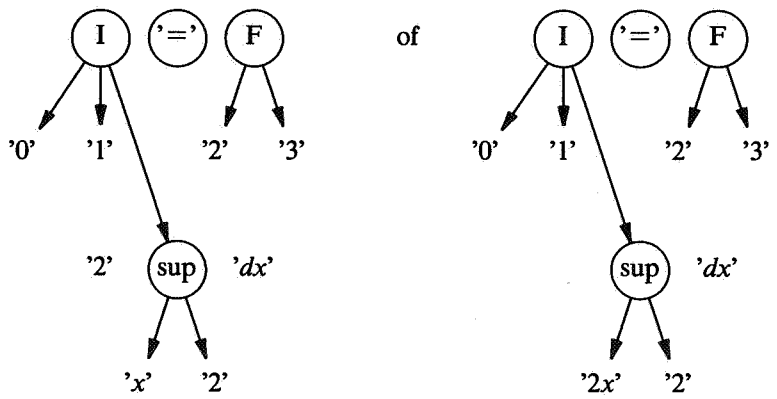
In fig. 2.2 is te zien, dat er bij Levison bovendien 2 mogelijkheden zijn om één en dezelfde formule

$$\int_0^1 2x^2 dx = \frac{2}{3}$$

Quint:



Levison:



FIGUUR 2.2

te representeren: zowel 'x' als '$2x$' kan als grondexpressie van machtsverheffen dienen. Het systeem representeert één formule op twee verschillende manieren, afhankelijk van de volgorde van invoer door de gebruiker.

Beide systemen schenken niet zoveel aandacht aan typografische regels die er voor wiskundige formules zijn ([13, 11]). Bijvoorbeeld: het is gebruikelijk dat variabelen in *Italic* gezet worden, maar de

'd' van de integraal-variabele in '$\int x dx$' en standaardfuncties als 'sin' in Roman. De juiste lay-out van een formule hangt dus af van de wiskundige betekenis. In beide systemen worden variabelen echter in roman gezet, tenzij de gebruiker aangeeft dat een variabele in *Italic* moet.

Tenslotte is het in beide systemen lastig om nieuwe wiskundige symbolen of constructies toe te voegen, hetgeen in de wiskunde vaak gebeurt. De oorzaak van deze moeilijkheid is, dat de syntax van formules in het programma verwerkt is: om de syntax te veranderen moet het programma veranderd worden.

Naar aanleiding van de uitgangspunten van ons systeem (paragraaf 2.1) en de discussie over andere systemen, kunnen we de volgende doelstellingen formuleren:

- het systeem moet interactief werken.
- het systeem moet gestuurd worden door een grammatica, die de syntax en de semantiek van formules beschrijft. De volgende doelstellingen leggen verdere eisen op aan deze grammatica.
- de grammatica moet extern t.o.v. het systeem zijn.
- de grammatica moet eenvoudig van vorm zijn. Samen met de vorige doelstelling is de gebruiker dan in staat nieuwe wiskundige symbolen of constructies toe te voegen. Het systeem zelf moet ook makkelijk met de grammatica kunnen werken.
- de grammatica moet éénduidig het gedrag van het systeem bepalen.
- de lay-out van formules, die het systeem produceert, moet in de grammatica nauwkeurig bepaald kunnen worden, zodat de formules er typografisch gezien goed uitzien.
- het systeem moet gebruiksvriendelijk zijn, zowel voor wiskundige- als niet-wiskundige gebruikers. Dit is een veelomvattende doelstelling: deze doelstelling houdt in dat het systeem zich consequent en voor de gebruiker verklaarbaar gedraagt, dat iedere actie van de gebruiker een reactie van het systeem veroorzaakt, en nog veel meer (psychologische) aspecten.
- het systeem moet terminal-onafhankelijk worden. D.w.z. dat slechts een beperkt deel van het systeem veranderd moet worden als het systeem een ander soort terminal moet gaan gebruiken.

Sommige doelstellingen zijn geen verrassing: wie wil er niet een gebruiksvriendelijk systeem? Andere doelstellingen zijn misschien te hoog gegrepen. Aan het slot van de scriptie zal worden beschreven wat er van de doelstellingen overeind is gebleven.

2.3. Verwante onderwerpen

Onze opdracht heeft relaties met andere onderwerpen.

2.3.1. Syntax-gestuurde editors. Ons systeem moet worden gestuurd door een grammatica. Dit is het terrein van de syntax-gestuurde editors (syntax-directed editors). Voor verschillende programmeertalen bestaan er editors, die kennis hebben van de syntax van die programmeertaal. Een voorbeeld hiervan is de editor voor de programmeertaal 'B' ([2, 8]). Om een while-statement in te voeren, tikt de gebruiker een 'w' in. Op het scherm verschijnt:

while □ : □

Dit is een *template.* De vierkante blokjes zijn *placeholders:* deze geven aan dat op die plaatsen nog invoer verwacht wordt. Wanneer de gebruiker aangeeft dat zij inderdaad een while-statement wil invoeren (en niet een write-statement dat ook met een 'w' begint), gaat de cursor op de eerste *placeholder* staan. De gebruiker tikt nu de conditie in.

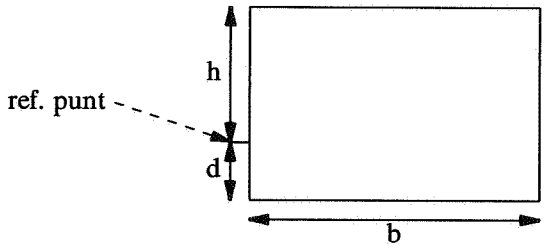
Analoog aan dit voorbeeld moet ons systeem zich gedragen. De gebruiker geeft het commando voor een integraal: op het scherm verschijnt het integraal-teken, en de cursor staat op de plaats waar de ondergrens moet komen.

Een ander voorbeeld van een syntax-gestuurde editor, is ALOA van Medina-Mora ([7]). Dit is een systeem dat, gegeven een grammatica, een bijbehorende syntax-gestuurde editor genereert.

Dit idee wordt in ons systeem gebruikt. Ons systeem werkt met een externe grammatica. Afhankelijk van deze externe grammatica wordt een syntax-gestuurde formule-editor gegenereerd. De grammatica kan zó gemaakt zijn, dat het gegenereerde systeem goed te gebruiken is door een niet-wiskundige. In hoofdstuk 3.2 gaan we hier verder op in.

2.3.2. Attribuutgrammatica's. Het systeem wordt gestuurd door een grammatica die syntax en semantiek van formules beschrijft. Om deze grammatica op te schrijven hebben we een notatie nodig . Een bekende notatie voor grammatica's is de zgn. BNF-vorm. Een andere, hiervan afgeleide, notatie is de attribuutgrammatica ([4, 15]). Met deze notatie kan syntactische en semantische informatie gemanipuleerd worden. Zie hiervoor hoofdstuk 3.4.

2.3.3. Lay-out met behulp van 'boxes'. Voor het bepalen van de lay-out van een formule, is het idee van een box ([5]) nuttig. Een box is gedefinieerd door zijn hoogte, breedte en diepte (zie fig. 2.3).



FIGUUR 2.3

De inhoud van een box is niet van belang: het kan een enkel karakter zijn, maar ook een complete formule. De hoogte en diepte van een box bepalen het referentiepunt van een box. De positie van het referentiepunt bepaalt de positie van de box (en daarmee de positie van zijn inhoud). Zie verder hoofdstuk 3.4.

3. PROBLEEMAANPAK

In dit hoofdstuk behandelen wij de structuur van formules, het ontwerp van de grammatica, de manier van invoer en de manier van uitvoer. De uitvoerbeschrijving bestaat uit een aantal regels, de zogenoemde display-regels. De grammatica en de manier van in- en uitvoer moeten voldoen aan de door ons gestelde doelstellingen (hoofdstuk 2).

3.1. Structuur van wiskundige formules

Een wiskundige formule is op een bepaalde manier opgebouwd. Deze opbouw wordt de structuur van een formule genoemd. Een breuk zal altijd uit een teller en een noemer bestaan.

Een simpele structuur is om een formule te beschouwen als een rij karakters. Het is ook mogelijk om een formule opgebouwd te denken uit een aantal brokstukken. Zo'n brokstuk kan op zijn beurt soms verder onderverdeeld worden. Bv. de formule '$a + b \times c^2$' bestaat in de simpele structuur uit 'a', '$+$', 'b', '$\times$', 'c' en '2'. In de tweede structuur is een mogelijke onderverdeling aan te brengen als '$a + b$', '$\times$' en 'c^2'.

De bedoeling is dat de verdeling maar op één manier kan gebeuren. Is het mogelijk in één formule meerdere toegestane onderverdelingen aan te brengen, dan is de structuur niet éénduidig maar ambigu.

De structuur van formules bepaalt het cursorgedrag. De cursor wijst een (deel)formule op het scherm aan. Heeft de aangewezen formule deelformules, dan kan de gebruiker op zo'n deelformule 'inzoomen', d.w.z. met de cursor zo'n deelformule aanwijzen. De omgekeerde handeling 'uitzoomen', dwz. van een deelformule naar de omvattende formule gaan, is ook mogelijk. Bestaat een formule uit meerdere deelformules, dan is het mogelijk naar de rechter of linker deelformule te gaan. De structuur van een formule geeft de volgorde van de deelformules aan. Dit heeft tot gevolg dat als de formule-structuur ambigu is, het cursorgedrag niet meer éénduidig is (zie voorbeeld paragraaf 3.1.1).

We bepalen een opdeling door prioriteiten (3.1.1), associativiteiten (3.1.2) en onderdelen binnen een wiskundige constructie (3.1.3) aan te geven.

3.1.1. De prioriteit. De prioriteit verdeelt een formule in brokstukken. De prioriteit is groter of gelijk aan nul. De operator die prioriteit nul heeft, heeft de zwakste binding. Deze binding bepaalt hoe de formule in delen, genaamd deelformules, wordt opgedeeld.

Bv. stel de prioriteit van '+' = 1, '×' = 2 en machtsverheffen = 5. Het is mogelijk om de binding van verschillende onderdelen, door het zetten van haakjes, aan te geven. Alle deelformules van één niveau staan tussen één paar haken. De formule '$a + b \times c^2$' wordt op de volgende manier opgedeeld: '$(a +(b \times (c^2)))$'. De delen 'a', '+' en '$b \times (c^2)$' staan dus op hetzelfde niveau.

Als de prioriteit anders gekozen wordt, verandert de opdeling. Neem bijvoorbeeld prioriteit '+' = 5, '×' = 2, machtsverheffen = 1, dan geldt: '$a + b \times c^2$' = '$(((a +b) \times c)^2)$'.

De regel voor prioriteitswaarden die wij met bepaalde toevoegingen hanteren is gebaseerd op de bekende regel: Meneer Van Dalen Wacht Op Antwoord. Dit geeft de volgende prioriteiten:

- gelijkheid: 0
- optellen/aftrekken: 1
- vermenigvuldigen/delen: 2
- puntoperator: 3
- concatenatie: 4
- machtsverheffen: 5
- indiceren: 6

Met deze keuze van de prioriteiten wordt de formule op een wiskundige manier opgedeeld. Door andere prioriteiten te kiezen, kan een minder wiskundige opdeling verkregen worden. Als alle prioriteiten bv. gelijke waarden krijgen, krijg je de 'simpele' opdeling van een formule als een rij karakters.

De punt- en concatenatie-operatoren zijn eigenlijk alternatieven voor de vermenigvuldiging, '$2a$' = '$2.a$' = '$2 \times a$'. Omdat de opdeling van '$ab.c \times d$' in '$((ab).c) \times d)$' natuurlijker overkomt dan de opdeling '$(ab.c \times d)$' hebben we voor verschillende prioriteiten gekozen.

Machtsverheffen en indiceren verschillen eveneens van prioriteit. Dit is gedaan om een formule op één manier op te delen, onafhankelijk van het feit in welke volgorde deze formule in het programma ingevoerd wordt.

Want stel dat de prioriteit van machtsverheffen en indiceren gelijk is. De grammatica is nu ambigu. Voor de invoer 'x', '↑' (machtsverheffen), 'y', '↓' (indiceren) en 'i' wordt de formule opgedeeld als '$(x^y)_i$'. Voor de invoer van 'x', '↓', 'i', '↑', en 'y' is de formule opgedeeld als '$(x_i)^y$'.

Het cursorgedrag is voor deze gevallen verschillend, één keer 'inzoomen' levert in het eerste geval (de cursor stond op de gehele formule), de formule 'x^y', en in het tweede geval 'x_i' op. Om dit te voorkomen hebben we de prioriteit verschillend gemaakt.

Het is namelijk de bedoeling de wiskundige structuur van een formule te gebruiken voor het maken van een externe grammatica. Deze grammatica moet het programma besturen.

Een uitzondering op de toekenning van wiskundige prioriteiten is, dat wiskundige constructies als wortel of integraal bij ons ook een prioriteit hebben gekregen. Dit is nodig om de constructie-delen bij elkaar te binden. Bv. de opdeling van de formule '$\int x\mathrm{d}x + 1$' is '$((\int x\mathrm{d}x) + 1)$' en <u>niet</u> '$(\int ((x\mathrm{d}x) + 1))$'.

3.1.2. De associativiteit. De associativiteit bepaalt mede de opdeling van een formule in onderdelen. De prioriteit werkt met operatoren van verschillend prioriteitsniveau, de associativiteit werkt met operatoren van gelijke prioriteit. De formule '$a \times b \times c$' wordt <u>niet</u> opgedeeld door op de prioriteit te letten, met alleen prioriteit zou deze formule als '$((a \times b) \times c)$' of '$(a \times (b \times c))$' gezien kunnen worden. Door nu de associativiteit te gebruiken staat de opdeling vast. Iedere infix-operator heeft een associativiteit. De waarden die de associativiteit kan aannemen, zijn:

- RIGHT: $a \times b \times c = a \times (b \times c)$
- LEFT: $a \times b \times c = (a \times b) \times c$
- NONE: Duidt aan dat een karakter of construct geen associativiteit heeft.

Een toevoeging aan de in de wiskunde gebruikte associativiteiten is, dat we de waarde LEVEL

ingevoerd hebben. Deze waarde geeft aan dat de associativiteit 'neutraal' is. In een vermenigvuldiging hoeft géén onderverdeling aangebracht te worden. Er is geen verschil tussen '$a \times (b \times c)$' en '$(a \times b) \times c$'.

Het cursorgedrag van een formule waarin de vermenigvuldiging associativiteit RIGHT of LEFT heeft, is niet makkelijk om mee te werken. Neem als voorbeeld de formule '$a \times b \times c$' met de associativiteit van vermenigvuldiging RIGHT, dan wordt de formule opgedeeld als '$(a \times (b \times c))$'. De cursor staat op de gehele formule. Eén keer 'inzoomen' levert 'a' op. Wordt er met de cursor naar rechts gegaan, dan wordt achtereenvolgens '$\times$' en '$b \times c$' aangewezen. Wil de gebruiker 'b' bereiken dan zal de gebruiker moeten 'inzoomen' terwijl de gebruiker zou verwachten met de cursor gewoon naar rechts te moeten 'lopen'.

Is de associativiteit LEVEL, dan wordt de formule opgedeeld als '$(a \times b \times c)$' en bevinden alle deelformules zich op één niveau. Deze deelformules 'a', '$\times$', 'b', '$\times$' en 'c' zijn met één keer 'inzoomen' en naar rechts lopen te bereiken. Dit laatste komt veel natuurlijker over.

Machtsverheffen en indiceren hebben wel een associativiteit, want met '$x_{(i)}$' wordt iets anders bedoelt dan met '$(x_i)_j$'.

Dit levert voor de operatoren de volgende associativiteiten op:
- gelijkheid: LEVEL
- optellen/aftrekken: LEVEL
- vermenigvuldigen/delen: LEVEL
- puntoperator: LEVEL
- concatenatie: LEVEL
- machtsverheffen: RIGHT
- indiceren: RIGHT

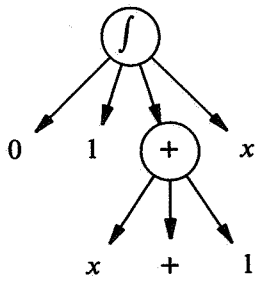
3.1.3. De onderdelen. De formule-opdeling wordt niet alleen door de twee voorgaande eigenschappen bepaald. In verschillende wiskundige constructies zoals b.v. een integraal, is met deze twee eigenschappen geen verdere onderverdeling aan te brengen. De constructies bestaan meestal wel uit deelformules. Voor een integraal zijn deze deelformules (formule 3.1): de ondergrens (1), de bovengrens (2), de integrand (3) en de integraal-variabele (4).

$$\int_{a(1)}^{b(2)} x\,(3)\,\mathrm{d}x\,(4) \qquad (3.1)$$

Om nu een structuur in de formule aan te brengen moeten de onderdelen aangegeven worden. Dit geldt alleen voor prefix-operatoren. De infix-operatoren bepalen door prioriteit en associativiteit de structuur.

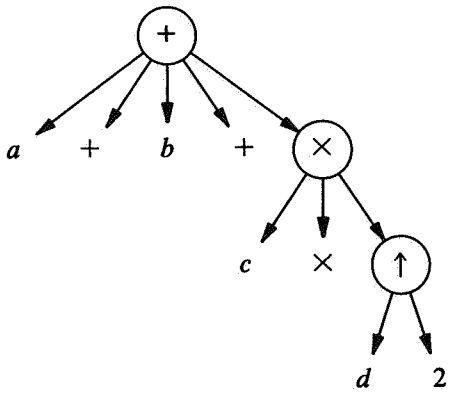
3.1.4. De boomrepresentatie. Het voorgaande verhaal maakt het mogelijk een formule intern in een boomstructuur te representeren. De boomstructuur wordt zichtbaar in een formule, door te bedenken dat een formule uit deelformules is opgebouwd. Deze deelformules vormen de deelbomen (kinderen) van de boom die de gehele formule representeert. De prioriteit en associativiteit bepalen de positie van iedere operator in de boom.

Een integraal heeft 4 onderdelen. Deze worden als kind-knopen aan de integraal-knoop gehangen. De formule '$\int_0^1 x - 1\,\mathrm{d}x$' (3.2) heeft de boomrepresentatie voorgesteld door fig. 3.1.



FIGUUR 3.1

De formule wordt in een boomstructuur omgezet door eerst naar de prioriteit te kijken. Een operator met een hogere prioriteit komt lager in de boom te hangen, d.w.z. verder van de wortel vandaan. De associativiteit verdeelt de verkregen delen verder. De formule '$a + b + c \times d^{2}$' geeft de boomrepresentatie als in fig. 3.2.



FIGUUR 3.2

Het is mogelijk om uitgaande van een formule de boomrepresentatie te tekenen, anderzijds kan uitgaande van een boom de formule geconstrueerd worden.

3.2. Structuur van de grammatica

Na het aanbrengen van een structuur in formules en de daarbij horende boomrepresentatie, is het mogelijk een grammatica te maken die deze boomstructuur weergeeft.

De grammatica moet aan de eerdergenoemde doelstellingen voldoen. De grammatica moet makkelijk uitbreidbaar zijn, dwz. de gebruiker moet makkelijk nieuwe wiskundige symbolen en constructies kunnen toevoegen. De relatie tussen de opgebouwde boom (intern) en de formule op het scherm moet éénduidig zijn. Eén cursorbeweging in de formule moet overeen komen met één stap in de boom. Dit voorkomt dat voor de ene cursorbeweging 3 boomstappen, en voor een andere cursorbeweging 6 boomstappen nodig zijn.

Het vinden van het juiste grammatica-mechanisme heeft ons veel tijd gekost. We begonnen met een BNF-vorm. Nadat we in [7] een beter grammatica-mechanisme hadden gevonden, moesten we deze nog aanpassen voor formules. Dit aanpassen was eveneens tijdrovend. Na iedere verandering hebben we gekeken of de formule en de boom overeen kwamen met ons idee. Eveneens hebben we erop gelet dat het doorlopen van een formule met een cursor "logisch" is. Met "logisch" bedoelen we dat er een wiskundige structuur zichtbaar wordt. Een voorbeeld hiervan staat beschreven in de paragraaf over de associativiteit (3.1.2). Het "logisch" doorlopen van de formule wordt daar met toevoegen van de associativiteit LEVEL bereikt. Het uiteindelijke resultaat wordt besproken in 3.2.3.

3.2.1. De BNF-grammatica. We zijn begonnen de structuur van formules uit te drukken in een BNF-grammatica. De BNF-grammatica wordt aan de hand van een voorbeeld beschreven. Dit voorbeeld staat op deze en de volgende bladzijde.

In deze grammatica moet aangegeven worden wat de prioriteit en associativiteit van een operator is. Dit gebeurt in dit grammatica-mechanisme door per prioriteitsniveau één productieregel aan de grammatica toe te voegen. De '+' en de '×' operator komen dan ook niet in één productieregel voor. Hebben verschillende operatoren dezelfde prioriteit, dan worden per productieregel de verschillende mogelijkheden aangegeven ('+' en '-', zie regel 1 van het voorbeeld). De grammatica verkrijgt de associativiteit door in de rechterkant van een productieregel links of rechts recursieve producties te gebruiken (regel 2).

Een formule wordt opgebouwd door uitgaande van een startsymbool, in het voorbeeld 'formule', deze te vervangen door één keuze uit de bijbehorende rechterkant. Dit proces herhaalt zich totdat er geen non-terminalen meer in de formule voorkomen. Non-terminalen zijn die symbolen, die in de grammatica aan de linkerkant van een productieregel voorkomen, b.v. 'term'. Alle andere symbolen worden terminalen genoemd (bv. 'a'), deze symbolen zijn dikgedrukt.

(regel 0) formule := expr | ϵ
(regel 1) expr := expr + term | expr - term | term
(regel 2) term := term × factor | factor
(regel 3) factor := expr ↑ expr | construct
(regel 4) construct := karakter | haakje | int
(regel 5) karakter := **a** | **b** | **c** | **d**
(regel 6) haakje := (expr)
(regel 7) int := $\int$ expr expr expr **d** expr

De formule '$a + b \times c - d$' wordt dan als volgt gegenereerd:

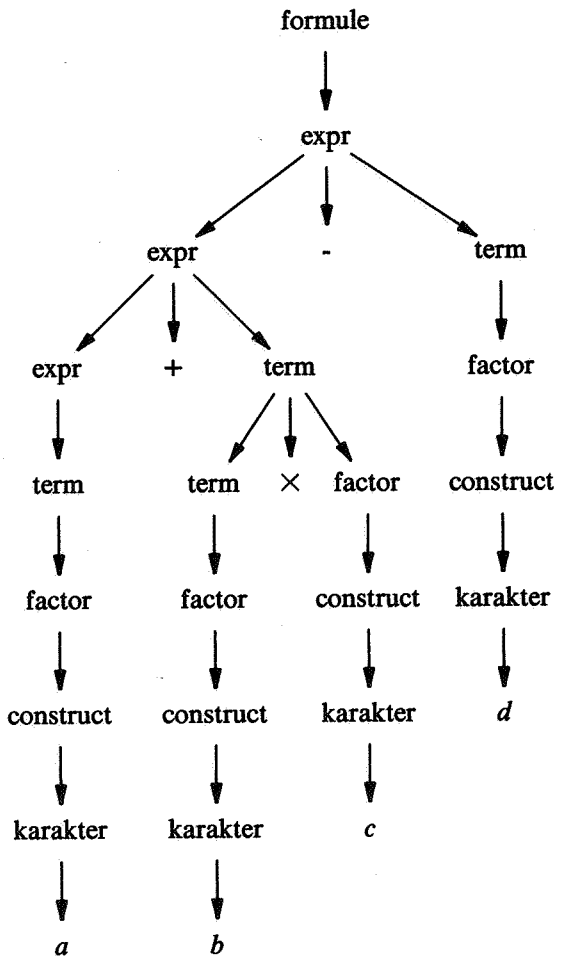
formule
expr
expr - term
(expr + term) - term
(term + term) - term
(factor + term) - factor
(factor + (term '×' factor)) - factor
(factor + (factor '×' factor)) - factor
(construct + (construct '×' construct)) - construct
(karakter + (karakter '×' karakter)) - karakter
(a + (b × c)) - d)

In deze grammatica heeft de optelling een lagere prioriteit dan de vermenigvuldiging. Beide operatoren hebben een linkse associativiteit.

Het invoeren van een nieuwe constructie gebeurt door een productieregel voor deze constructie in de grammatica toe te voegen, en het mogelijk te maken deze regel te selecteren. In de productieregel wordt gespecificeerd uit hoeveel delen de constructie bestaat.

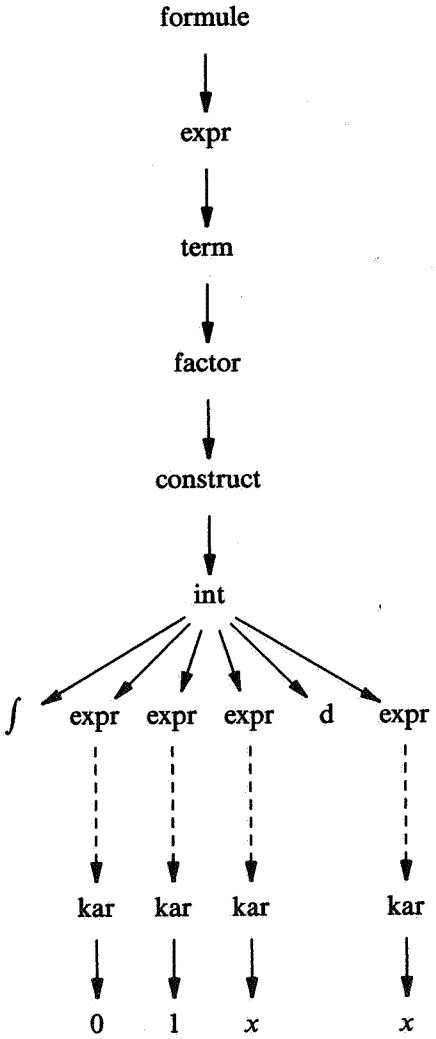
Bijvoorbeeld voor een integraal:
int := $\int$ expr expr expr **d** expr
In fig. 3.3 en 3.4 staan 2 voorbeelden van een formule met de bijbehorende boom, volgens deze grammatica opgebouwd.

$a + b \times c - d$



FIGUUR 3.3

$\int_0^1 x\,dx$



FIGUUR 3.4

Het voordeel van deze grammatica is, dat we zijn bekend met de BNF-grammatica. Nadelen zijn:

1) Machtsverheffen is ambigu, b.v. 'a^{b^c}' kan gezien worden als '$\left[a^b\right]^c$' of '$a^{\left[b^c\right]}$'. Dit is met enige moeite wel op te lossen.
2) Er zijn veel loze knopen in de boom. Loze knopen zijn interne knopen met maar één kind dat eveneens een interne knoop is, b.v. de knoop met label 'factor' (fig. 3.3). Dit nadeel is met een BNF-grammatica niet op te lossen. De loze knopen veroorzaken dat één cursorbeweging in de formule niet met één cursorbeweging in de boom overeenkomt, bv. de cursor gaat van '*a*' naar '+' (fig. 3.3), dit is één cursorbeweging. De corresponderende boombeweging heeft 7 stappen nodig. Staat de cursor op de integraal (fig. 3.4) en 'zoomt' de gebruiker in op de ondergrens (1 beweging), dan komt dat in de boom met 6 stappen overeen.
3) Er zijn "niet bereikbare" knopen in de boom. "Niet bereikbare" knopen zijn knopen die wel

informatie bevatten maar niet door de formulecursor aangewezen mogen worden. Het integraalteken hangt wel in de boom maar mag door een gebruiker niet geselecteerd worden. Het is de gebruiker namelijk door ons niet toegestaan om een integraal te veranderen in een sommatie door het $\int$ -teken in een $\sum$ -teken om te zetten.

3.2.2. De grammatica van Medina-Mora. In [7] staat een geheel ander grammatica-mechanisme. In dit mechanisme wordt onderscheid gemaakt tussen TERMINALEN, NTERMINALEN en CLAUSES.

TERMINALEN zijn die knopen die als bladeren van de boom voorkomen. De bladeren representeren die formules die geen deelformules hebben, bv. '*a*'.

NTERMINALEN zijn de interne knopen in een boomstructuur. De knopen representeren formules die wel deelformules hebben, bv. integraal. In de bijbehorende productieregel staan de klassen van de betreffende deelformules.

Met behulp van CLAUSES wordt de opeenvolging van TERMINALEN en NTERMINALEN vastgelegd. Voor iedere gebruikte klasse staat in de grammatica een lijst met klassen (zie voorbeeld). Deze lijst geeft aan dat na een element uit de klasse aan de linkerkant van een CLAUSE-regel, een element uit een klasse van de bijbehorende rechterkant moet volgen.

De toegestane volgorde van karakters in een formule wordt in een BNF-vorm door middel van productieregels vastgelegd. De formule '$a++b$' is volgens de productieregels van het voorbeeld in 3.2.1 dan ook niet te maken. Met het CLAUSE-mechanisme wordt de volgorde van karakters in een formule door de klassen van deze karakters bepaald. Volgens de grammatica van het onderstaande voorbeeld mag na een karakter met klasse OPERATOR (b.v. '+') alleen een karakter met klasse CHAR of CONSTRUCT komen. Een karakter met klasse OPERATOR is niet toegestaan. De formule '$a++b$' is daarom in deze grammatica eveneens niet toegestaan.

Een TERMINAL of NTERMINAL heeft eigenschappen. Deze keuze van eigenschappen is niet een vast onderdeel van het grammatica-mechanisme. Eén eigenschap is wél verplicht, en dat is de eigenschap 'klasse'; m.b.v. deze eigenschap werkt het CLAUSE-gedeelte.

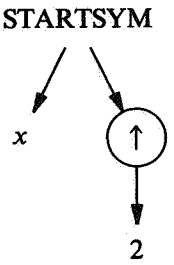
Met dit mechanisme hebben we een grammatica opgesteld. Pas na veel werk hadden we een grammatica waar we tevreden over waren. Eén van onze eerste pogingen zag er uit als de grammatica in onderstaand voorbeeld.

```
STARTSYM    : EXPR

TERMINALEN
                                                    aantal
            klasse                  prioriteit assoc    deelformules
a           : CHAR ,                    10    , NONE ,      0
b           : CHAR ,                    10    , NONE ,      0
c           : CHAR ,                    10    , NONE ,      0
d           : CHAR ,                    10    , NONE ,      0
0           : CHAR ,                    10    , NONE ,      0
1           : CHAR ,                    10    , NONE ,      0
+           : OPERATOR ,                 1    , NONE ,      0
-           : OPERATOR ,                 1    , NONE ,      0
×           : OPERATOR ,                 2    , NONE ,      0
(           : HO ,                      10    , NONE ,      0
)           : HS ,                      10    , NONE ,      0

NTERMINALEN
SUP         : OPERATOR ,                 5    , RIGHT ,     1     , EXPR
INT         : CONSTRUCT ,               10    , NONE ,      4     , EXPR EXPR EXPR EXPR
```

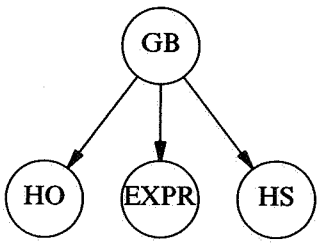
```
(          : CONSTRUCT ,    10   , NONE ,    3    , ( EXPR HS
GB         : CONSTRUCT ,    10   , NONE ,    3    , HO EXPR HS

CLAUSES
CONSTRUCT  : OPERATOR HS
CHAR       : OPERATOR HS
OPERATOR   : CHAR CONSTRUCT
EXPR       : CHAR CONSTRUCT
```

Als eigenschappen hebben we, naast de klasse, ook de prioriteit, de associativiteit, het aantal deelformules en de klassen van de deelformules genomen. Aan de hand van dit voorbeeld kunnen we laten zien hoe we tot onze huidige grammatica (zie Appendix I) zijn gekomen. De voor- en nadelen van de grammatica uit het voorbeeld zijn:

- voordelen:
  1) De prioriteit en de associativiteit wordt rechtstreeks in de grammatica gezet. De loze knopen van de BNF-grammatica worden hierdoor voorkomen.
  2) Machtsverheffen is niet ambigu meer.
  3) De "niet bereikbare" knopen worden geëlimineerd door de inhoud van deze knopen als extra informatie te beschouwen. Deze extra informatie komt in de display-regels tot uitdrukking (zie sectie 3.4).
- nadelen:
  1) De gewenste boomstructuur is niet uit deze grammatica op te bouwen. B.v. de formule 'x^2' levert de boom:



  2) Indeling van de TERMINALEN en NTERMINALEN is niet natuurlijk. De '+' operator staat bij de TERMINALEN terwijl machtsverheffen bij de NTERMINALEN staat, beide zijn echter operatoren. Deze opdeling is nodig omdat beide operatoren een verschillend gedrag hebben, wat betreft het vinden van de bijbehorende operanden.
     De '+'-operator vindt de beide operanden door prioriteit en associativiteit. De machtsverheffen-operator vindt de eerste operand op dezelfde manier als de '+'-operator. De tweede operand bestaat echter uit alle invoer die na de operator wordt ingetikt. De invoer voor de macht wordt beëindigd door een END-commando hiervoor te geven.
  3) Een TERMINAL heeft nooit een associativiteit, en het aantal deelformules is altijd nul, zodat dit voor een terminaal overbodige informatie is.
  4) De '+' en de integraal hebben dezelfde waarde voor de associativiteit terwijl in beide gevallen iets anders bedoeld wordt. In het geval van de '+' wordt bedoeld dat de associativiteit LEVEL is, en voor de integraal NONE (zie 3.1.2).
  5) Haakjes komen 2 keer in de grammatica voor, een keer voor groeihaak en een keer voor niet-groeihaak. Een groeihaak is een haak waarvan de afmetingen afhangen van de expressie tussen de haken. Het invoeren van een groeihaak gebeurt door eerst een commando te geven en vervolgens het gewenste type haakje in te tikken. Na het commando is de boomstructuur als in fig. 3.5 Het



FIGUUR 3.5

nu ingetikte haakje komt in de knoop HO (Haakje Openen) te staan. Een niet-groeihaak is ook toegestaan. Het werkt niet makkelijk als de gebruiker hiervoor ook een commando moet geven. Voor een niet-groeihaak tikt de gebruiker alleen het gewenste haakje in. Na het intikken van het haakje is er geen HO-knoop aanwezig, dus moet de structuur zoals in fig. 3.5 aangemaakt worden. I.p.v. GB (Grow Bracket) komt NB (Non-grow Bracket) in de bovenste knoop te staan. De voorafgaande invoer is dus van belang voor het bepalen van de juiste actie als er een haakje ingetikt wordt.

6) De display-regels zijn niet makkelijk toe te voegen. Bij elkaar horende onderdelen, zoals de grondexpressie en de macht van machtsverheffen, staan niet bij elkaar. De samenhang komt uit de grammatica niet duidelijk naar voren.
7) Nieuwe symbolen en constructies toevoegen is niet makkelijk, dit is in strijd met de doelstelling. Dit komt omdat de opdeling niet 'logisch' is, zie b.v. punt 2 hierboven.

3.2.3. De huidige grammatica. Van de huidige grammatica staat een klein gedeelte in Appendix I. De grammatica benadert de wiskundige structuur meer dan de voorgaande grammatica.

Een TERMINAL heeft geen overbodige informatie meer. Er wordt dus een duidelijk onderscheid tussen TERMINALEN en NTERMINALEN aangebracht.

De '+'-operator is nu duidelijk te herkennen als een infix-operator met als beide operanden een expressie. De '+' komt nu dan ook bij de NTERMINALEN voor. Haakjes staan 1 keer in de grammatica, nl. voor Non-Grow Bracket. Voor een groeihaak moet nu een commando gegeven worden waarin het soort haakje al verwerkt is. Zo is er een commando voor een rond groeihaakje (GBR) en een recht groeihaakje (GBS). Beide soorten haken (groei en niet-groei) zijn NTERMINALEN en staan ook bij de NTERMINALEN, terwijl dat eerst niet het geval was. De gebruiker hoeft nu niet meer een sluithaakje in te tikken. De sluithaak is dezelfde als de openingshaak en wordt door het systeem erbij gegenereerd. De voordelen van de vorige poging (zie sectie 3.2.2) gelden hier echter nog steeds.

Het opnemen van de klasse START is nodig om gedrag verschillend van EXPR te veroorzaken. Een EXPR geeft aan dat er een expressie verwacht wordt. In de boom wordt een expressie-knoop gehangen, en het eerstvolgende karakter wordt hier ingevuld, als dat toegestaan is. Dat invullen is toegestaan als bij de CLAUSES van EXPR, de klasse van het ingetikte karakter staat. Is het invullen niet mogelijk, het karakter heeft een andere klasse, dan wordt het karakter volgens zijn prioriteit en associativiteit in de boom gehangen.

Een START-knoop geeft aan dat de nu komende deelboom (dus deelformule) een op zichzelf staand geheel is. Bij een macht hebben we gekozen voor de mogelijkheid dat alles wat na het machtsverheffen-commando wordt ingetikt, in de macht verschijnt. Dit voorkomt onnodig gebruik van haakjes. Het einde van de invoer van een op zichzelf staand geheel wordt door END aangegeven. Voorbeeld: de invoer van '2', '↑', 'x', '+' en '1' levert de formule '2^{x+1}', en niet de formule '2^x+1' op. In het eerste geval heeft machtsverheffen in de grammatica een START als tweede deelformule, in het tweede geval is dat een EXPR.

De voordelen van deze grammatica zijn:

1) De lay-out informatie kan later eenvoudig toegevoegd worden.
2) De structuur van de grammatica is gemakkelijk te begrijpen.
3) Er wordt geen overbodige informatie gegeven.
4) Het toevoegen van nieuwe symbolen en constructies is eenvoudig.

Een standaardoperator van de grammatica is EMPTY: dit is de concatenatieoperator. Deze is standaard gemaakt, omdat deze operator altijd wel gebruikt zal worden, net als de letters en de cijfers. Het systeem gaat er vanuit dat deze operator en de letters en de cijfers in de grammatica staan. Dit is nodig om deze operator en de letters en cijfers lay-out regels te kunnen geven. Dit maakt de grammatica wel erg omvangrijk.

3.3. Structuur van de invoer

De gebruiker kan met behulp van de muis commando's (bv. integraal), speciale karakters (bv. α) en standaardfuncties (bv, sin) invoeren. Het toetsenbord wordt gebruikt om ascii-karakters in te voeren. Het invoeren van een formule zou ook met functie-toetsen kunnen gebeuren, maar wij hebben voor de aanpak m.b.v. de muis gekozen. De gebruiker hoeft dus de toets voor ieder commando niet uit zijn hoofd te leren. Alle mogelijkheden zijn tergelijkertijd te zien.

3.4. Structuur van de display-regels

In de grammatica staat behalve informatie over de structuur van formules (syntax) ook informatie over de lay-out (semantiek) (zie Appendix II). Deze lay-out informatie wordt gebruikt voor de bepaling van de afmeting van een formule en de berekening van de positie van de formule.

De afmetingen van een formule zijn de hoogte, breedte en diepte van de omvattende box (zie [5]). De afmetingen van een TERMINAL zijn zoals deze in het font met de gewenste puntgrootte staan vermeld.

Het bepalen van de afmeting van de formule van een NTERMINAL gebeurt door de afmetingen van de deelformules en symbolen te berekenen en vervolgens uit deze gegevens de formule-afmeting af te leiden (zie 3.4.1).

Een formule is niet altijd te beschrijven in alleen deelformules. Sommige constructies hebben extra tekens nodig. De integraal bijvoorbeeld, het integraal-teken hoort niet bij de deelformules maar maakt wel deel uit van de integraal. Deze extra tekens worden door ons symbolen genoemd.

De posities van de deelformules en symbolen worden bepaald door de posities relatief ten opzichte van de formulebox te geven. Het referentiepunt (zie 2.2.3) van deze box heeft coördinaat (0, 0). Het programma rekent dit om naar absolute posities.

De formule wordt op het scherm getekend door de box die deze formule bevat, op het scherm op de juiste positie te plaatsen. Het referentiepunt van de box komt op de aangegeven positie. De afmetingen van de box definiëren op het scherm de grenzen waarbinnen de inhoud van de box ligt, mits de formule niet buiten de box steekt. Informatie over de te tekenen symbolen staan in de bijbehorende fonts.

3.4.1. De display-regels van Medina Mora. In [7] wordt een vorm van lay-out informatie gegeven voor statements in een programmeertaal. De grammatica-regel is bijvoorbeeld:

FOR: loopvar exp exp stepexp stmts

De display-regels zijn van de vorm:

"for" @1 "=" @2 "to" @3 "step" @4@+@n@5@-

De @<nr>, $1 \leqslant nr \leqslant 5$, verwijzen naar non-terminals in de rechterkant van de productieregel, @1 = loopvar, @2 = exp, ..., @5 = stmts. De andere tekens als @n, enz. geven lay-out informatie. B.v. @n = regelovergang, @+ = verhoog indentatieniveau met 4 spaties, @- = verlaag indentatieniveau, enz. In deze display-regels worden geen posities aangegeven. Alle karakters komen achter elkaar. Voor een formule is dit niet mogelijk, een formule is 2-dimensionaal, b.v. de ondergrens van een integraal moet lager staan dan de integrand. Naar aanleiding van dit voorbeeld hebben wij onze display-regels ontwikkeld.

3.4.2. De vormgeving van de display-regels. In de display-regels moeten constructies komen als 'teken de tweede deelformule naast de eerste deelformule', 'het derde symbool komt op positie (0, 0)', enz. Er moet dus een mogelijkheid zijn om de i-de deelformule en het j-de symbool aan te duiden. Een deelformule aanduiding begint met $. Het $-teken wordt gevold door een nummer, $1 = eerste deelformule, $2 = tweede deelformule, enz. Een deelformule heeft afmetingen die gebruikt worden bij het berekenen van de formule-afmetingen. Het moet dus mogelijk zijn deze afmetingen op te vragen. Zo geeft $1.h, de hoogte van de eerste deelformule. Deze 'h' is een attribuut. Andere attributen als 'd' (diepte) en 'w' (breedte) zijn ook toegestaan voor een deelformule.

Hetzelfde geldt voor een symbool. Om onderscheid tussen een symbool-aanduiding en een deelformule-aanduiding te maken, begint een symbool-aanduiding met @. @1.h is de hoogte van het eerste symbool. De attributen 'd' (diepte) en 'w' (breedte) zijn ook toegestaan. Een symbool heeft bovendien de attributen 'f' (font) en 'i' (index), deze zijn dus niet toegestaan voor een deelformule.

In de display-regels wordt van ieder symbool de volgende informatie verwacht:

- het font: In dit font is informatie over de afmetingen en de vormgeving van het symbool te vinden. De puntgrootte wordt door het programma berekend, de gebruiker geeft dit niet expliciet aan.
- de index: De index bepaalt waar in het font de informatie over het karakter te vinden is. De index hoeft niet uit één karakter te bestaan. Standaardfuncties als b.v. 'sin' bestaan uit meerdere karakters maar worden door ons als één geheel opgevat. Bestaat de index uit meerdere karakters dan worden deze één voor één afgehandeld, d.w.z. de afmetingen worden bepaald door de afmetingen van de karakters afzonderlijk uit te rekenen en vervolgens de afmetingen van het geheel te bepalen.
- één afmeting: Door middel van de gegeven afmeting is het mogelijk een bijpassende puntgrootte te vinden. In deze gevonden puntgrootte worden de andere afmetingen uitgerekend. Bv. de hoogte van het integraal-teken hangt af van de hoogte van de integrand.
- de positie waar het symbool getekend moet worden: Bij de positie staat ook een schalingsfactor voor de puntgrootte. De informatie geeft aan dat het zoveelste symbool op positie (x, y) getekend moet worden, in de huidige puntgrootte vermenigvuldigd met de schalingsfactor.

Voor een deelformule staat in de display-regels alleen informatie over de positie van deze deelformules, en de schalingsfactor voor de puntgrootte. De positie is evenals die voor een symbool relatief ten opzichte van het referentiepunt van de omvattende formulebox.

In de display-regels moet verder per productieregel informatie staan over de bepaling van de afmetingen van de bijbehorende formule. Deze formule wordt met '$0' aangeduid, dus '$0' is een aanduiding voor de linkerkant van een productieregel. De afmetingen van de formule hangen af van de deelformules '$<nr>', met nr $\geqslant$ 1, en de symbolen '@<nr>', nr $\geqslant$ 1.

De volgorde van de verschillende onderdelen in de display-regels is verplicht als volgt:

1) de positie van de deelformules en symbolen
2) een beschrijving van de symbolen met hun font en index
3) de afmetingen van de omvattende formule

Alleen voor TERMINALEN zijn display-regels niet verplicht. Zijn deze regels niet aanwezig, dan worden ze door het programma gegenereerd. De standaardwaarde voor de index en het font worden ingevuld. De index = ascii-waarde van de TERMINAL en het font = *I(talic)*. De afmetingen van de TERMINAL worden uitgerekend.

Iedere display-regel wordt afgesloten met een puntkomma. Een blok display-regels behorend bij één TERMINAL of NTERMINAL wordt beëindigd door een spoorwegkruisings-teken ('#').

3.4.3. De standaardfuncties en de constanten van de display-regels. In de display-regels kunnen constanten en functies gebruikt worden. Deze vergemakkelijken het gebruik van de display-regels.

Voor het attribuut 'index' van een symbool, kan als waarde een getal of een karakter gegeven worden. Zo wordt door de index '_' het symbool 'underscore' verkregen van een bepaalde puntgrootte. Als de gebruiker nu een horizontale streep van een bepaalde lengte wil, kan de constante HORBAR als index gebruikt worden. VERTBAR geeft een verticale streep van een bepaalde lengte. B.v. :

```
@1.i = HORBAR;
```

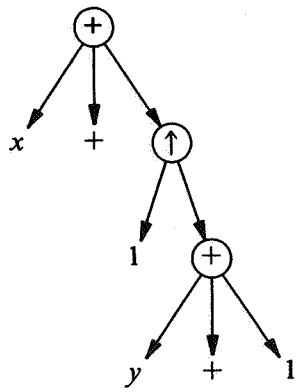
@1.w = $1.w;

Door het gebruik van standaardfuncties neemt het programma een gedeelte van het rekenwerk voor de positiebepaling over. B.v. voor centreren. Moet '$2' (tweede deelformule) boven '@1' (eerste symbool) gecentreerd worden, dan moet '$2' op positie ((lengte @1 - lengte $2) / 2, hoogte @1 + diepte $2) beginnen. Voor de gebruiker is het omslachtig om dit soort standaard-rekenoperaties steeds opnieuw in de display-regels te zetten. Hiervoor is dan ook een standaardfunctie 'center_above()' ingevoerd. De standaardfunctie 'print()' tekent de eerste parameter op de positie (tweede parameter, derde parameter) in de huidige puntgrootte maal de vierde parameter.

We geven nu een voorbeeld om de werking van de display-regels uit te leggen.

```
(1) SUP : COP , 5 , RIGHT , EXPR START ,
(2)   print($1, 0, 0, 1.0);
(3)   print($2, $1.w, $1.h, 0.66);
(4)   $0.w = $1.w + $2.w;
(5)   $0.d = $1.d;
(6)   $0.h = $1.h + $2.h;
   #
```

De eerste regel geeft aan dat machtsverheffen (SUP) klasse COP heeft. De prioriteit is vijf en de associativiteit is RIGHT, dwz. '$x + c^{2'} = '(x + (c\,\lceil 2' \rceil))'$. De eerste operand heeft klasse EXPR en de tweede operand heeft klasse START; bij invoer 'x', '+', '1', '↑', 'y', '+' en '1' is de formule-opdeling '$(x + (1^{y + 1}))$' en de boomopbouw als in fig. 3.6.



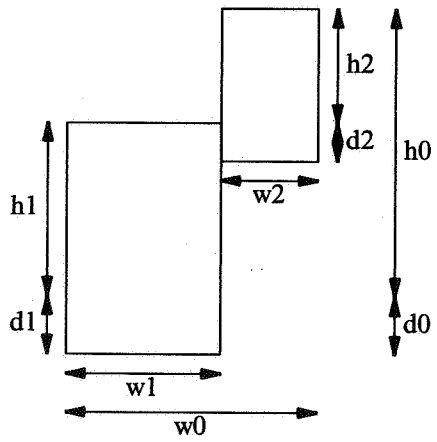
FIGUUR 3.6

De overige regels geven de lay-out informatie. Regel 2: Het referentiepunt van de grondexpressie ($1) moet op positie (0, 0) komen. De grondexpressie moet in de huidige puntgrootte * 1.0 getekend worden. Regel 3: De macht wordt in een kleinere puntgrootte, nl 2/3 maal de huidige puntgrootte, getekend. De positie van het referentiepunt van de macht is op de hoogte van de grondexpressie en naast deze expressie (fig. 3.7).

Regel 4: De breedte van de formule is gelijk aan de breedte van de grondexpressie plus de breedte van de macht. Regel 5: De diepte hangt alleen af van de grondexpressie. Regel 6: De hoogte wordt uit de 2 deelformules berekend.

Alle standaardfuncties staan in Appendix III beschreven.

De display-regels zijn eigelijk afgeleid van een attribuutgrammatica. De positie van de formule wordt aan de deelformules en de symbolen 'naar beneden' doorgegeven. De deelformules 'erven' de informatie. Voor het berekenen van de afmetingen van de omvattende formulebox is kennis van de afmetingen van de deelformules en symbolen nodig. De formule gebruikt informatie uit de rechterkant, en geeft de uitgerekende afmetingen 'naar boven' door.



FIGUUR 3.7

B.v. SUP[↓[psize, x, y], ↑[(h1+h2), d1, (w1+w2)]] :=
EXPR[↓[psize, x, y], ↑[h1, d1, w1]],
START[↓[psize×0.66, x+w1, y+h1], ↑[h2, d2, w2]];

↓: in de linkerkant betekent dit, krijgt de informatie 'van boven' af binnen ('inherited'). In de rechterkant betekent dit, geef de informatie 'naar beneden' door.

↑: in de linkerkant betekent dit , geef de informatie 'naar boven' door ('synthesized'). In de rechterkant betekent dit, krijgt de informatie 'van beneden' binnen dus geef naar de linkerkant door.

SUP krijgt de puntgrootte en de positie binnen en geeft de afmetingen van de formule terug. De afmetingen zijn berekend uit gegevens die door de deelformules teruggegeven worden.

De gebruikte notatie is een uitbreiding van de standaard attribuutgrammatica's. In de standaard attribuutgrammatica's staan expressies van attributen tussen de productieregels in. Het voordeel van de uitbreiding is, dat de regels compacter en overzichtelijker worden. Zie [15] voor meer informatie. We hebben de display-regels niet als een attribuutgrammatica genoteerd, maar op de manier die in Appendix III beschreven wordt. De reden hiervoor is, dat we denken dat onze manier makkelijker te begrijpen en te gebruiken is voor de gebruiker.

4. ONTWERP

Dit hoofdstuk beschrijft het ontwerp van het programma. De uitgangspunten van het ontwerp zijn, dat het programma zo onafhankelijk mogelijk is van:

- de implementatie van de gebruikte datastructuren
- het gebruikte beeldscherm
- de manier van invoer (gebruikers-interface)
- de grammatica

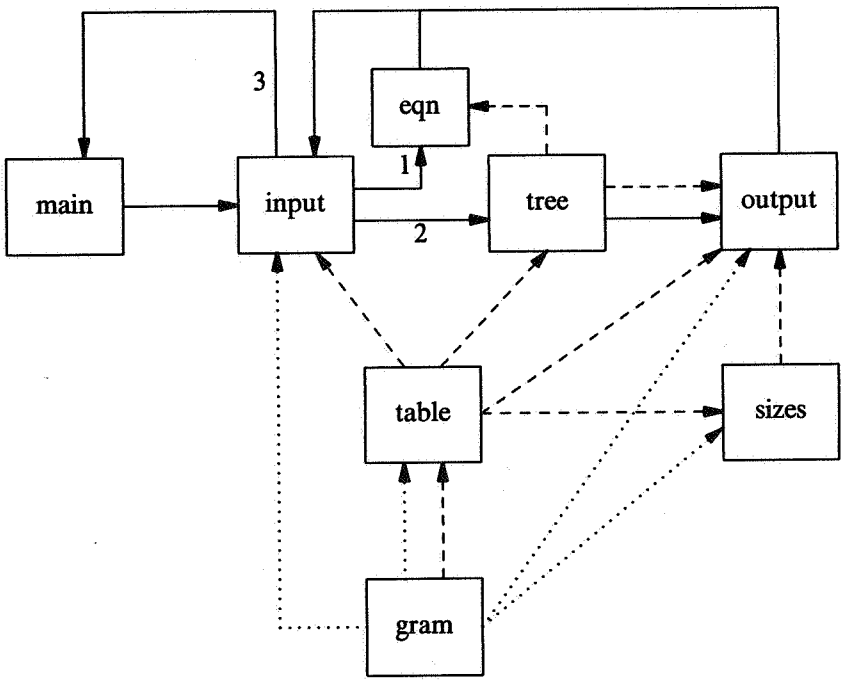
Daartoe is het programma verdeeld in modules. Iedere sectie in dit hoofdstuk beschrijft één module. Per module beschrijven we de functie, de (abstracte) datastructuur en (indien van toepassing) enkele routines. Meer gedetailleerde gegevens zijn te vinden in de documentatie van het programma. Er zijn 7 modules: 'main', 'table', 'tree', 'sizes', 'input', 'output' en 'eqn'. Ze werken globaal op de volgende manier samen.

Het programma begint in 'main': hier worden de initialisatie-routines van enkele modules aangeroepen. De belangrijkste routine van deze is de initialisatie-routine van de module 'table'. Deze vult een tabel met informatie uit de grammatica. De rest van het programma kan deze informatie gebruiken. Vervolgens start 'main' een loop in module 'input' op. De loop begint met het wachten op invoer. Zodra de gebruiker iets invoert, wordt deze invoer in de module 'input' omgezet in een interne code.

De invoer van de gebruiker hoeft niet altijd een gedeelte van een formule te zijn. Als de gebruiker

aan het 'editen' is, wordt een cursor bestuurd volgens de invoer van de gebruiker. De cursor wordt dan getekend door module 'output'. Ook kan de gebruiker het commando geven om eqn-code te genereren. Dan wordt de hoofd-routine in module 'eqn' aangeroepen. In alle andere gevallen is de gebruiker een formule aan het invoeren. Deze invoer wordt verwerkt door de hoofdroutine van module 'tree' aan te roepen.

Als de gebruiker een formule aan het invoeren is, moet hiervan een interne boomrepresentatie gemaakt worden. De hoofdroutine van de module 'tree' verwerkt de interne code en bouwt gaandeweg de boom op. De module 'tree' heeft de interne code nu verwerkt in de boom, maar nu moet de representatie van de formule op het scherm nog aangepast worden. Daarvoor wordt een routine in module 'output' aangeroepen. Deze routine bepaalt eerst m.b.v. de module 'sizes' de afmetingen van alle onderdelen van de formule. Als al deze afmetingen bekend zijn, kan de formule in de juiste vorm op het scherm getekend worden.



a– – –>b: programma-activiteit verloopt van a naar b

a- - ->b: informatiestroom verloopt van a naar b

a · · · >b: uit a wordt (een deel van) b gegenereerd

1: invoer was EQN-commando

2: invoer van formule

3: invoer was STOP-commando

FIGUUR 4.1

In ieder geval, wanneer de invoer verwerkt is, wordt op de volgende invoer gewacht. Als de gebruiker het STOP-commando geeft, wordt deze loop verlaten, en zijn we terug in module 'main'. Deze ruimt verder het een en ander op, en stopt tenslotte het programma. Bovenstaand verhaal is in fig. 4.1 in tekening gebracht.

4.1. De module 'main'
Deze module zorgt voor het initialiseren van de diverse modules, het opstarten van de lees-verwerk-print-loop en het laten eindigen van het programma. Het is belangrijk dat de modules in de juiste volgorde worden geïnitialiseerd, d.w.z. eerst 'table', dan 'tree' en tenslotte 'input'. Dit is nodig omdat b.v. 'tree' gegevens uit de tabel, die 'table' aanmaakt, gebruikt.

4.2. De module 'table'
De grammatica staat extern van het programma op een file. Het programma moet informatie uit de grammatica gebruiken. Deze module zorgt er dan ook voor dat de benodigde informatie uit de grammatica in een tabel wordt opgeslagen. De rest van het programma kan via een aantal routines over deze informatie beschikken.

4.2.1. Datastructuur. In de tabel moet informatie komen over iedere TERMINAL, NTERMINAL en CLAUSE uit de grammatica. Een NTERMINAL heeft een productie-regel met een aantal symbolen erin, en iedere CLAUSE heeft een rij symbolen: deze symbolen noemen we in het programma items. Het maximum aantal items per NTERMINAL of CLAUSE is MAXITEMS. Het startsymbool STARTSYM staat ook in de tabel: deze wordt opgevat als een bijzondere NTERMINAL. Het heeft als item het startsymbool van de grammatica.

De informatie is per soort symbool verschillend. In tabel 4.2 staat een overzicht van deze informatie. Zie voor de grammatica appendix I.

entrytype	class	priority	assoc	items-count	items	syms-count	allowed
TERMINAL	. . .	MAX	NONE	0	ILL	. . .	. . .
NTERMINAL	. . .	. . .	. . .	. . .	. . .	. . .	. . .
STARTSYM	START	-1	NONE	1	. . .	0	. . .
CLAUSE	ILL	START = MIN EXPR = MAX rest = ILL	ILL	. . .	. . .	0	. . .

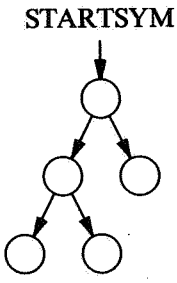
ILL: illegale waarde
. . .: waarde die in de grammatica is ingevuld
rest: default waarde

TABEL 4.2

Ieder symbool in de grammatica krijgt een index toegewezen. Deze index geeft het rij-nummer in de tabel. Het maximum aantal rijen is MAXENTRYS. Het toewijzen van een index aan een symbool gebeurt door 'get_index()'. Deze procedure moet veranderd worden als er een symbool in de grammatica verandert, verdwijnt of bijkomt. Deze procedure wordt dan ook automatisch uit de grammatica gegenereerd door het programma 'make_proc'. Dit programma leest de grammatica en geeft iedere terminal, non-terminal of clause een uniek getal, de index.

4.3. De module 'tree'
De formule die de gebruiker invoert, wordt intern gerepresenteerd in een boomstructuur. Een deel van het programma moet dus zorgen voor het creëren, manipuleren en opruimen van deze boomstructuur. Dit moet zodanig gebeuren, dat dit deel details van de feitelijke implementatie verbergt, en er voor zorgt dat de rest van het programma in termen van de boomstructuur de formule bewerkt.

4.3.1. Datastructuur. De abstracte datastructuur van de formuleboom ziet er als volgt uit:



De wortel is een unieke knoop: deze heeft geen ouders en heeft als inhoud STARTSYM. De wortel heeft één kind: dit kind representeert de hele formule.

Iedere knoop in de boom bevat informatie over zijn inhoud en heeft relaties met zijn ouder, naaste broers en kinderen. Eén knoop heet de huidige knoop: deze wordt aangegeven door een boomcursor, genaamd 'active'.

Iedere knoop kan een lijst van 0 of meer symbolen hebben. Deze lijst van symbolen bevat informatie over het uiterlijk van de formule, terwijl de boom de structuur van de formule weergeeft.

4.3.2. Routines. De belangrijkste routine van deze module, en eigenlijk van het hele programma, is de routine 'tree_process()', met de twee hulproutines 'process_token()' en 'process_end()'. Deze routines sturen het opbouwen van de boom aan de hand van de invoer van de gebruiker en m.b.v. informatie uit de grammatica. Deze routines zullen we zeer gedetailleerd bespreken.

'tree_process()' krijgt een token (interne code) van de module 'input'. Is het token een END-commando, dan wordt 'process_end()' aangeroepen. Anders wordt een deel van de formule ingevoerd.

In 'prev_class' wordt de klasse van de voorgaande invoer bijgehouden. Als de huidige invoer volgens de grammatica niet mag volgen na de vorige invoer, dan probeert de routine of het tussenvoegen van de concatenatie-operator, tussen de vorige en de huidige invoer, een toegestane invoer oplevert. Als dat zo is, dan wordt dus eerst de concatenatie-operator aan 'process_token()' gegeven en dan de huidige invoer; zo niet, dan wordt alleen de huidige invoer aan 'process_token()' gegeven. In dit laatste geval ontstaat in de boom een EXPR-knoop, die aangeeft dat op deze plaats nog andere invoer wordt verwacht.

Nu zullen we de routines 'process_token()' en 'process_end()' in detail beschrijven.

'process_token()'

Deze routine hangt een token op in de boom. Ook wordt 'active' aangepast: het is de bedoeling dat 'active' steeds aangeeft waar de volgende invoer in de boom moet worden geplaatst. Het is handig om nu eerst drie termen te verklaren:

Knoop 1 past in knoop 2, als knoop 2 een EXPR-knoop is, en knoop 1 associativiteit NONE heeft. Knoop 1 kan b.v. een terminal zijn. Een EXPR-knoop geeft aan dat er nog invoer op de plaats van deze knoop wordt verwacht, en een knoop met associativiteit NONE voldoet aan deze verwachting.

Een knoop heet open, als de knoop een EXPR- of START-knoop is. Een open knoop is dus een knoop die nog verder uitgewerkt kan worden.

Twee knopen komen overeen, als ze dezelfde prioriteit en associativiteit hebben. Een '+'-knoop en een '−'-knoop komen dus overeen.

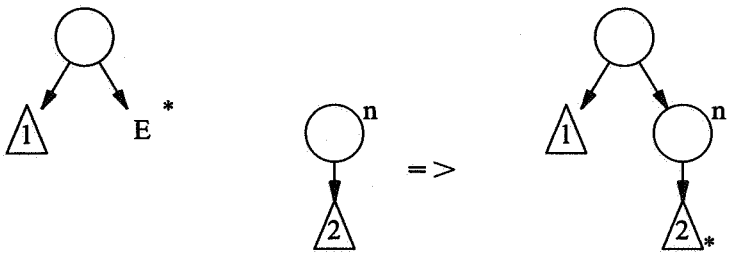
De routine 'process_token()' wordt uitgelegd aan de hand van een aantal voorbeelden en tekeningen. Deze voorbeelden hebben betrekking op de grammatica van Appendix I. In de tekeningen wordt de volgende notatie gebruikt: met 'n' wordt de nieuwe knoop aangegeven; met '*' wordt de huidige knoop aangegeven; een knoop met inhoud '>' (of '<', '≤', etc.) geeft een knoop aan met prioriteit groter dan die van de nieuwe knoop; een streepjes-pijl geeft aan dat de aanhangende knoop er *kan*

zijn; een driehoek geeft een deelboom aan; 'E' is een EXPR-knoop; 'S' is een start-knoop.

De tekeningen bestaan uit drie delen: het eerste deel is de huidige boom, het tweede deel is de boom voor de invoer, dan een '=>' en tenslotte de boom die ontstaat door de invoer in de huidige boom te verwerken.

Nu de werking van de routine. Er is dus een token ingevoerd. Er wordt eerst een nieuwe knoop voor dit token gemaakt. Deze knoop kan kinderen hebben. De informatie over de eventueel aan te maken kinderen wordt uit de grammatica gehaald: de nieuwe knoop heeft de linkerkant van een productie als inhoud, en de kinderen hebben ieder een item van de rechterkant van de productie als inhoud.

Deze nieuwe knoop met kinderen moet nu in de boom gehangen worden. Als de nieuwe knoop *past in* de huidige knoop, dan komt de nieuwe knoop met zijn kinderen op de plaats van de huidige knoop:



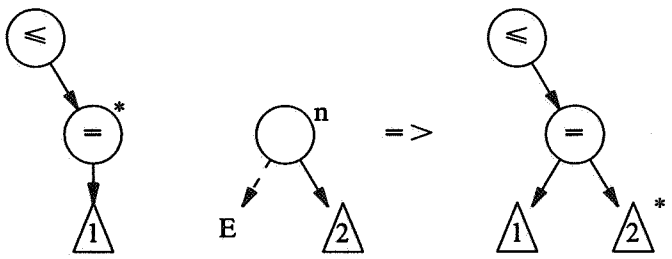
Als de nieuwe knoop niet past, wordt m.b.v. de prioriteit de plaats opgezocht waar de nieuwe knoop moet komen. Zolang de prioriteit van de nieuwe knoop kleiner is dan de prioriteit van de huidige knoop, gaat 'active' (d.w.z. de huidige knoop) de ouder van de huidige knoop aangeven. Dit stopt altijd, omdat de wortel van de boom, STARTSYM, prioriteit MIN (d.w.z. zo klein mogelijk) heeft. Wanneer dit gestopt is, geldt dat de prioriteit van de nieuwe knoop groter of gelijk is aan de prioriteit van de huidige knoop. Deze twee gevallen worden apart behandeld.

Gelijkheid:

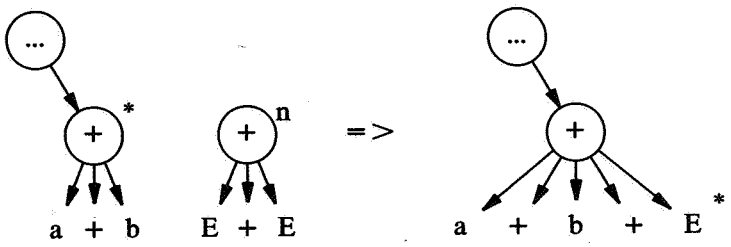
Om nu de plaats van de nieuwe knoop in de boom te bepalen, is de associativiteit van de nieuwe knoop van belang:

LEVEL:

de nieuwe knoop moet op hetzelfde niveau als de huidige knoop komen. Als het eerste kind van de nieuwe knoop een EXPR-knoop is, dan 'matched' deze met het laatste kind van de huidige knoop:

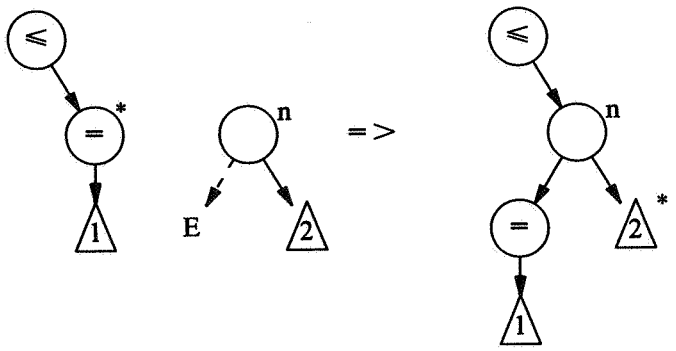


Bijvoorbeeld, neem als associativiteit van '+' de waarde LEVEL:

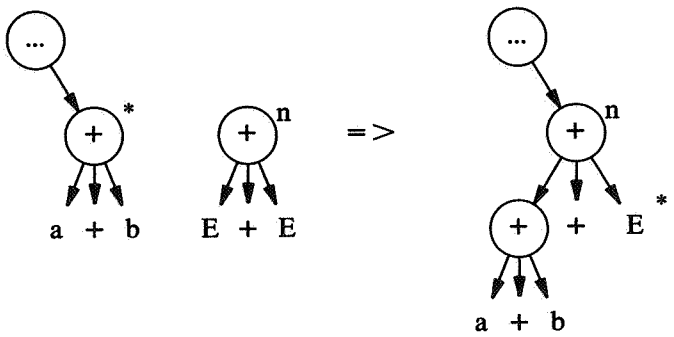


LEFT:

zolang de nieuwe knoop en de ouder van de huidige knoop *overeenkomen,* wordt de ouder van de huidige knoop nu zelf de huidige knoop. Bij LEVEL worden de kinderen van de nieuwe knoop toegevoegd, hier wordt de nieuwe knoop tussen de huidige knoop en zijn ouder gevoegd:

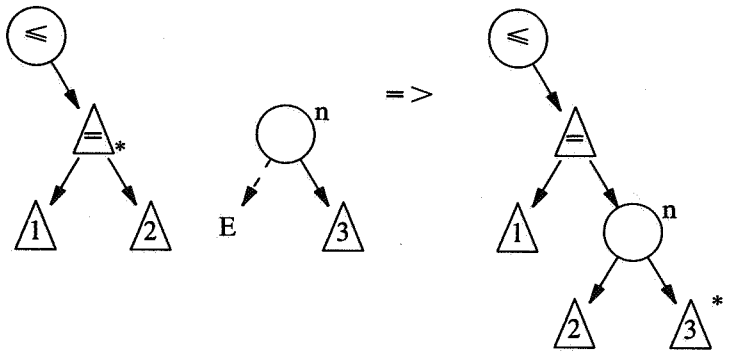


Bijvoorbeeld, met associativiteit LEFT voor '+':

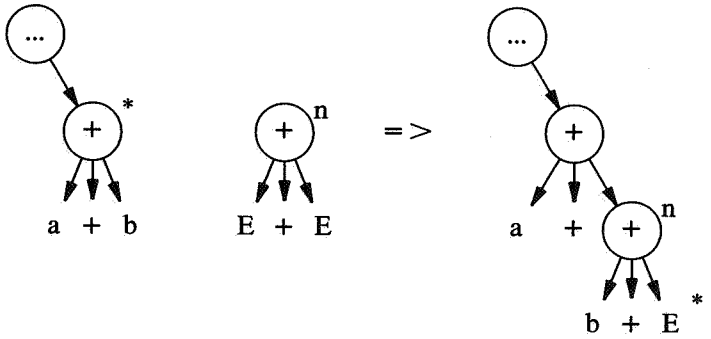


RIGHT:

dit geval is ongeveer het spiegelbeeld van LEFT: zolang de nieuwe knoop en het laatste kind van de huidige knoop *overeenkomen,* wordt het laatste kind van de huidige knoop nu zelf de huidige knoop. Daarna wordt de nieuwe knoop tussen de huidige knoop en z'n laatste kind gevoegd:



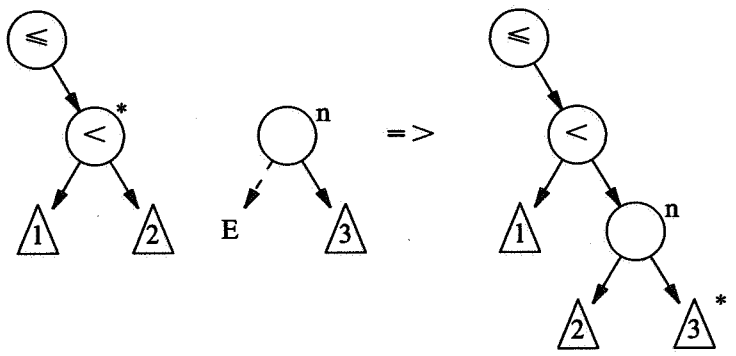
Bijvoorbeeld, '+' is nu rechts-associatief:



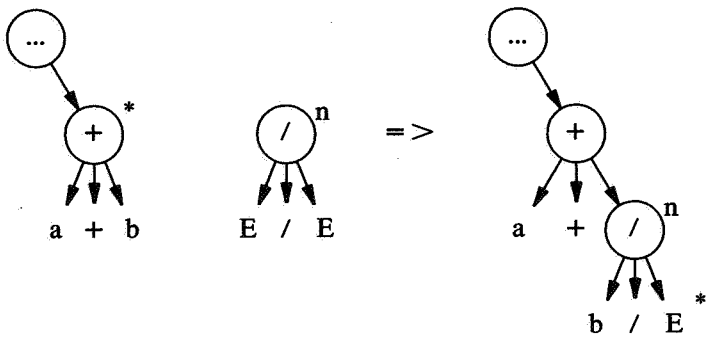
Groter dan:

De huidige knoop kan een START-knoop zijn (heeft een zo klein mogelijke prioriteit MIN). Dit is een bijzonder geval, want dit houdt in dat hier een op zichzelf staande formule moet komen (b.v. de ondergrens van een integraal). De rest van de boom mag dus geen invloed hebben op de plaats van de nieuwe knoop in de boom. De nieuwe knoop wordt in dit geval dus als kind van de START-knoop toegevoegd.

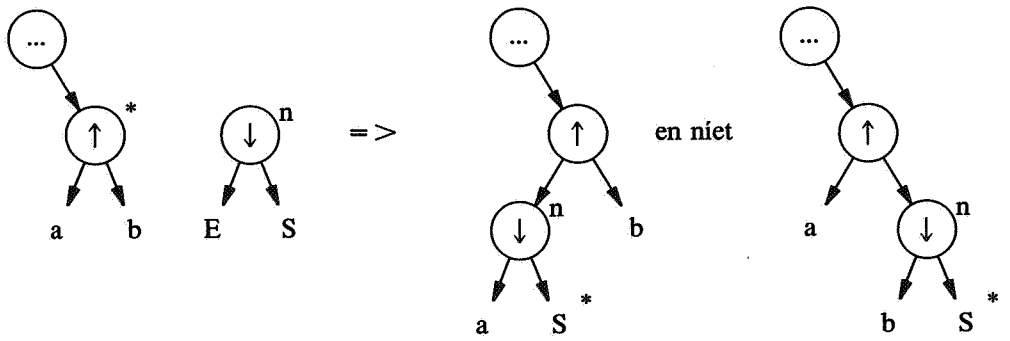
Als de huidige knoop geen START-knoop is, hebben we in feite dezelfde situatie als bij gelijkheid met associativiteit RIGHT:



Bijvoorbeeld, prioriteit van '+' is 1 en van '/' is 2:

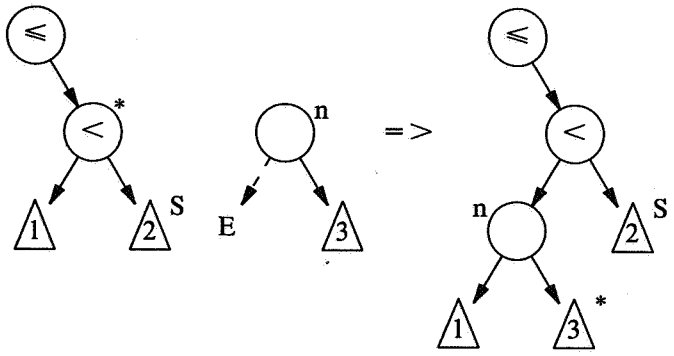


De nieuwe knoop wordt dus weer tussen de huidige knoop en z'n laatste kind gevoegd. Er is echter een uitzondering. Het volgende voorbeeld laat dit zien:



In dit voorbeeld is de huidige knoop 'machtsverheffen'; de macht (het tweede kind) is in de grammatica een START-item. Nu wordt 'index' ingevoerd. Deze heeft een hogere prioriteit dan machtsverheffen (de index hoort namelijk bij de '*a*', en niet bij de macht). Echter, de nieuwe knoop mag nu niet aan de rechtertak van de huidige knoop komen: deze tak hoort bij een START-item, en

de rest van de boom mag geen invloed uitoefenen op zo'n tak. Als dus het laatste item van de huidige knoop een START-item is, wordt de volgende actie gedaan:



Nu is het token in alle gevallen in de boom verwerkt. Tenslotte moet nog 'prev_class' de waarde van de klasse van dit token krijgen, zodat de volgende invoer op geldigheid gecontroleerd kan worden. Als de huidige knoop echter een START-knoop is (d.w.z. nu wordt er met een op zichzelf staande formule begonnen), dan wordt 'prev_class' op START gezet.

'process_end()'

Wanneer een END-commando is gegeven, is de formule die aan een START-knoop hangt, beëindigd en moet deze START-knoop verwijderd worden. Verder moeten 'active' en 'prev_class' aangepast worden.

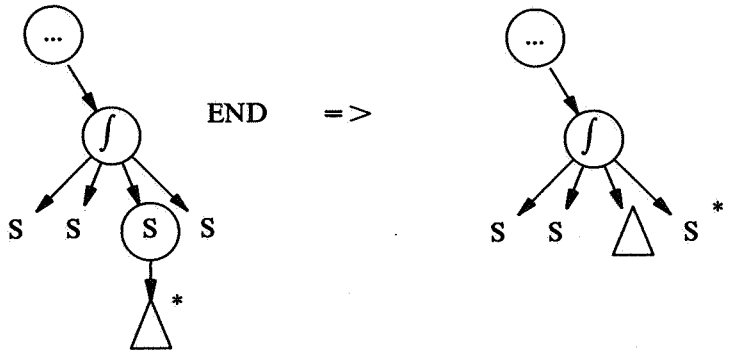
De gebruiker heeft dus een END-commando gegeven. Als de huidige knoop een START-knoop is, betekent dit dat de gebruiker op deze plaats geen subformule heeft ingevoerd (anders zou 'active' wel ergens in de subformule naar een knoop ongelijk aan een START-knoop wijzen). Deze START-knoop wordt dan ook niet verwijderd: het representeert een plaatsaanduiding van een niet-verplichte subformule (b.v. een grens van een integraal).

Als de huidige knoop geen START-knoop is, dan wordt 'active' net zolang op z'n ouder gezet, totdat de huidige knoop wél een START-knoop is. Deze START-knoop heeft één ouder en één kind. De START-knoop wordt uit de boom verwijderd en z'n kind wordt het kind van de ouder van de START-knoop.

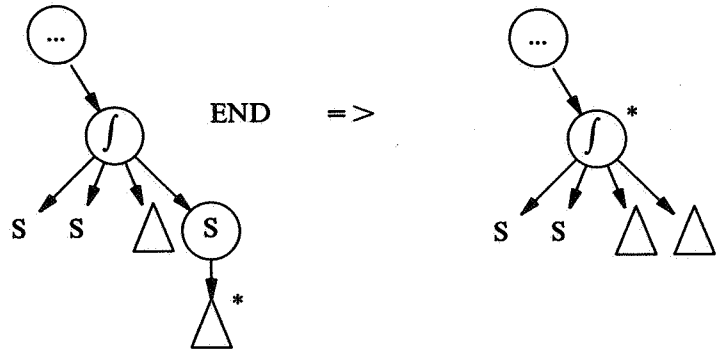
Nu moet 'active' naar de juiste knoop gaan verwijzen. Dit hangt af van de associativiteit van de ouder van de (verdwenen) START-knoop:

NONE of LEVEL:

'active' wordt op de rechter broer van de (verdwenen) START-knoop gezet. Is er geen rechter broer, dan op de ouder. Neem b.v. de integraal. De gebruiker heeft de grenzen leeg gelaten en beëindigd nu de integrand:

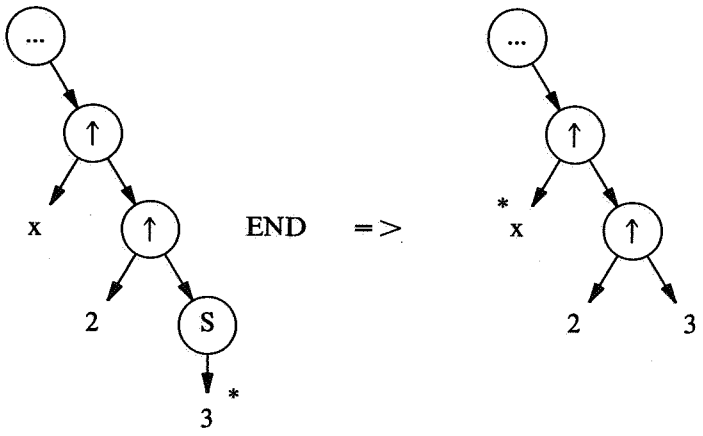


De gebruiker vult nu de integraal-variabele in en geeft een END:



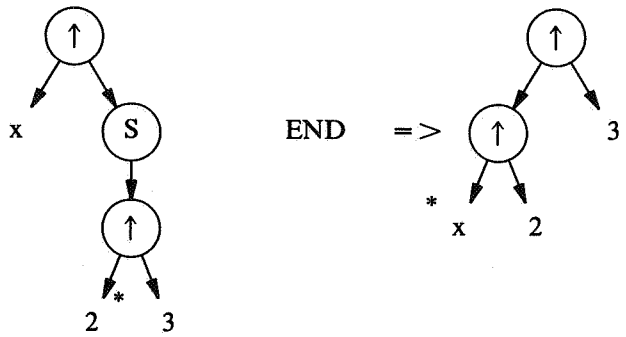
RIGHT:

beginnend op de ouder van de START-knoop, wordt 'active' op zijn ouder gezet zolang deze *overeenkomen.* Dan wordt 'active' op z'n eerste kind gezet. Bijvoorbeeld,

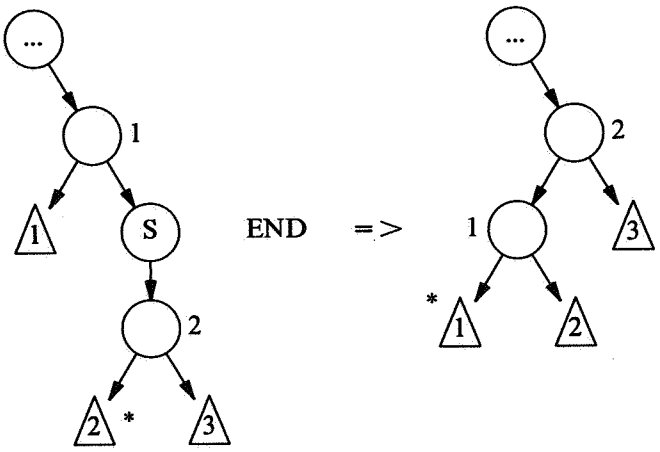


LEFT:

Dit geval is ongeveer het spiegelbeeld van RIGHT: beginnend op de ouder van de START-knoop, wordt 'active' op z'n eerste kind gezet zolang deze twee *overeenkomen.* Daarna wordt 'active' op z'n eerste kind gezet. Vóór dit alles, moet echter, als de ouder en het kind van de START-knoop *overeenkomen,* de boom nog aangepast worden. Bijvoorbeeld, neem nu associativiteit van '↑' als LEFT:



In het algemeen ziet dit er als volgt uit:



Tenslotte wordt 'prev_class' aangepast: dit gaat op dezelfde manier als bij 'process_token()'. Als de huidige knoop nu echter de wortel van de boom is, dan was de laatste END de END van de hele formule, en niet van een deelformule. Het programma moet dan van de READ-toestand naar de EDIT-toestand overgaan. De huidige knoop moet in dit geval de knoop worden die de hele formule representeert, dus 'active' gaat nu naar het eerste (en enige) kind van de wortel wijzen.

4.4. De module 'sizes'

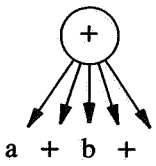
In deze module worden afmetingen bepaald. Op het laagste niveau zijn dit afmetingen van karakters, d.w.z. de hoogte, diepte en breedte van een box om het karakter heen. Het verkrijgen van deze afmetingen is afhankelijk van het systeem waar het programma op draait.

Op een hoger niveau moeten de afmetingen van de box om een hele formule worden uitgerekend bij een bepaalde puntgrootte. Het omgekeerde moet ook kunnen: bepaal bij een gegeven afmeting van een symbool, de puntgrootte die het symbool (ongeveer) die afmeting geeft.

4.4.1. Datastructuur. De datastructuur die deze module gebruikt is een font. Een font heeft een naam en een index. De toegestane namen zijn "I" (Italic), "R" (Roman) en "S" (Special). In het font staat informatie over karakters, zoals afmetingen en vormgeving van karakters. De informatie over één bepaald karakter is bereikbaar door een index in het font te gebruiken. Meestal is de index van een karakter de karaktercode van dat karakter. Een afmeting van het karakter 'a' in het font "I" is dus te verkrijgen door het font "I" te indiceren met de karaktercode van 'a' = 97 (voor ascii).

4.4.2. Routines. Voor de andere modules is de routine 'box()' beschikbaar: deze vult van een knoop de afmetingen in van de box om de betreffende formule. De routine doet dit door zichzelf recursief aan te roepen om de afmetingen van de deelformules en symbolen te berekenen. Deze routine wordt uit de display-regels gegenereerd door het programma 'make_display'. Dit is een vrij rechtstreeks proces: als in de display-regels '$<nr>' voorkomt, wordt de code 'ith_child(node, <nr>)' gegenereerd, en '@<nr>' levert de code 'ith_sym(node, <nr>)' op.

De associativiteit LEVEL levert een probleem op. Neem b.v. de operator '+' met associativiteit LEVEL en de productieregel '+: EXPR + EXPR'. In de display-regels staat o.a '$0.w = $1.w + $2.w + $3.w;', d.w.z. de breedte van de hele formule is gelijk aan de som van de breedtes van de deelformules. Neem nu de formule '*a* + *b* + *c*' met de boom:



De interne knoop '+' heeft vijf kinderen, terwijl in de display-regels er echter maar met drie rekening wordt gehouden. Dit wordt opgelost door in 'box()' niet voor '$1.w' de code 'ith_child(node, 1)' te genereren, maar de code 'ith_child(node, from + 1)'. 'from' is een integer variabele. Deze wordt in een loop als volgt gebruikt:

Bij de eerste iteratie heeft 'from' de waarde '0'. Dan worden de afmetingen van kinderen 1, 2 en 3 berekend. Bij de tweede iteratie is 'from' opgehoogd met het aantal items van de huidige knoop min 1, dus voor '+' is 'from' opgehoogd met 2. Nu worden de afmetingen van kinderen 3, 4 en 5 berekend (dus kind 3 komt, helaas, twee keer aan de beurt). Nu alle kinderen aan de beurt zijn geweest, stopt de loop.

De variabele 'from' geeft aan vanaf waar de kinderen geteld moeten worden.

4.5. De module 'input'

Deze module zorgt voor de invoer en andere beeldscherm-afhankelijke processen (behalve de uitvoer). De module initialiseert het scherm, leest invoer van de muis of het toetsenbord, en converteert deze invoer naar een interne code, zodat de rest van het programma met beeldscherm-onafhankelijke invoer kan werken. Ook is op deze manier een verandering van de gebruikers-interface beperkt tot deze module.

Welke formule-commando's het programma kan verwerken is afhankelijk van de grammatica. Het programma 'make_proc' genereert dan ook voor deze module hulp-routine's die grammatica-afhankelijke commando's inlezen.

Verder zorgt deze module voor het beschikbaar zijn van fonts voor de rest van het programma en voor de opvang van externe signalen.

4.5.1. Datastructuur. Gebruikte fonts worden in een queue opgeslagen. Vaak gebruikte fonts staan vóór in de queue en blijven geopend.

De reden van deze organisatie is, dat fonts op schijf staan en dat het dus veel tijd kost om een font te openen (d.w.z. van de schijf in te lezen). Zo'n geopend font neemt echter weer veel interne geheugenruimte in beslag, dus het maximum aantal geopende fonts (d.w.z. het maximum aantal queue-elementen = MAXQUEUE) is beperkt.

De afhandeling van signalen ('signals', zoals een interrupt) is weer beeldscherm- en computer-afhankelijk. De module moet in ieder geval zorgen dat de formule weer opnieuw getekend wordt, wanneer een signaal is gegeven dat aangeeft dat de formule op het scherm beschadigd is geraakt.

4.6. De module 'output'

Deze module zorgt voor de uitvoer. De uitvoer-routines zijn in drie categoriën te verdelen:

- routines die beeldscherm-afhankelijk zijn
- routines voor het tekenen van de formule m.b.v de boomstructuur
- routines voor het berekenen van de posities van een (deel)formule. Dit gebeurt door 'position()'. Deze wordt, op een zelfde manier als de routine 'box()' (zie module 'sizes'), gegenereerd uit de grammatica door het programma 'make_display'. Ook hier moet met het genereren rekening worden gehouden met de associativiteit LEVEL.

4.7. De module 'eqn'
Het programma maakt van een willekeurige formule een interne representatie. De eqn-preprocessor kan van een willekeurige formule een afdruk op papier maken. De module 'eqn' zorgt nu voor een omzetting van de interne representatie naar eqn-code, zodat een interactief-ontworpen formule op papier kan worden afgedrukt door het batch-systeem 'eqn'. De eqn-code wordt op de file "eqn.uit" geschreven.

Routines in deze module moeten veranderd worden als in de grammatica producties veranderen of bijkomen. Het is echter niet goed mogelijk om routines voor deze module automatisch uit de grammatica te genereren. Dit moet dus met de hand gebeuren.

5. Implementatie

In dit hoofdstuk beschrijven wij de huidige stand van de formuleverwerker, zie sectie 5.1. De mogelijke verbeteringen en veranderingen aan ons systeem behandelen wij in sectie 5.2. Als laatste onderdeel van dit hoofdstuk behandelen wij de mogelijke uitbreidingen, zie sectie 5.3.

5.1. Huidige stand van het systeem

Het systeem werkt op een SUN-1 werkstation. Dit is een computer met een grafisch beeldscherm, d.w.z. het scherm heeft een redelijke resolutie (aantal puntjes op het scherm per oppervlakte), en ieder puntje (pixel) is adresseerbaar (bit-mapped). De mogelijkheden van deze computer, nl. de muis, het grafische pakket om het scherm te besturen en het toetsenbord, worden door ons gebruikt.

5.1.1. De grammatica. Het systeem werkt met een externe grammatica (zie hoofdstuk 3). Deze grammatica bestuurt het systeem, d.w.z. geeft informatie over de lay-out van de formules, maar geeft ook informatie over de toegestane formules.

Het uitbreiden van de grammatica is simpel te doen, het programma zelf hoeft niet aangepast te worden. De gebruiker hoeft alleen een beschrijving van de structuur en de lay-out van de gewenste formule te geven. Deze beschrijving is volgens bepaalde regels opgebouwd (zie Appendix III).

Na het veranderen van de grammatica genereert het systeem 3 procedures en 2 "include" files. Op deze "include" files staan de definities van de interne code's. Het programma moet wel opnieuw vertaald worden. Dit vertalen kan gebeuren door 'run' met een optie aan te roepen. De plaats van de verandering bepaalt de optie die gebruikt moet worden.

-a: De source-code van 'make_proc' en 'make_display', (deze programma's genereren met behulp van de grammatica procedures voor de in- en uitvoer-afhandeling), zijn beide veranderd. Eventueel zijn ook procedures van het systeem veranderd.
-d: De source-code van 'make_display', het programma om de procedures voor de uitvoer te genereren, is veranderd.
-p: De display-regels in de grammatica zijn veranderd. De procedures voor de uitvoer afhandeling moeten opnieuw gegenereerd worden.
-m: Alleen de source-code van een procedure van het systeem is veranderd.
-c: De source-code van 'make_proc', het programma om de index te genereren, is veranderd.
-g: De grammatica-regels zijn veranderd, maar de display-regels niet.

Het opstarten van het systeem gebeurt door het commando 'inform' in te tikken. Het systeem kan alleen vanuit een SUN-window opgestart worden.

5.1.2. De invoer. De invoer maakt gebruik van de mogelijkheden van de SUN-1 computer. Op het grafische beeldscherm definiëren wij een scherm ('screen'). Dit scherm bestaat momenteel uit vier delen, drie optie-windows en een display-window.

De optie-windows bestaan uit een rechthoek gevuld met woorden of symbolen tussen haken. Deze woorden heten opties. De speciale karakters staan niet tussen haken maar worden vooraf gegaan door een gradient-teken (∇). Dit is een eigenaardigheid van het SUN-systeem.

De optie-windows bevatten de commando's (b.v. INT), de speciale karakters (b.v. α) en de

standaardfuncties (b.v. sin) als opties. Dit is nodig om commando's e.d. in te voeren, de commando's zijn niet als toets op het toetsenbord aanwezig. Alle opties zijn tergelijkertijd te zien.

De gebruiker kan zijn keus kenbaar maken door de gewenste optie met de cursor van de muis aan te wijzen, en de selectieknop op de muis in te drukken. De selectieknop is de linker of middelste knop op de muis. De optie verschijnt in 'reverse video' (zwarte achtergrond met witte letters). Laat de gebruiker de knop los terwijl de muiscursor op de optie blijft staan, dan is deze optie 'geclicked', d.w.z. de optie wordt intern doorgegeven.

Op het display-window tekent het programma de formule. Alleen als de muiscursor in het display-window staat, kan de gebruiker karakters van het toetsenbord als invoer geven.

5.1.3. De uitvoer. Voordat een formule getekend kan worden moeten alle posities en afmetingen bekend zijn. Het berekenen van de afmetingen en de posities gebeurt door vanaf de wortel van de formuleboom, deze waarden voor alle knopen uit te rekenen (zie hoofdstuk 4).

Als de posities van iedere deelformule en ieder symbool bekend zijn, kan de formule getekend worden. Dit gebeurt door functies uit het grafische pakket (zie [13, 14]) te gebruiken. Voor de gebruikte functies, zie Appendix I van de documentatie. Een gedeelte van de formule (soms de gehele formule) zal opnieuw getekend moeten worden. Dit houdt in dat het symbool op de positie waar het nu staat wordt gewist. Om deze wis-operatie zo weinig mogelijk tijd te laten kosten, worden waar het kan pixel-operaties toegepast. Deze pixel-operaties zijn bit-operaties en kosten minder tijd dan een gedeelte van het scherm te overschrijven met de inverse formule. Voor het overschrijven moeten namelijk de karakters waaruit de formule bestaat uit de fonts worden gehaald. Na het wissen van gedeelten van de formule verschijnt deze in zijn geheel weer op het scherm door de gewiste delen op de nieuwe posities te tekenen.

Het is ook mogelijk altijd de gehele formule te wissen en opnieuw te tekenen, maar dit geeft een flikkerend beeld omdat het tekenen veel tijd in beslag neemt. Dit komt voor de gebruiker zeer onrustig over.

5.1.4. De edit-toestand. Na het beëindigen van de invoer voor de formule kan de gebruiker in de huidige versie alleen nog maar door de formule lopen met een cursor. De gebruiker kan met de cursor bepaalde gedeelten van de formule aanwijzen. De aangewezen delen komen in 'reverse video' te staan. De toegestane operaties zijn: 'inzoomen', 'uitzoomen', 'naar rechts lopen' en 'naar links lopen'. De commando's hievoor zijn respectievelijk 'd' (down), 'u' (up), 'r' (right) en 'l' (left).

5.1.5. De toegestane formules. De toegestane formules bestaan op dit moment alleen uit integraal ('$\int$'), wortel ('$\sqrt{\ }$'), breuk ('$-$'), haakjes (rond '(' en recht '['), groeihaakjes (eveneens rond '(' en recht '['), standaardfuncties ('sin' en 'cos'), operatoren en letters uit het alfabet. Het alfabet is beperkt tot 26 letters 'a'-'z', de 10 cijfers '0'-'9', 3 griekse letters 'α', 'β' en 'γ' en 3 griekse hoofdletters 'A', 'B' en 'I'. De operatoren zijn '=', '+', '-', '$\times$', '/', '.', '$\downarrow$', '$\uparrow$' en concatenatie.

5.2. Verbeteringen

Het systeem is op vele punten nog voor verbetering vatbaar. Dit komt omdat we voor het project een half jaar de tijd hadden en er nog niets op het CWI aan een formuleverwerker was gedaan. We geven in deze paragraaf de delen aan, die volgens ons nog verbeterd moeten worden. Indien mogelijk wordt ook aangegeven wat de verbetering kan zijn.

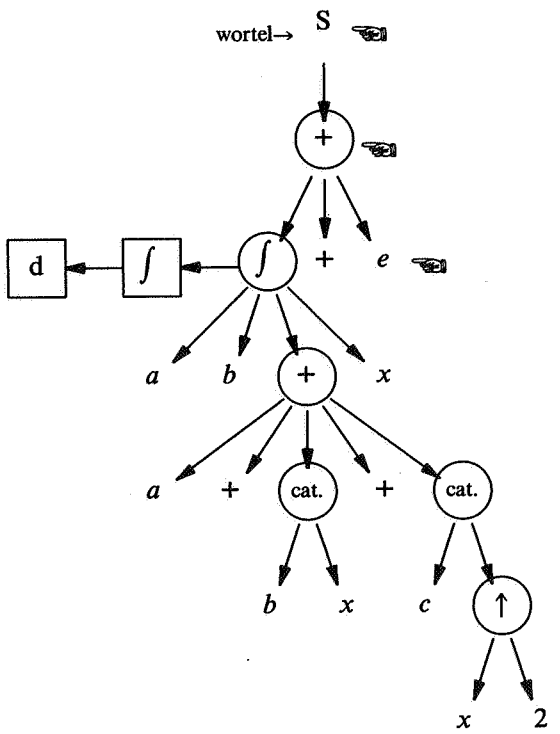
5.2.1. De invoer. De manier van invoer is nog niet optimaal, zo moet de muis in het display-window staan om invoer van het toetsenbord mogelijk te maken. De volgorde van de opties is ad hoc gekozen. Het is waarschijnlijk beter om veel gebruikte opties vooraan te zetten. De plaats van de opties wordt door het systeem bepaald, alle opties staan daarom naast elkaar. Dit werkt niet erg prettig. De plaats van de optie-windows zelf zou voor verbetering in aanmerking komen. In plaats van dat ze boven het display-window staan, is het ook mogelijk ze rechts of links van het display-window te zetten.

Aangezien ons systeem deel moet gaan uitmaken van een geïntegreerde interactieve tekstverwerker, hebben we weinig aandacht geschonken aan de huidige "user-interface".

5.2.2. De uitvoer. De bepaling van de afmetingen van een (deel)formule gebeurt door bij de wortel van de formuleboom te beginnen en van álle knopen de afmetingen opnieuw te berekenen. Dit is voor een ingewikkelde formule rekenintensief, voornamelijk omdat voor ieder 'groeisymbool' de puntgrootte opnieuw bepaald moet worden. Voor deze bepaling moeten fonts geopend worden en juist dit openen kost veel tijd.

Een betere oplossing zou zijn daar in de boom te beginnen waar de nieuw ingevoerde knoop staat en alléén de afmetingen van de voorouders en de bij de voorouders behorende symbolen te berekenen. Voor de formule (5.1) geldt dat bij invoer van 'e' alleen die knopen, die een '☜' hebben in fig. 5.1, veranderende afmetingen kunnen hebben.

$$\int_a^b a + bx + cx^2 \, dx + e \tag{5.1}$$



FIGUUR 5.1

Dit heeft tot gevolg dat de afmetingen van het $\int$-teken niet opnieuw berekend worden, en juist deze operatie is rekenintensief.

Een moeilijkheid van het berekenen op deze manier is dat de puntgrootte waarin 'e' getekend moet worden, bekend moet zijn. Er wordt van een blad naar de wortel van de boom gegaan. De puntgrootte echter wordt juist van de wortel naar de bladeren doorgegeven. De voorouders van 'e' hebben invloed op de puntgrootte en moeten dus bezocht zijn, voordat de puntgrootte in 'e' bekend is.

Het tekenen en wissen van een formule zou versneld kunnen worden door gedeeltes van een

formule te verplaatsen. De karakters van dat gedeelte hoeven dan niet uit de fonts gehaald te worden. De bitrepresentatie van het scherm wordt dan aangepast door gedeelten van de bitmap te verplaatsen, en deze operatie gaat sneller dan het openen van de fonts voor de karakters. Hierbij moet wel rekening gehouden worden met het feit dat gedeelten elkaar niet moeten overlappen. Dit houdt in dat er een volgorde bepaald moet worden om gedeelten te verplaatsen, zodat een gedeelte van de formule niet gezet wordt over een nog niet verplaatst gedeelte.

5.2.3. De overige verbeteringen. Het systeem sneller te laten werken is een extra verbetering. Het versnellen van het programma kan gebeuren door alle fonts die het programma kan gebruiken open te houden. Waarschijnlijk werkt het ook al sneller als het berekenen van afmetingen, het wissen en het tekenen van een formule efficienter gaat.

Een verbetering behelst ook het programma tegen foute invoer bestand te laten zijn. Op dit moment stopt het programma bij foute invoer. Het zou beter zijn de foute invoer te negeren en een waarschuwing te geven. Verder moet de grammatica gecontroleerd worden (b.v. associativiteit LEVLE is fout; '$4' mag niet voorkomen bij een NTERMINAL als deze maar 3 items heeft, enz.).

5.3. De uitbreidingen

5.3.1. De grammatica. De grammatica heeft zeker een uitbreiding nodig. Een aantal uitbreidingen zijn makkelijk, dwz. het programma hoeft niet aangepast te worden. Dit is het geval voor toevoegen van constructies met een vast aantal deelformules, b.v. vereniging, doorsnede, enz. Voor andere uitbreidingen moet het programma wel veranderd worden. Dit is het geval als bv. nieuw toe te voegen constructies geen vast aantal deelformules hebben, zoals matrices. De lay-out beschrijving kan niet met de aanwezig zijnde functies beschreven worden. Verder moeten er van één formule meerdere representaties mogelijk zijn. Dit laatste is nuttig als er verschil in de lay-out gemaakt is voor als een formule in een tekstregel voorkomt of op een aparte regel staat.

5.3.2. De editor. Er is nu nog geen editor aanwezig. Een noodzakkelijke uitbreiding is dus om deze toe te voegen. De editor zal alleen simpele commando's als delete, insert en add hebben en géén voor formules specifieke commando's. B.v. als de index van een sommatie verandert, is het mogelijk alle indices van de sommant eveneens te veranderen. De sommatie over '*i*' verandert in de sommatie over '*j*', dit heeft tot gevolg in voorbeeld 5.2 dat de index van '*x*' in een '*j*' verandert.

$$\sum_{i=1}^{n} x_i \quad => \quad \sum_{j=1}^{n} x_j \tag{5.2}$$

Aangezien de formuleverwerker deel gaat uitmaken van een geïntegreerd geheel, is het de bedoeling algemeen toepasbare edit-commando's te definiëren en niet speciale edit-commando's voor formules.

5.3.3. De opslag van de formule. Er is op dit moment nog geen enkele mogelijkheid om een ingevoerde formule op te slaan en later met deze formule verder te werken. Een code voor de formule moet op een file gezet worden en later weer ingelezen worden. Deze code moet de structuur van de formule representeren. Eén mogelijkheid is om voor deze code de 'eqn'-code te gebruiken. Het systeem kan al 'eqn'-code genereren voor de huidige formule. Als het systeem deze code ook nog kan inlezen is dit een goede opslag manier. De 'eqn'-code representeert de formule op een unieke manier. Een andere lineaire representatie zie [9] is ook mogelijk.

6. SAMENVATTING EN CONCLUSIES

6.1. Het huidige systeem en de doelstellingen

In het kader van onze afstudeeropdracht hebben we een half jaar aan het project gewerkt. Een half jaar is gebruikelijk voor een afstudeeropdracht, maar voor zo'n project eigenlijk nog te kort. Toch hebben we al veel van onze doelstellingen (zie hoofdstuk 2) bereikt.

- de gebruiker kan commando's en symbolen invoeren, en op ieder moment staat op het beeldscherm de formule in zijn huidige vorm getekend: het systeem is dus interactief. Soms moet de gebruiker echter enkele seconden wachten voordat invoer van symbolen met variabele afmetingen (zoals '$\int$') verwerkt zijn. De formule kan niet veranderd worden, wel kan de gebruiker met een cursor door de ingevoerde formule 'lopen'.
- via een externe grammatica kan op een redelijk eenvoudige manier de formule met zijn lay-out beschreven worden. Als deze beschrijving verandert, veranderen automatisch enkele delen van het systeem mee. Deze delen moeten dan wel opnieuw gecompileerd worden.
  Deze doelstelling was moeilijk om te bereiken, en heeft dan ook een aanzienlijk deel van onze tijd in beslag genomen.
- de grammatica bepaalt éénduidig het gedrag van het systeem.
- de lay-out kan goed bepaald worden, maar dit geldt niet voor alle gevallen. B.v. 'x_i^2' wordt gezet als 'x_i ^{2}', omdat het kwadraat rechtsboven de denkbeeldige box van de grondexpressie ('x_i') staat, i.p.v. rechtsboven de denkbeeldige box van 'x'. Blijkbaar is de huidige manier van lay-out beschrijving niet volledig.
- aangezien de formuleverwerker deel gaat uitmaken van één geïntegreerde documentverwerker, is nog niet veel aandacht geschonken aan de huidige "user-interface". Het systeem is ondanks dit niet moeilijk om te (leren) gebruiken.
- wanneer het systeem een ander soort beeldscherm zou moeten gebruiken, hoeft slechts een beperkt deel van het systeem opnieuw geschreven te worden. We denken niet dat ons systeem hierbij moeilijkheden oplevert. Gezien onze ervaringen met de SUN en de bijbehorende matige documentatie denken we eerder dat er moeilijkheden zullen zijn, om met het software-pakket van de beeldscherm-fabrikant te leren werken. Het heeft ons redelijk veel tijd gekost om dit met de SUN voor elkaar te krijgen.

6.2. Het huidige systeem en de toekomst

Zoals ook in hoofdstuk 5 is aangegeven, kan er nog veel gedaan worden aan de grammatica die het systeem stuurt: een rijker scala aan formules (zoals matrices), een krachtiger lay-out beschrijving die meer mogelijkheden biedt, en per formulebeschrijving meerdere lay-out beschrijvingen, die afhangen van de context waarin de formule staat.

Een interactieve formuleverwerker die gestuurd wordt door een externe grammatica, zoals ons systeem, is vrij uniek. Toen we met onze opdracht begonnen, bestond zoiets voor zover we weten nog niet. Pas in de laatste maand van onze opdracht (tijdens het schrijven van dit verslag) lazen we een pas uitgegeven artikel ([10]) over een systeem dat verder in deze richting gaat.

Verder is ons systeem een prototype. Wellicht kunnen gedeeltes beter geprogrammeerd worden: een wachttijd van enkele seconden is voor een interactief systeem niet gewenst. Tenslotte moet het systeem met een editor uitgebreid worden; hierbij moet, net als bij de "user-interface", rekening worden gehouden met het feit dat de formuleverwerker onderdeel wordt van één documentverwerker.

6.3. Slotwoord

De opdracht hebben we uitgevoerd op de afdeling 'programmatuur' van het CWI. We vonden dit een prettige afdeling om bij te werken.

Tijdens de opdracht heeft Hans van Vliet ons begeleid. Veel ideëen die we gebruiken zijn van hem afkomstig. Ook heeft hij geholpen om dit verslag in een acceptabele vorm te krijgen. Bij deze danken wij hem hartelijk voor de prettige en opbouwende manier van begeleiden.

Ook willen we Jos Warmer bedanken, die ons vaak met raad en daad heeft bijgestaan.

REFERENCES

1. M. HAMMER ET AL. (June 1981). The Implementation of Etude, an Integrated and Interactive Document Production System. *Proceedings of the ACM SIGPLAN/SIGOA Symposium on Text Manipulation*, 137-146.
2. F. A. H. VAN HARMELEN (1983). *On the Implementation of an Editor for the B Programming Language,* report IW 220/83, Department of Computer Science, Mathematical Centre, Amsterdam.
3. B. W. KERNIGHAN and L. L. CHERRY (March 1975). A System for Typesetting Mathematics. *Communications of the ACM18.3*, 151-157.
4. D. E. KNUTH (February 1968). Semantics of Context-Free Languages. *Mathematical Systems Theory2.2*, 127-145.
5. D. E. KNUTH (1979). *TEX and METAFONT: New Directions in Typesetting*, Digital Press.
6. M. LEVISON (February 1983). Editing Mathematical Formulae. *Software-Practice-Experience13.2*, 189-195.
7. R. MEDINA-MORA (March 1982). *Syntax-Directed Editing: Towards Integrated Programming Environments,* Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh.
8. A. NIENHUIS (1983). *On the Design of an Editor for the B Programming Language,* report IW 248/83, Department of Computer Science, Mathematical Centre, Amsterdam.
9. V. QUINT (March 1983). An Interactive System for Mathematical Text Processing. *Technology and Science of Informatics2.3*, 169-179.
10. V. QUINT and I. VATTON (April 1986). Grif: An Interactive System for Structured Document Manipulation. *Proceeding of the Conference on Text Processing and Document Manipulation*, Nottingham, England.
11. B. K. REID (January 1980). A High Level Approach to Computer Document Formatting. *Conference Report of the Seventh Annual ACM Symposium on Principles of Programming Languages*, 24-31.
12. R. SCHENKMAN (1978). *The Typing of Mathematics*, Repro Handbook, California.
13. SUN MICROSYSTEMS INC (November 1983). *Programmers Reference Manual for the Sun Window System*, California.
14. SUN MICROSYSTEMS INC (January 1984). *Sun Window's Programmers Guide*, California.
15. D. A. WATT and O. L. MADSEN (February 1983). Extended Attribute Grammars. *The Computer Journal26.2*, 142-153.
16. K. WICK (1965). *Rules for Type-setting Mathematics*, Mouton & Co., Den Haag.

Appendix I

Huidige Grammatica

Dit is de grammatica zoals die per 18-03-1986 in het programma gebruikt wordt. De grammatica begint met een STARTSYM te geven. Daarna komen de TERMINALS, deze hebben als attribuut alleen de klasse. Na de TERMINALS staan de NTERMINALEN. Deze hebben als attributen respectievelijk: de klasse, de prioriteit, de associativiteit en de klassen van de items. Na de NTERMINALEN komen de CLAUSES. Deze hebben per klasse een lijst van klassen.

De verschillende attributen staan gescheiden door een komma. De items en de elementen van de klassenlijst staan gescheiden door spaties. Komma's en dubbele punten moeten omringd staan door spaties.

```
STARTSYM        : START

TERMINAL
a               : CHAR
b               : CHAR
c               : CHAR
d               : CHAR
0               : CHAR
1               : CHAR


NTERMINAL
+               : OPERATOR ,    1    , LEVEL ,    EXPR + EXPR
-               : OPERATOR ,    1    , LEVEL ,    EXPR - EXPR
*               : OPERATOR ,    2    , LEVEL ,    EXPR * EXPR
EMPTY           : OPERATOR ,    4    , LEVEL ,    EXPR EXPR
SUP             : COP ,         5    , RIGHT ,    EXPR START
INT             : CONSTRUCT ,   10   , NONE ,     START START START START
(               : CONSTRUCT ,   10   , NONE ,     START
GBR             : CONSTRUCT ,   10   , NONE ,     START


CLAUSE
```

```
COP             : OPERATOR COP
CONSTRUCT       : OPERATOR COP
CHAR            : OPERATOR COP
OPERATOR        : CHAR CONSTRUCT
START           : CHAR CONSTRUCT
EXPR            : CHAR CONSTRUCT
```

Appendix II

Huidige Grammatica met Display-regels

De huidige grammatica met de display-regels. "R" staat voor het romanfont, het special font wordt met "S" aangegeven.

```
STARTSYM     : START #

TERMINAL
a            : CHAR , #
b            : CHAR , #
c            : CHAR , #
d            : CHAR , #
0            : CHAR , #
             @1.f = "R";
             @1.i = "0";
             #
1            : CHAR ,
             @1.f = "R";
             @1.i = "1";
             #
NTERMINAL
+            : OPERATOR ,  1 , LEVEL , EXPR + EXPR ,
             print($1, 0, 0, 1.0);
             print_next($2, $1, 1.0);
             print_next($3, $2, 1.0);
             $0.w = $1.w + $2.w + $3.w;
             $0.h = max(max($1.h, $2.h), $3.h);
             $0.d = max(max($1.d, $2.d), $3.d);
             #
-            : OPERATOR ,  1 , LEVEL , EXPR - EXPR ,
             print($1, 0, 0, 1.0);
             print_next($2, $1, 1.0);
             print_next($3, $2, 1.0);
             $0.w = $1.w + $2.w + $3.w;
```

```
            $0.h = max(max($1.h, $2.h), $3.h);
            $0.d = max(max($1.d, $2.d), $3.d);
            #
*           : OPERATOR , 2 , LEVEL , EXPR * EXPR ,
            print($1, 0, 0, 1.0);
            print_next($2, $1, 1.0);
            print_next($3, $2, 1.0);
            $0.w = $1.w + $2.w + $3.w;
            $0.h = max(max($1.h, $2.h), $3.h);
            $0.d = max(max($1.d, $2.d), $3.d);
            #
EMPTY       : OPERATOR , 4 , LEVEL , EXPR EXPR ,
            print($1, 0, 0, 1.0);
            print_next($2, $1, 1.0);
            $0.w = $1.w + $2.w;
            $0.d = max($1.d, $2.d);
            $0.h = max($1.h, $2.h);
            #
SUP         : COP , 5 , RIGHT , EXPR START ,
            print($1, 0, 0, 1.0);
            print($2, $1.w, $1.h, 2.0/3);
            $0.w = $1.w + $2.w;
            $0.h = $1.h + $2.h;
            $0.d = $1.d;
            #
INT         : CONSTRUCT , 10 , NONE , START START START START ,
            print(@1, max(max($1.w - @1.w, $2.w - @1.w) / 2, 0), 0, 1.0);
            center_under($1, @1, 0.66);
            center_above($2, @1, 0.66);
            print($3, max(max($1.w - @1.w, $2.w - @1.w) / 2, 0) + @1.w * 3/4, 0, 1.0);
            print_next(@2, $3, 1.0);
            print_next($4, @2, 1.0);
            @1.f = "S";
            @1.i = "5";
            @1.h = max(max($3.h, $4.h) + 5, char.h + 5);
            @2.f = "R";
            @2.i = "d";
            @2.h = char.h;
            $0.h = max(max($3.h, $4.h), @1.h + $2.h + $2.d);
            $0.d = max(max($3.d, $4.d), @1.d + $1.h + $1.d);
            $0.w = max(max($1.w - @1.w, $2.w - @1.w) / 2, 0);
            $0.w = $0.w + @1.w * 3/4 + $3.w + @2.w + $4.w;
            #
(           : CONSTRUCT , 10 , NONE , START ,
            print(@1, 0, 0, 1.0);
            print_next($1, @1, 1.0);
            print_next(@2, $1, 1.0);
            @1.f = "R";
            @1.i = "(";
            @1.h = char.h;
            @2.f = "R";
```

```
            @2.i = 51";
            @2.h = char.h;
            $0.h = max(@1.h, $1.h);
            $0.d = max(@1.d, $1.d);
            $0.w = @1.w + @2.w + $1.w;
            #
GBR         : CONSTRUCT , 10 ,  NONE , START ,
            print(@1, 0, 0, 1.0);
            print_next($1, @1, 1.0);
            print_next(@2, $1, 1.0);
            @1.f = "R";
            @1.i = "(";
            @1.h = max($1.h + 5, char.h + 5);
            @2.f = "R";
            @2.i = 51";
            @2.h = max($1.h + 5, char.h + 5);
            $0.h = max(@1.h, $1.h);
            $0.d = max(@1.d, $1.d);
            $0.w = @1.w + @2.w + $1.w;
            #

CLAUSE
COP         : OPERATOR COP #
CONSTRUCT   : OPERATOR COP #
CHAR        : OPERATOR COP #
OPERATOR    : CHAR CONSTRUCT #
START       : CHAR CONSTRUCT #
EXPR        : CHAR CONSTRUCT #
END
```

Appendix III

Syntax van de Display-regels

In deze appendix wordt de vorm van de display-regels beschreven. Hierin worden de standaardfuncties met de bijbehorende parameters behandeld. Optionele delen staat tussen '{' en '}'.

- @<nr>{.<attr @>} : referentie naar een symbool.
  <nr>: nr ⩾ 1, nr is een element van de natuurlijke getallen
  <attr @>: de mogelijke waarden zijn:
    h : hoogte
    d : diepte
    w : breedte
    f : fontnaam
    i : index
- $<nr>{.<attr $>} : referentie naar een deelformule.
  <nr>: nr ⩾ 0, nr is een element van de natuurlijke getallen, $0 = linkerkant van de productieregel; $<nr> met nr ⩾ 1 zijn de deelformules
  <attr $>: de mogelijke waarden zijn:
    h : hoogte
    d : diepte
    w : breedte
- char.<attr $>: afmeting van het symbool in huidige puntgrootte.
- expressies bestaan uit: +, -, ×, /, $<nr>.<attr $>, @<nr>.<attr @>, char.<attr $>, een getal of een expressie tussen haakjes.
- assignment is: $<nr>.<attr $> = expressie,
  @<nr>.<attr @> = expressie.
- standaardfuncties:
  - int max(int n1, int n2): maximum van n1, n2.
  - print($<nr> of @<nr>, int x, int y, float psize): nr ⩾ 1,
    teken de eerste parameter op positie (x, y) met puntgrootte verandering 'psize'.
  - print_next($<nr> of @<nr>, $<nr> of @<nr>, float psize): nr ⩾ 1,
    teken de eerste parameter rechts naast de tweede parameter, met puntgrootte verandering 'psize'.
  - center_above($<nr> of @<nr>, $<nr> of @<nr>, float psize): nr ⩾ 1,
    teken en centreer de eerste parameter boven de tweede parameter, met puntgrootte verandering 'psize'.
  - center_under($<nr> of @<nr>, $<nr> of @<nr>, float psize): nr ⩾ 1,

teken en centreer de eerste parameter onder de tweede parameter, met puntgrootte verandering 'psize'.

- constanten:
  - HORBAR: horizontale streep
  - VERTBAR: verticale streep
- Terminalen zijn niet verplicht display-regels te hebben. Bij afwezigheid hiervan wordt ingevuld:
  print(@1, 0, 0, 1.0);
  @1.f = "I";
  @1.i = <ascii-waarde van terminaal>;
- Nonterminalen zijn *verplicht* display-regels te hebben.
- alle display-regels worden beëindigd met ';'
- na alle display-regels van één (non)-terminaal staat een '#'
- alleen één afmeting van een symbool moet ingevuld worden. De andere waarden worden erbij gezocht.
- volgorde van display-regels:
  - positiebepaling
  - informatie over symbolen
  - afmeting berekening van formule

Voor een voorbeeld van het gebruik van de display-regels zie sectie 3.4.3.