stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IN 24/83 JUNI

H.D.A. TAN

VLSI-ALGORITMEN VOOR HERKENNING VAN CONTEXT-VRIJE TALEN IN LINEAIRE TIJD

VLSI algorithms for recognition of context-free languages in linear time

kruislaan 413 1098 SJ amsterdam

VLSI-algoritmen voor herkenning van context-vrije talen in lineaire tijd

door

H.D.A. Tan†

SAMENVATTING

VLSI-versies van de bekende algoritmen van Cocke, Kasami en Younger en van Earley voor de herkenning van context-vrije talen worden gepresenteerd. Terwijl de sequentiële versies kubische tijdcomplexiteit hebben, gebruiken de VLSI-versies slechts lineaire tijd. De nadruk valt op een VLSI-versie van de Earley algoritme, die geen enkele eis aan de vorm van de grammatica stelt.

†TH Twente, Afdeling Electrotechniek.

Dit is het verslag van een stage op de Afdeling Informatica van het Mathematisch Centrum in de periode maart tot en met mei 1983. De stage is begeleid door J. Heering.

1. INLEIDING

Dit verslag gaat over het parallel herkennen van context-vrije talen. Er worden 2 sequentiële algoritmen gepresenteerd. De eerste ervan is ontwikkeld door Cocke, Kasami en Younger, terwijl de andere door Earley bedacht is. Deze sequentiële algoritmen hebben een berekeningstijd van maximaal cn^3, met n de lengte van de te herkennen zin en c een voldoend grote positieve constante. Daarnaast worden 3 VLSI-algoritmen behandeld, die bij de herkenning van parallellisme gebruik maken, en zo de berekeningstijd reduceren tot $O(n)$. Het eerste hoofdstuk behandelt de sequentiële versie en de VLSI-versie van de Cocke-Kasami-Younger algoritme en het tweede hoofdstuk gaat over de Earley algoritme.

NOTATIES EN DEFINITIES:

Definitie 1: Een context-vrije grammatica is een 4-tupel $G=(V, \Sigma, P, S)$, met
V een eindige niet-lege verzameling, het totale alfabet van nonterminals en terminals,
$\Sigma \subset V$ de niet-lege verzameling van terminals,
$V-\Sigma=N$ de verzameling van nonterminals,
$S \in N$ het startsymbool,
P een eindige verzameling van produktieregels van de vorm $A \rightarrow \alpha$, waar $A \in N$ en $\alpha \in V^*$.

Voor $\phi, \psi \in V^*$ geldt de relatie $\phi \overset{*}{\Rightarrow} \psi$, als $\phi=\psi$ of als een eindige rij $\omega_1, \omega_2, \ldots, \omega_n$ $(n>1, \omega_i \in V^*)$ bestaat zodat $\omega_1=\phi$, $\omega_n=\psi$ en $\omega_i \Rightarrow \omega_{i+1}$ voor $1 \leqslant i \leqslant n-1$.

Voor $\phi, \psi \in V^*$ geldt de relatie $\phi \overset{+}{\Rightarrow} \psi$, als $\phi \overset{*}{\Rightarrow} \psi$ en $\phi \neq \psi$.

Definitie 2: De taal, $L(G)$, gegenereerd door G is de verzameling:

$$L(G)=\{\omega \in \Sigma^* \mid S \overset{*}{\Rightarrow} \omega\}$$

Een zin is een $\beta \in L(G)$ en als $\beta=a_1 \cdots a_k$, $a_i \in \Sigma$, dan is k de lengte van β, $\lg(\beta)$.

Definitie 3: Een context-vrije grammatica $G=(V, \Sigma, P, S)$, met V, Σ, P en S als in definitie 1, is in Chomsky-normaalvorm als P bestaat uit produktieregels van de vorm:

i) $A \rightarrow BC$ met $B, C \in N$

ii) $A \rightarrow a$ met $a \in \Sigma$

iii) $S \rightarrow \Lambda$

Bovendien, als $S \rightarrow \Lambda \in P$, dan $B, C \in N-\{S\}$ in (i).

Definitie 4:
$O(f(n))=\{g(n) \mid$ er bestaan positieve constanten c, N_0, zodat $|g(n)| \leqslant cf(n)$, voor alle $n \geqslant N_0\}$

2. DE COCKE-KASAMI-YOUNGER ALGORITME

Cocke, Kasami en Younger hebben een sequentiële algoritme ontwikkeld voor de herkenning van context-vrije talen. De grammatica's van deze talen moeten echter wel in Chomsky-normaalvorm zijn (zie § 1, definitie 3) en mogen daarbij de lege produktie niet bevatten. Maar dit is principieel geen beperking, omdat elke context-vrije grammatica omgezet kan worden naar Chomsky-normaalvorm. Bovendien wordt de lege produktie niet gebruikt bij de generatie van zinnen, die niet leeg zijn. Deze algoritme heeft een berekeningstijd, die maximaal cn^3 is, als n de lengte van de te herkennen zin is en c een voldoend grote positieve constante is. Chu en Fu [1] hebben echter een VLSI-versie van het Cocke-Kasami-Younger algoritme ontwikkeld, die gebruik maakt van parallelle berekeningen, om zo de berekeningstijd te reduceren tot $O(n)$. In dit hoofdstuk wordt de sequentiële algoritme, de VLSI-algoritme en de simulatie van de VLSI-algoritme behandeld. Ook worden er

enkele kanttekeningen bij het ontwerp van Chu en Fu geplaatst.

2.1. De sequentiële algoritme

***Algoritme* 1**

Stel

a) $G=(V, \Sigma, P, S)$ is een context-vrije grammatica in Chomsky-normaalvorm zonder de regel $S \rightarrow \Lambda$,

b) $w=a_1 a_2 \cdots a_n$, $a_k \in \Sigma$, $1 \leqslant k \leqslant n$, is de te herkennen zin.

Construeer de "strictly upper-triangular" $(n+1)\times(n+1)$ herkenningsmatrix **T**, waarvan elk element $t_{i,j}$ een -in het begin lege- deelverzameling van $V-\Sigma$ is, als volgt:

begin

loop1: **for** $i:=0$ **to** $n-1$ **do**
$t_{i,i+1}:=\{A \mid A \rightarrow a_{i+1} \in P\}$;

loop2: **for** $d:=2$ **to** n **do**
for $i:=0$ **to** $n-d$ **do**
begin
$j:=i+d$;
$t_{i,j}:=\{A \mid$ er bestaat $k, i+1 \leqslant k \leqslant j-1$,
zodat $A \rightarrow BC \in P$ voor een $B \in t_{i,k}, C \in t_{k,j}\}$
end

end.

Er geldt dat $A \in t_{i,j}$ dan en slechts dan als $A \stackrel{+}{\Rightarrow} a_{i+1} \cdots a_j$ (zie Graham, et al., [5]). Dus $w \in L(G)$ dan en slechts dan als $S \in t_{0,n}$. Merk op, dat *loop2* de elementen van de matrix **T** per diagonaal berekent en dat alle matrixelementen in die diagonaal onafhankelijk van elkaar berekend worden. Een diagonaal $\{t_{i,j} \mid j-i=d\}$ kan pas zijn eindresultaten bereiken als alle eindresultaten van de diagonalen $\{t_{i,j} \mid j-i=g\}$ met $g<d$ bekend zijn. Stel dat een paring binnen een zekere vaste tijd (stel r) en onafhankelijk van n kan gebeuren en dat het resultaat van diagonaal $d-1$ klaar is op tijdstip t_{d-1}. Dan is het eindresultaat van diagonaal d op zijn vroegst op tijdstip $t_d=t_{d-1}+r$ klaar. Omdat er maar n van die diagonalen zijn, zou dus de matrix klaar kunnen zijn in een tijd $O(n)$. Dit geeft ruwweg het principe voor het VLSI-algoritme van Chu en Fu, dat in de volgende paragraaf wordt besproken.

2.2. VLSI-algoritme 1

Algoritme 1 vereist voor de berekening van een matrixelement $t_{i,j}$ alle paringen van matrixelementen $t_{i,k}$ en $t_{k,j}$, met $i+1 \leqslant k \leqslant j-1$. Guibas, et al., [3] hadden echter al een parallelle VLSI-algoritme voor soortgelijke paringen die in lineaire tijd werkt. Het dataflow-patroon en de algemene structuur van die chip zijn dan ook letterlijk door Chu en Fu overgenomen. De VLSI-algoritme wordt hier behandeld in drie delen, namelijk de data-representatie, het VLSI-ontwerp en de preprocessing.

2.2.1. Data-representatie

Chu en Fu stellen de volgende data-representaties voor. Alle verschillende nonterminals worden opeenvolgend genummerd. Dus de verzameling $\{S, A\}$ wordt $\{A_1, A_2\}$, met $A_1=S$ en $A_2=A$. Daarmee is het mogelijk een verzameling van nonterminals te coderen in een binair woord ter lengte van het aantal verschillende nonterminals. Bijvoorbeeld: Stel het aantal nonterminals is gelijk aan 4 dan wordt $\{A_1\}$ gecodeerd als 1000. Evenzo met de verzameling van terminals. Stel nu, er zijn s

verschillende nonterminals $A_1, \ldots, A_s$. Daarvan worden dan alle permutaties van nonterminal-paren $A_1A_1, \ldots, A_1A_s, \ldots, A_sA_1, \ldots, A_sA_s$ berekend. Van elk paar wordt onderzocht door welke nonterminal(s) dit paar geproduceerd wordt. Deze nonterminal(s) worden dan gecodeerd in een binair woord ter lengte van het aantal nonterminals (in dit voorbeeld s). Bijvoorbeeld: Stel $A_1 \rightarrow A_2A_3$ en $A_4 \rightarrow A_2A_3$. Dan wordt A_2A_3 geproduceerd door A_1 en A_4. Als s gelijk is aan 4, dan wordt $\{A_1, A_4\}$ gecodeerd als 1001. Op deze wijze wordt dan een tabel verkregen met als ingang een binair woord van lengte s^2. Elk bit van dit woord correspondeert met een nonterminal-paar en dient als sleutel voor de tabel. De waarde bij die sleutel is dan een s-bits woord, waarin gecodeerd is de verzameling van nonterminal(s) die dit paar produceert. Deze tabel (met de dimensie $s^2 \times s$), die eigenlijk de codering van de produktieregels van de grammatica is, wordt gebruikt bij dit VLSI-ontwerp voor de berekening van de matrixelementen.

De te herkennen zin, w, wordt als volgt gecodeerd. Voor elke voorkomende terminal wordt de verzameling van nonterminal(s), die deze terminal produceert, bepaald en op bovenstaande wijze in een s-bits woord gecodeerd. Dit leidt dan tot lg(w) van dergelijke woorden.

2.2.2. VLSI-ontwerp

De struktuur van de chip voor deze VLSI-algoritme is dezelfde als de "strictly upper-triangular" matrix uit §2.1 . Zie figuur 1. Elk element $t_{i,j}$ in de herkenningsmatrix correspondeert met een cel $c_{i,j}$ in de struktuur. Alle cellen zijn wat betreft functies en "layout" identiek aan elkaar. Deze regelmaat vergemakkelijkt de "layout" van de chip. Een andere voorwaarde voor een goede VLSI-algoritme is een eenvoudig dataflow-patroon tussen de cellen onderling en operaties binnen de cellen, die eenvoudig zijn en in constante tijd plaatsvinden.

Dataflow-patroon. Dit is in het kort als volgt: Als de afstand tussen een cel $c_{i,j}$ (met i,j resp. de kolom- en rij-index) en de hoofddiagonaal is gelijk aan $t = j - i$ (in Algoritme 1 d genoemd), dan is het resultaat van $c_{i,j}$ klaar op tijdstip $2t$. Want een cel kan maximaal 2 paringen tegelijkertijd uitvoeren en $c_{i,j}$ moet $2(t-1)$ elementen paren. De cel transporteert dan zijn resultaat naar zijn boven- en rechterburen, zodat andere cellen het resultaat alvast kunnen gebruiken. Een cel $c_{i,j}$ begint dan te rekenen op tijdstip $2t - \frac{1}{2}(t-1)$ als t oneven is of op tijdstip $2t - \frac{1}{2}t$ als t even is. Eerst is de transportsnelheid van dit resultaat 1 cel per tijdseenheid. Na t tijdseenheden zal de snelheid vertraagd worden naar 1 cel per 2 tijdseenheden en dit blijft zo totdat de berekening van de matrix beëindigd is. Bijvoorbeeld: $c_{4,9}$ vereist een paring van $c_{4,5}$ en $c_{5,9}$; $c_{4,5}$ is klaar met zijn berekening op tijdstip 2. Het resultaat wordt op tijdstip 3 vertraagd en is op dat moment in $c_{4,6}$. Het verschijnt dan in $c_{4,9}$ op tijdstip 9. $c_{5,9}$ is op tijdstip 8 klaar en het resultaat ervan bereikt $c_{4,9}$ op tijdstip 9. Dus zowel het resultaat van $c_{4,5}$ als dat van $c_{5,9}$ komen op hetzelfde tijdstip aan in $c_{4,9}$, waardoor $c_{4,9}$ de vereiste paring kan uitvoeren. Om dit dataflow-patroon te verkrijgen bezit elke cel 3 kanalen per buur om met zijn 4 buren te kunnen communiceren (zie figuur 1). Een kanaal, de "fast belt", transporteert data van de linker- resp. onderbuur van de cel via de cel zelf naar de rechter- resp. bovenbuur van de cel en wel met een snelheid van 1 cel per tijdseenheid. Evenzo voor het "slow belt" kanaal, de snelheid is dan echter 1 cel per 2 tijdseenheden. Het laatste kanaal dient als een controlelijn en vervoert slechts 1 bit. Om de "slow belt" en de "fast belt" te implementeren heeft elke cel 6 registers (zie figuur 2). Namelijk de 2 registers HF en VF, die resp. de horizontale en de verticale "fast belts" implementeren, en de 4 registers HS1, HS2, VS1 en VS2, die de "slow belts" implementeren. Elk register heeft een breedte van s bits (s is het aantal verschillende nonterminals) en transporteert in elke tijdseenheid zijn inhoud naar het register erna, terwijl het data ontvangt van het register ervoor. Tevens bezit elke cel een accumulator (ACCU) waarin het voorlopige resultaat van de cel bewaard wordt. Dit voorlopige resultaat wordt het eindresultaat op tijdstip $2t$ voor alle cellen $c_{i,j}$ met diagonaalindex $t = j - i$, want op dat tijdstip zijn alle paringen voor die cellen voltooid. Op tijdstip $2t$ wordt dan het eindresultaat door de datatransfer-module DT1 van de ACCU naar HF en VF ("fast belt") getransporteerd. De DT1 wordt gecontroleerd met behulp van de horizontale controlelijn (HCTL). De HCTL

verbindt een cel met zijn linker- en rechterbuur en transporteert slechts 1 bit van links naar rechts. Als het bit dat in de cel komt 1 is, dan wordt de DT1 geactiveerd. Om te zorgen dat het activatiebit in $c_{i,j}$ is op tijdstip $2(j-i)$, is de transportsnelheid van HCTL gelijk aan 1 cel per 2 tijdeenheden. De overblijvende controlelijn (VCTL) verbindt een cel met zijn onder- en bovenburen en transporteert 1 bit naar boven. De VCTL activeert de datatransfer-module DT2. Deze laatste wordt gebruikt om het eindresultaat van een cel te vertragen, dat wil zeggen van de "fast belts" (HF,VF) naar de "slow belts" (HS1,VS1). Het eindresultaat van $c_{i,j}$ is op het moment dat het vertraagd moet worden in $c_{p,q}$ met $q-p=2(j-i)$. Om dan te zorgen dat de VCTL op het goede moment en in de goede cel DT2 activeert, is de snelheid van deze controlelijn gelijk aan 2 cellen per 3 tijdseenheden. Het activatiesignaal arriveert dan in $c_{p,q}$ op het tijdstip $3(q-p)/2$, met $q-p$ een veelvoud van 2. Hiermee is dan het benodigde dataflow-patroon verklaard.

Functiemodule. Het resultaat van de cel wordt berekend door de functiemodule (FM). De cel kan per tijdseenheid slechts 2 paringen tegelijkertijd uitvoeren. Aan de hand van de inhoud van de registers HF, VF, HS1 en VS1, die de resultaten van $c_{i,k}$ en $c_{k,j}$ bevatten, worden voor $c_{i,j}$ door de FM de paringen uitgevoerd. Het verkregen resultaat wordt dan, verenigd met het eventueel aanwezig resultaat van vorige paringen (in ACCU), bewaard in ACCU. Figuur 3 laat het ontwerp van de FM zien. Elke FM bevat twee permutatiemodulen (PM1 en PM2), die identiek aan elkaar zijn, en een geheugenmodule (MM). Deze laatste module is eigenlijk een tabel, waarin de gecodeerde produktieregels aanwezig zijn (zie §2.2.1). PM1 en PM2 worden gebruikt om de paringen van nonterminals, die gecodeerd in HF, VF, HS1 en VS1 zitten, uit te voeren. Dit wordt gedaan doordat elk permutatiemodule de logische functie $a_{i,j}=b_i$ AND c_j voor $1 \leqslant i,j \leqslant s$ uitvoert met s het aantal verschillende nonterminals waarbij elke b_i van de horizontale registers (HF en HS1) en elke c_j van de verticale registers (VF en VS1) komt. De $a_{i,j}$ van het uitgangswoord, lengte s^2, wordt dan in dezelfde volgorde gerangschikt als dat ook bij de ingang van de gecodeerde produktietabel gebeurd is. Tevens worden de uitgangen van de permutatiemodulen met elkaar verenigd, dus van elk bit j van PM1 wordt de logische OR met bit j van PM2 uitgevoerd, en het resultaat aangeboden aan de ingang van MM, om zo van elk aanwezige paar nonterminals de producerende nonterminals op te zoeken. Dus voor elk bit in het ingangwoord van de MM, die 1 is (d.w.z. het corresponderende nonterminalpaar is aanwezig), wordt de s-bits waarde (d.i. de gecodeerde nonterminals die het betreffende paar produceren) opgezocht. De waarden en de inhoud van ACCU worden met elkaar verenigd (OR-operatie) en in ACCU bewaard. Het geheel moet natuurlijk geïnitialiseerd worden en veel daarvan kan tijdens de preprocessing gedaan worden.

2.2.3. Preprocessing

Het coderen van de nonterminals en de terminals zowel als het berekenen van de gecodeerde produktietabel kan gedaan worden voordat een berekening gestart wordt, immers deze hangen slechts af van de grammatica en niet van de te herkennen zin. Meestal zal deze preprocessing gedaan worden door de host computer, waarvan de chip als functionele eenheid onderdeel uitmaakt. Deze kan tevens *loop1* van algoritme 1 kan uitvoeren.

2.3. Simulatie

Het simulatieprogramma voor deze VLSI-algoritme is geschreven in de programmeertaal SUMMER, zie Klint [6]. Omdat deze algoritme vrij eenvoudig is, is besloten om bij de simulatie niet met binaire woorden maar met verzamelingen van nonterminals (d.i. hoofdletters) te werken. Het programma is verdeeld in 3 delen, namelijk het inlezen van de grammatica en de preprocessing, het printen van het resultaat en het simuleren van de dataflow en van de celakties. De eerste twee zijn vrij triviaal en voor de laatste kan van het SUMMER class-mechanisme gebruik gemaakt worden. Alle cellen zijn gedefinieerd als instanties van class *cel* en de operaties op dit datatype zijn de gesimuleerde functies van de functiemodulen, de datatransfer-modulen en de "fast" en "slow belts". Voor elke

tijdseenheid worden de posities van de HCTL- en VCTL-signalen berekend en de cellen met dezelfde positie gewaarschuwd. Daarna worden de cellen één voor één geactiveerd. Tevens worden de voorlopige resultaten geprint. Hiermee is het mogelijk, om totdat de herkenningsmatrix voltooid is, de toestanden van alle cellen op alle momenten te bekijken en te zien hoe de VLSI-algoritme werkt.

2.4. Kanttekeningen

Het VLSI-ontwerp van Chu en Fu vereist in elke cel een tabel (de geheugenmodule), die s^3 bits groot is, met s gelijk aan het maximale aantal verschillende nonterminals waarop bij de bouw van de chip gerekend is. Stel we definiëren nu twee tabellen. Tabel 1 dient om van elke nonterminal B de nonterminals A op te zoeken met de relatie $A \rightarrow BC$. D.w.z. B staat *links* aan de rechterzijde van de produktieregel. De andere tabel, tabel 2, dient dan om van elke nonterminal C de nonterminals A op te zoeken met de relatie $A \rightarrow BC$. D.w.z. C staat *rechts* aan de rechterzijde van de produktieregel. Dan worden van alle nonterminals in een horizontaal kanaal via tabel 1 de producerende nonterminals gegenereerd. Evenzo voor die van een verticaal kanaal, echter nu via tabel 2. De resultaten van tabel 1 en tabel 2 worden met elkaar verenigd door middel van een AND-operatie en dit resultaat is dan de verzameling van nonterminals, die nonterminal-paren BC met B van het horizontale kanaal en C van het verticale kanaal, produceren. Elk van de tabellen is slechts s^2 bits groot. We vervangen nu de permutatiemodulen en de geheugenmodule door de tabellen 1 en 2. Van zowel tabel 1 als tabel 2 zijn er elk 2 stuks benodigd, omdat de cel 2 paringen tegelijkertijd kan doen. Dus in totaal zijn er 4 tabellen, van elk s^2 bits groot. Samen is dat dus $4s^2$ bits. Dat betekent, dat wanneer s veel groter is dan 4, die 4 tabellen veel kleiner zijn dan de oorspronkelijke s^3 geheugenmodule. En daarbij blijven de operaties in een cel eenvoudig en duren constant. Dus op deze wijze is het mogelijk om het VLSI-ontwerp wat te verbeteren.

3. DE EARLEY ALGORITME

Earley ontwikkelde een sequentiële algoritme voor het herkennen van context-vrije talen. De grammatica's van deze talen kunnen alle vormen van context-vrije grammatica's zijn. Dit is het grote voordeel van Earley's algoritme in vergelijking met dat van Cocke, Kasami en Younger. De constructie van de afleidingsboom (die verder in dit verslag buiten beschouwing blijft) en het melden van fouten tijdens het herkenningsproces kunnen geheel plaatsvinden in termen van de oorspronkelijke syntactische categorieën.

De hier gebruikte algoritme is door S.Graham, et al., [5] uit Earley's oorspronkelijke algoritme ontwikkeld. Deze sequentiële algoritme heeft een maximale berekeningstijd cn^3, met n de lengte van de te herkennen zin en c een voldoend grote positieve constante. Deze algoritme wordt in dit hoofdstuk vergeleken met die van Cocke, Kasami en Younger. Ook wordt een VLSI-versie gepresenteerd, die de berekeningstijd reduceert tot $O(n)$. Deze VLSI-algoritme is door mij bedacht en wordt vergeleken met de VLSI-algoritme van Chiang en Fu [2]. Achtereenvolgens worden nu de sequentiële algoritme, de VLSI-algoritme, de simulatie van de VLSI-algoritme en een vergelijking tussen de verschillende VLSI-algoritmen behandeld.

3.1. De sequentiële algoritme

***Algoritme* 2.** S. Graham, et al., [5] ontwikkelden de volgende variant op Earley's algoritme.

Stel

a) $G=(V, \Sigma, P, S)$ is een context-vrije grammatica,

b) $w=a_1a_2\cdots a_n$, $a_k \in \Sigma$, $1 \leqslant k \leqslant n$ is de te herkennen zin.

c) Een dotted rule is gedefineerd als volgt. Laat $\cdot$ een metasymbool zijn dat geen element is van V. Dan is voor elke $A \to \alpha\beta \in P$, met $\alpha, \beta \in V^*$, $A \to \alpha \cdot \beta$ een dotted rule.

d) De functie PRED, die nonterminals associeert met verzamelingen van dotted rules, is gedefineerd als volgt:

$\text{PRED}(A) = \{B \to \alpha \cdot \beta \mid B \to \alpha\beta \in P \wedge \alpha \stackrel{*}{\Rightarrow} \Lambda\} \wedge (\exists \gamma \in V^*)[A \stackrel{*}{\Rightarrow} B\gamma]\}$.

e) De functie PREDICT, die verzamelingen van nonterminals associeert met verzamelingen van dotted rules, is gedefinieerd als volgt:

$\text{PREDICT}(X) = \bigcup_{A \in X} \text{PRED}(A)$.

De variabele SEEN heeft als waarden verzamelingen van nonterminals. Construeer nu de "upper-triangular" $(n+1) \times (n+1)$ herkenningsmatrix **T**, waarvan elk element $t_{i,j}$ een -in het begin lege- verzameling van dotted rules is, als volgt:

begin

$t_{0,0} :=$ PREDICT($\{S\}$);

loop1: **for** $j := 1$ **to** n **do**

begin

SEEN $:= \varnothing$;

loop2: **for** $i := j-1$ **downto** 0 **do**

begin

scanner: $t_{i,j} := \{B \to \alpha a \beta \cdot \gamma \mid B \to \alpha \cdot a\beta\gamma \in t_{i,j-1} \wedge a = a_j \wedge \beta \stackrel{*}{\Rightarrow} \Lambda\}$;

completer: $t_{i,j} := t_{i,j} \cup \{B \to \alpha B \beta \cdot \gamma \mid (\exists k, i+1 \leqslant k \leqslant j-1)$
$[B \to \alpha \cdot A\beta\gamma \in t_{i,k} \wedge A \to \sigma \cdot \in t_{k,j} \wedge \beta \stackrel{*}{\Rightarrow} \Lambda]\}$;

$t_{i,j} := t_{i,j} \cup \{B \to \alpha A \beta \cdot \gamma \mid B \to \alpha \cdot A\beta\gamma \in t_{i,i} \wedge \beta \stackrel{*}{\Rightarrow} \Lambda$
$\wedge ((\exists C \in N)[A \stackrel{*}{\Rightarrow} C \wedge C \to \sigma \cdot \in t_{i,j}])\}$;

SEEN $:=$ SEEN $\cup \{A \in N \mid B \to \alpha \cdot A\beta \in t_{i,j}\}$;

end;

predictor: $t_{j,j} :=$ PREDICT(SEEN);

end

Deze algoritme berekent de herkenningsmatrix kolom voor kolom en kan bovendien de berekening van $t_{j,j} :=$ PREDICT(SEEN) pas uitgevoeren als de vorige kolom berekend is. Dus voor een parallelle berekening is deze algoritme niet geschikt. Maar wanneer $t_{j,j} :=$ PREDICT(SEEN) verandert wordt in $t_{j,j} :=$ PREDICT(N), met $N = V - \Sigma$, dan is het mogelijk algoritme 2 te veranderen zodat een parallele berekening mogelijk wordt. Zonder de verzwakking van de predictor geldt dat $A \to \beta \cdot \sigma \in t_{i,j}$ dan en slechts dan als $S \stackrel{*}{\Rightarrow} a_1 \cdots a_i A \gamma$, $\gamma \in V^*$ en $\beta \stackrel{*}{\Rightarrow} a_{i+1} \cdots a_j$. Met de verzwakking geldt slechts dat $A \to \beta \cdot \gamma \in t_{i,j}$ dan en slechts dan als $\beta \stackrel{*}{\Rightarrow} a_{i+1} \cdots a_j$ geldt (zie Harrison [4]). De herschreven algoritme 2 ziet er dan als volgt uit:

***Algoritme* 3.** Laat G, w, T en PREDICT dezelfde betekenis hebben als bij algoritme 2. Construeer dan de herkenningsmatrix T als volgt:

begin

$t_{0,0}$:=PREDICT($\{S\}$);

loop1: **for** $i:=1$ **to** n **do**
$t_{i,i}$:= PREDICT(N);

loop2: **for** $d:=1$ **to** n **do**
for $i:=0$ **to** $n-d$ **do**
begin $j:=i+d$

scanner: $t_{i,j}:=\{B\rightarrow\alpha a\beta\cdot\gamma \mid B\rightarrow\alpha\cdot a\beta\gamma \in t_{i,j-1} \wedge a=a_j \wedge \beta\stackrel{*}{\Rightarrow}\Lambda\}$;

completer: $t_{i,j}:=t_{i,j}\cup\{B\rightarrow\alpha A\beta\cdot\gamma \mid (\exists k, i+1\leqslant k\leqslant j-1)$
$[B\rightarrow\alpha\cdot A\beta\gamma\in t_{i,k} \wedge A\rightarrow\sigma\cdot \in t_{k,j} \wedge \beta\stackrel{*}{\Rightarrow}\Lambda]\}$;
$t_{i,j}:=t_{i,j}\cup\{B\rightarrow\alpha A\beta\cdot\gamma \mid B\rightarrow\alpha\cdot A\beta\gamma\in t_{i,i} \wedge \beta\stackrel{*}{\Rightarrow}\Lambda$
$\wedge((\exists C\in N)[A\stackrel{*}{\Rightarrow}C \wedge C\rightarrow\sigma\cdot\in t_{i,j}])\}$;
end;

end

Merk op, dat $t_{0,0}$:= PREDICT($\{S\}$) en dat het eerste statement van de completer (zie algoritme 3) matrixelementen op dezelfde wijze paart als *loop2* van algoritme 1. Dus net als bij algoritme 1 kan bij algoritme 3 een diagonaal parallel berekend worden.
Ook kan men de gelijkenis van deze algoritme met algoritme 1 vrij gemakkelijk zien. Stel dat algoritme 3 gebruikt wordt met een grammatica in Chomsky-normaalvorm zonder de lege produktie, dan zijn er enkele dingen op te merken:

a) Omdat de lege produktie ontbreekt, wordt de "dot" slechts 1 positie naar rechts verplaatst.

b) PREDICT introduceert slechts dotted rules van de vorm $A\rightarrow\cdot BC$ of $A\rightarrow\cdot a$, met $A, B, C\in N$ en $a\in\Sigma$.

c) De scanner levert alleen iets op voor $i=0$, want de completer introduceert slechts dotted rules waar aan de rechterzijde van de pijl niets anders dan nonterminals staan en de scanner zelf levert een dotted rule met de dot aan het eind (dus onbruikbaar voor de "volgende" scanner). Daarom kan de scanner uit *loop2* gehaald worden en in een eigen loop geplaatst worden (met slechts 1 **for** statement).

d) Het eerste statement van de completer begint te werken bij $d=2$, want dan pas bestaan er waarden k, die voldoen aan de eis $i+1\leqslant k\leqslant j-1$.

e) In $t_{i,j}$ $(j>i)$ komen geen dotted rules voor met een dot aan het begin, want zowel de scanner als de completer voegen dotted rules toe waarvan de dot 1 symbool naar rechts is verplaatst.

f) Het eerste statement van de completer voegt slechts dotted rules toe van de vorm $A\rightarrow BC\cdot$. Nu zijn dotted rules in $t_{i,j}$ met de dot aan het eind niet van belang voor $t_{i,q}$ (met $q>j$) omdat de dot niet meer te verplaatsen is. Het tweede statement van de completer zorgt er dan voor dat deze regels eventueel verder ontwikkeld kunnen worden. Want als $A\rightarrow BC\cdot\in t_{i,j}$ dan kan het tweede statement regel(s) van de vorm $D\rightarrow A\cdot E$ opleveren.
Al met al is het mogelijk om algoritme 3 voor grammatica's in Chomsky-normaalvorm en zonder de lege produktie om te schrijven naar algoritme 4.

***Algoritme* 4.** Laat G, w, T en PREDICT dezelfde betekenis hebben als bij algoritme 2 en stel G is in Chomsky-normaalvorm zonder de lege produktie. Construeer dan de herkenningsmatrix T als volgt:

begin

$t_{0,0}$:=PREDICT({S});

loop1: **for** i:=1 **to** n **do**
  $t_{i,i}$:= PREDICT(N);

loop2: **for** i:=0 **to** $n-1$ **do**
  $t_{i,i+1}$:={$B\rightarrow a\cdot \mid B\rightarrow\cdot a \in t_{i,i} \wedge a=a_{i+1}$};
  $t_{i,i+1}$:=$t_{i,i+1}\cup${$B\rightarrow A\cdot C \mid B\rightarrow\cdot AC \in t_{i,i} \wedge A\rightarrow\sigma\cdot \in t_{i,i+1}$};

loop3: **for** d:=2 **to** n **do**
  for i:=0 **to** $n-d$ **do**
    begin j:=$d+1$;
      $t_{i,j}$:=$t_{i,j}\cup${$B\rightarrow AC\cdot \mid (\exists k, i+1\leqslant k\leqslant j-1)$
        $[B\rightarrow A\cdot C \in t_{i,k} \wedge C\rightarrow\sigma\cdot \in t_{k,j}]$};
      $t_{i,j}$:=$t_{i,j}\cup${$B\rightarrow A\cdot C \mid B\rightarrow\cdot AC \in t_{i,i} \wedge A\rightarrow\sigma\cdot \in t_{i,j}$};
    end

end

De overeenkomst tussen algoritme 4 en algoritme 1 is nu heel duidelijk te zien. *Loop2* van algoritme 4 komt overeen met *loop1* van algoritme 1 en *loop3* van algoritme 4 is equivalent met *loop2* van algoritme 1. *Loop1* van algoritme 4 en de tweede statements van *loop2* en *loop3* van algoritme 4 zijn nodig omdat algoritme 4 gebruikt maakt van dotted rules.
Zoals eerder opgemerkt kan bij algoritme 3 een diagonaal parallel berekend worden, dus net als bij algoritme 1 kan dan de hele matrix op parallelle wijze ontwikkeld worden om zo de berekeningstijd te reduceren tot O(n) (met n de lengte van de te herkennen zin).

3.2. VLSI-algoritme 2

Omdat er een grote gelijkenis van algoritme 3 met algoritme 1 is, is het onvermijdelijk dat de VLSI-versie van algoritme 3 ook een grote gelijkenis vertoont met die van algoritme 1. Bijvoorbeeld het eerste statement van de completer in algoritme 3 paart dezelfde matrixelementen als *loop2* van algoritme 1. Daarom is dan ook voor dezelfde VLSI-structuur en dataflow-patroon gekozen als die van Chu en Fu [1] bij algoritme 1. Alleen de data-representatie en het ontwerp binnen een cel is anders. Deze zijn door mij bedacht onafhankelijk van het ontwerp van Chiang en Fu [2], dat pas tegen het eind van de stage beschikbaar kwam. Achtereenvolgend zullen de VLSI-structuur, de data-representatie, het ontwerp van een cel en de preprocessing besproken worden.

3.2.1. VLSI-structuur en dataflow

Zoals eerder gezegd zijn de VLSI-structuur en het dataflow-patroon exact hetzelfde als die van algoritme 1. Dus de structuur is die van een "upper-triangular" matrix, die dezelfde $(n+1)\times(n+1)$ dimensie heeft als de herkenningsmatrix T in algoritme 3. Ook de implementatie van de kanalen en de datatransfer van de "fast" naar de "slow belt" zijn hetzelfde. Zo heeft elk cel $c_{i,j}$ (met i,j resp. de rij en kolom-index) ook 3 kanalen om met zijn buren te communiceren en de transportsnelheid van de data in die kanalen is dezelfde gebleven. Er is echter 1 kanaal toegevoegd, namelijk een invoerkanaal. Dit kanaal loopt parallel aan de verticale "slow belt" en heeft dezelfde transportsnelheid (nl. 1 cel per 2 tijdseenheden). Het is toegevoegd omdat de scanner (zie algoritme 3) informatie nodig heeft over de te herkennen zin. Dit kanaal transporteert in kolom j het j^e symbool ($=a_j$) van de te herkennen zin. Zie ook figuur 4.

3.2.2. Data-representatie

De data-representatie is door mij bedacht. Het is een vrij eenvoudige representatie. Een binair woord ter lengte van het aantal dotted rules (stel s) wordt gebruikt om een verzameling dotted rules te coderen. Elk bit van zo'n woord correspondeert met een specifieke dotted rule. Het is gelijk aan 1 als de betreffende dotted rule tot de verzameling behoort en anders 0. Deze representatie kan gebruikt worden om het ingangswoord en het uitgangswoord van een tabel te coderen. Een tabel heeft een s-bits ingangswoord en elk bit van het ingangswoord dient als sleutel voor de tabel. Elke sleutel is geassocieerd met een s-bit waarde. Het uitgangswoord van de tabel is het resultaat van de logische OR van alle waarden, waarvan de corresponderende sleutel gelijk aan 1 is. Bijvoorbeeld: Stel het ingangswoord is de gecodeerde verzameling van dotted rules met de dot niet aan het eind. Elke sleutel, die 1 is, correspondeert dan met een dotted rule van de vorm $A \rightarrow \alpha \cdot B\beta\gamma$. De waarde van zo'n sleutel kan dan zijn de gecodeerde verzameling van dotted rules van de vorm $A \rightarrow \alpha B\beta \cdot \gamma$, met $\beta \stackrel{*}{\Rightarrow} \Lambda$. Dus met deze tabel is het mogelijk om van een verzameling dotted rules van de vorm $A \rightarrow \alpha \cdot B\beta\gamma$, de verzameling van dotted rules van de vorm $A \rightarrow \alpha B\beta \cdot \gamma \in P$, met $\beta \stackrel{*}{\Rightarrow} \Lambda$ op te zoeken. Dit is natuurlijk ook mogelijk voor :

a) Om bij een verzameling van dotted rules $C \rightarrow \sigma \cdot$, de verzameling van dotted rules $A \rightarrow \omega \cdot C\beta$ op te zoeken.

b) Om bij een verzameling van dotted rules $C \rightarrow \sigma \cdot$, de verzameling van dotted rules $A \rightarrow \omega \cdot B\gamma$, met $B \stackrel{*}{\Rightarrow} C$ op te zoeken.

Met deze tabellen is het mogelijk om de scanner en de completer van algoritme 3 te implementeren.

3.2.3. Ontwerp van een cel

De cel bezit (zie figuur 4) registers om de "slow" en de "fast belts" te implementeren. Deze registers hebben dezelfde functies als die van de VLSI-versie van algoritme 1 en ook dezelfde namen (HF, VF, HS1, HS2, VS1 en VS2). Het invoerkanaal is net als de "slow belt" geïimplementeerd met behulp van twee registers, IN1 en IN2. Deze registers hebben dezelfde eigenschappen als VS1 en VS2. Verder bezit de cel tabellen om de scanner en completer acties uit te voeren en een accumulator (ACCU), waarin het resultaat van de berekening bewaard wordt. Zie §3.2.2. voor de beschrijving van de tabellen. Ook bezit de cel een register Tii waar PREDICT(N) (of PREDICT($\{S\}$) i.g.v. $c_{0,0}$) bewaard wordt. Om van een gecodeerde verzameling van dotted rules bepaalde dotted rules te selecteren heeft de cel enkele maskers. Deze maskers zijn s bits breed en elk bit correspondeert met een dotted rule. Is een bepaald dotted rule gewenst dan wordt het corresponderend bit op 1 gezet. Door deze maskers te verenigen met een binaire representatie van een verzameling van dotted rules (AND operatie), is het mogelijk om bepaalde bits van de representatie te selecteren. Bijvoorbeeld om van een verzameling van dotted rules de dotted rules $A \rightarrow \sigma \cdot$ te selecteren.
De cel berekent zijn resultaat als volgt. Per tijdseenheid paart hij twee paren matrixelementen met elkaar. De twee paren zitten dan in de verticale en horizontale "slow" en "fast belts". Uit algoritme 3 is duidelijk te zien dat de scanner en het tweede statement van de completer uitgesteld kunnen worden totdat de laatste 2 paren aan de beurt zijn, namelijk op het tijdstip $2(j-i)$ voor cel $c_{i,j}$, met i,j resp. de rij- en kolom-index. De scanner moet tot dit tijdstip uitgesteld worden omdat de data van $c_{i,j-1}$ dan pas in $c_{i,j}$ arriveert. Op dat tijdstip voert de cel alle 3 statements uit terwijl op de andere tijdstippen de cel slechts de eerste statement van de completer uitvoert. Het paart data van de horizontale "belt" met die van de verticale "belt". De inhoud van een verticale "belt" (d.i. een gecodeerde verzameling van dotted rules) dient dan als ingangswoord voor TABEL I (zie figuur 4). Met deze tabel worden dan voor elke dotted rule $A \rightarrow \sigma \cdot$ in die verzameling alle dotted rules $B \rightarrow \alpha \cdot A\beta\gamma$ in P gegenereerd. De output van TABEL I wordt dan door middel van een logische AND verenigd met de inhoud van een horizontale "belt". De verkregen output is dan het resultaat van $B \rightarrow \alpha \cdot A\beta\gamma$ in $c_{i,k}$ en $A \rightarrow \sigma \cdot$ in $c_{k,j}$ (zie algoritme 3), en deze wordt aangeboden aan de ingang van TABEL II. Deze tabel wordt gebruikt om van een dotted rule $B \rightarrow \alpha \cdot A\beta\gamma$ de verzameling van

dotted rules $B \rightarrow \alpha A \beta \cdot \gamma$, met $\beta \not\Rightarrow \Lambda$ op te zoeken. De output van TABEL II wordt dan verenigd (OR-operatie) met de inhoud van ACCU en het resultaat ervan dient als nieuwe waarde voor de ACCU. De scanner en het tweede statement van de completer worden op soortgelijke wijze uitgerekend. Voor de scanner wordt de inhoud van de horizontale "fast belt", die op tijdstip $2(j-1)$ in cel $c_{i,j}$ het eindresultaat van $c_{i,j-1}$ bevat, verenigd (AND-operatie) met de inhoud van het invoerregister INP1 om zo de $B \rightarrow \alpha \cdot a\beta\gamma$ van $t_{i,j-1}$, $a = a_j$, te verkrijgen. Het verkregen binaire woord wordt aangeboden aan TABEL II en .. etc. Het tweede statement van de completer is bijna hetzelfde als het eerste statement van de completer, alleen in plaats van TABEL I wordt er gebruikt gemaakt van TABEL III en de data komt van de ACCU en een register Tii (dat PREDICT(N) bewaard). Het horizontale controlesignaal (HCTL) zorgt ervoor dat de scanner en het tweede statement van de completer op het goede moment geactiveerd worden. Het moet op tijdstip $2(j-i)$ in cel $c_{i,j}$ arriveren. Dit is dus 1 tijdseenheid eerder dan in de VLSI-versie van algoritme 1 het geval is. De cel moet dan 1 tijdstip later (tijd $2(j-i)$) voor $c_{i,j}$ zijn eindresultaat (in ACCU) transporteren naar de "fast belts". Dit wordt gedaan door de datatransfer-unit DT1. Deze DT1 wordt geactiveerd door het HCTL signaal dat in de cel via een tweetraps FIFO buffer 1 tijdseenheid vertraagd wordt. Dus op tijdstip $2(j-i)$ arriveert HCTL in cel $c_{i,j}$ en wordt ook in de 1e trap van de buffer geklokt. Eén tijdstip later wordt de informatie in de buffer naar de 2e trap, en daarmee de uitgang, geklokt en activeert dan de DT1. Hiermee kan de cel dus onthouden dat 1 tijdseenheid geleden de HCTL geweest is. Het tweede statement van de completer moet uitgevoerd worden nadat de scanner en het eerste statement van de completer uitgevoerd zijn. Om deze acties te synchroniseren is de ACCU uitgevoerd met een "ready" signaal (RDY). Dit signaal vertelt of de ACCU zijn inhoud heeft gekregen. Als de ACCU het resultaat van de scanner en het eerste statement van de completer-acties bevat, dan wordt de RDY actief en activeert het tweede statement van de completer.
Het dataflow-patroon vereist dat het resultaat van $c_{i,j}$ op tijdstip $3(j-i)$ van de "fast belt" vertraagd wordt naar de "slow belt". Dit wordt gedaan net als bij de VLSI-versie van algoritme 1, namelijk door de datatransfer-module DT2 en gecontroleerd door het verticale controlesignaal (VCTL). Dit controlesignaal heeft de snelheid van 2 cellen per 3 tijdseenheden en arriveert in $c_{i,j}$ op tijdstip $3(j-i)/2$, met $j-i$ een veelvoud van 2. In figuur 4 zijn ook enkele poorten te zien. Ze hebben een uitgang, twee ingangen en een controlesignaal. Ze zijn getekend als het controlesignaal niet actief is. Is dit wel het geval dan wordt de uitgang met de andere ingang verbonden. De tabellen zorgen ervoor dat de berekeningen in constante tijd plaatsvinden. Met dit ontwerp is het dus mogelijk om algoritme 3 parallel te berekenen in tijd $O(n)$.

3.2.4. Preprocessing

Het berekenen van de tabellen moet gebeuren voordat de herkennings matrix berekend wordt. Hetzelfde geldt voor de berekening van PREDICT(N) en PREDICT($\{S\}$). Meestal zal dit gedaan worden door een host computer, waarvan de chip onderdeel uitmaakt. Deze computer moet ook ervoor zorgen dat tabellen en maskers in een cel geïnitialiseerd worden. Dan wordt de te herkennen zin aan de chip aangeboden en de berekening gestart.

3.3. Simulatie

Het simulatieprogramma voor deze VLSI-algoritme is geschreven in de programmeertaal SUMMER, zie Klint [6]. Het programma is ruwweg verdeelt in drie delen, namelijk a) een deel dat de preprocessing doet, b) een deel dat de input en output verzorgt en c) een deel dat de simulatie kontroleert.

Ad a) De preprocessing procedures zijn verzameld onder de namen:

NULL SET	Deze procedures berekenen de verzameling $\{A \mid A \in N \wedge A \stackrel{*}{\Rightarrow} \Lambda\}$.
CHAIN SET	Deze procedures berekenen voor een nonterminal A de bijbehorende verzameling $\{C \mid C \in N \wedge C \stackrel{*}{\Rightarrow} A\}$.
PREDICT SET	Deze procedures berekenen PREDICT(N) en PREDICT($\{S\}$).
CODING SET	Deze procedures codeert de verzameling van, of tabellen van dotted rules als binaire woorden.
DOTTING SET	Deze procedures berekenen de verzameling van dotted rules en zetten verzamelingen en tabellen van produktieregels om naar verzamelingen en tabellen van dotted rules. Hier worden de tabellen voor de cel berekend.
PREPROCESSING	Deze procedure roept de vorige procedures in de juiste volgorde aan.

Ad b) De I/O-procedures zijn verzameld onder de namen:

INPUT	Leest de grammatica in.
OUTPUT	Drukt de resultaten af.

Ad c) De simulatieprocedures zijn verzameld onder de namen:

SIMULATIE	Deze procedure simuleert het dataflow-patroon en zorgt dat de cellen in de juiste volgorde geactiveerd worden. Om voor de hand liggende redenen wordt de simulatie sequentieël uitgevoerd.
EARLEYSIM.sm	Deze procedure kontroleert de simulatie.

Dan blijft DATATYPES over. Dat zijn de gebruikte class-definities.

3.4. Vergelijking tussen de verschillende VLSI-algoritmen

Chiang en Fu [2] hebben ook een VLSI-ontwerp voor algoritme 3. De overeenkomsten tussen beide VLSI-algoritmen zijn de VLSI-structuur, het dataflow-patroon en de implementatie van de dataflow. Verschillen zijn er in de vorm van de grammatica, de data-representatie en inherent daaraan het ontwerp van een cel. Chiang en Fu hebben hun VLSI-algoritme beperkt tot context-vrije grammatica's zonder de lege produktie. Dit is geen principiële beperking, maar wel een nadeel in de praktijk (zie de inleiding tot §3). Dit doen ze om er zeker van te zijn dat de berekeningen in een cel in een constante tijd gebeuren. De dot hoeft dan elke keer slechts één plaats naar rechts verschoven te worden. Ik heb dat opgelost door van te voren (preprocessing) de dot in de dotted rules op te schuiven en deze in een tabel op te slaan. Voorwaarde is natuurlijk wel dat deze tabel groot genoeg is. Daarom accepteert VLSI-algoritme 2 alle context-vrije grammaticas zonder beperking.

Het aantal bits, dat tussen de cellen getransporteerd wordt, is voor beide VLSI-versies van Earley's algoritme even groot. Bovendien is dit aantal ongeveer even groot als bij de VLSI-versie van het Cocke-Kasami-Younger algoritme, want het aantal produktieregels r in de Chomsky-normaalvorm van een context-vrije grammatica is ongeveer gelijk aan het aantal dotted rules s in de oorspronkelijke grammatica. De VLSI-versie van Chiang en Fu is het zuinigst wat betreft de geheugenruimte per cel. Deze heeft per cel $O(s)$ bits nodig met s het aantal dotted rules. VLSI-algoritme 2 gebruikt per cel 4 s^2-bits tabellen, terwijl de VLSI-algoritme van Chu en Fu [1] een r^3-bits tabel benodigd en r ongeveer evengroot als s. Dit is echter eveneens reduceerbaar tot 4 r^2 (zie §2.1.4). Met de keuze van een andere data-representatie is het ontwerp in een cel natuurlijk anders. Chiang en Fu hebben in woorden geschreven wat een cel moet doen en ook een logisch ontwerp van een cel gepresenteerd. Deze is echter helaas onduidelijk.

4. CONCLUSIES

De 3 VLSI-algoritmen kunnen een willekeurige context-vrije taal herkennen in een tijd lineair in de lengte van de te herkennen zin. Echter, voor 2 van deze algoritmen moet de grammatica herschreven worden. De VLSI-versie van het Cocke-Kasami-Younger algoritme vereist een grammatica in Chomsky-normaalvorm en Chiang en Fu's versie van het Earley algoritme vereist een grammatica zonder lege produkties. VLSI-algoritme 2 kan alle context-vrije grammatica's aan. In principe zijn de herschrijvingen geen beperking omdat elke context-vrije grammatica in de vereiste vorm herschreven kan worden, maar het feit dat de afleidingsboom en de foutmeldingen in termen van de herschreven grammatica worden geproduceerd kan een nadeel zijn. In dat opzicht is VLSI-algoritme 2 dus het beste.

Omdat niet alle cellen gedurende het hele herkenningsproces actief zijn, is het mogelijk in alle 3 gevallen de chip te "pipelinen", dat wil zeggen reeds een nieuwe zin ter herkenning aan te bieden terwijl de vorige nog niet klaar is.

REFERENTIES

[1] Chu, K.H., Fu, K.S., "VLSI architectures for high speed regcognition of context-free languages and finite-state languages", *Proc. Ninth Annual Symp. on Computer Architecture, SIGARCH Newsletter,* **10** (1982), 3, pp. 43-49.

[2] Chiang, Y.T., Fu, K.S., "A VLSI architecture for fast context-free language regcognition (Earley's algorithm)", *Proc. Third Int. Conf. on Distributed Comp. Systems,* 1982, pp. 864-869.

[3] Guibas, L.J., Kung, H.T., Thompson, C.D., "Direct VLSI implementation of combinatorial algorithms", *Proc. Conf. on VLSI, Caltech, Jan.* 1979., pp. 509-526.

[4] Harrison, M.A., *Introduction to formal language theory* , Addison-Wesley, 1978, § 4.3, § 12.4, § 12.6.

[5] Graham, S.L., Harrison, M.A., *Parsing of general context-free languages* , Advances in Computers, vol.14, 1976, pp. 107-115, 122-139.

[6] Klint, P., *From SPRING to SUMMER* , Dissertatie, 1982, Mathematisch Centrum, Amsterdam.

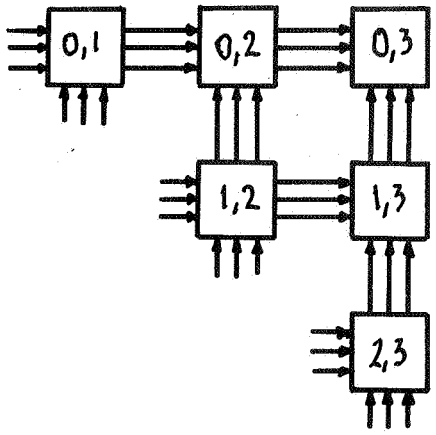

Fig.1 Structuur van een chip

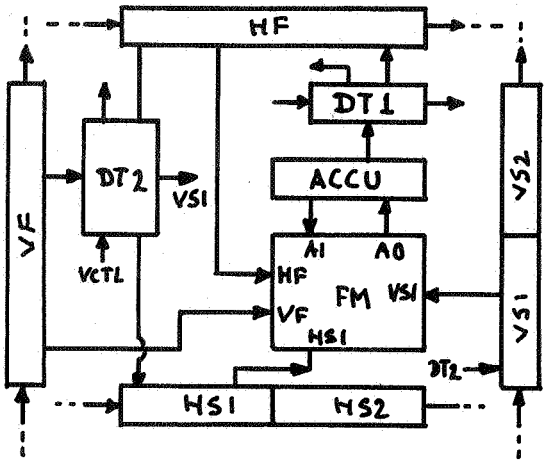

Fig.2 Ontwerp van een cel

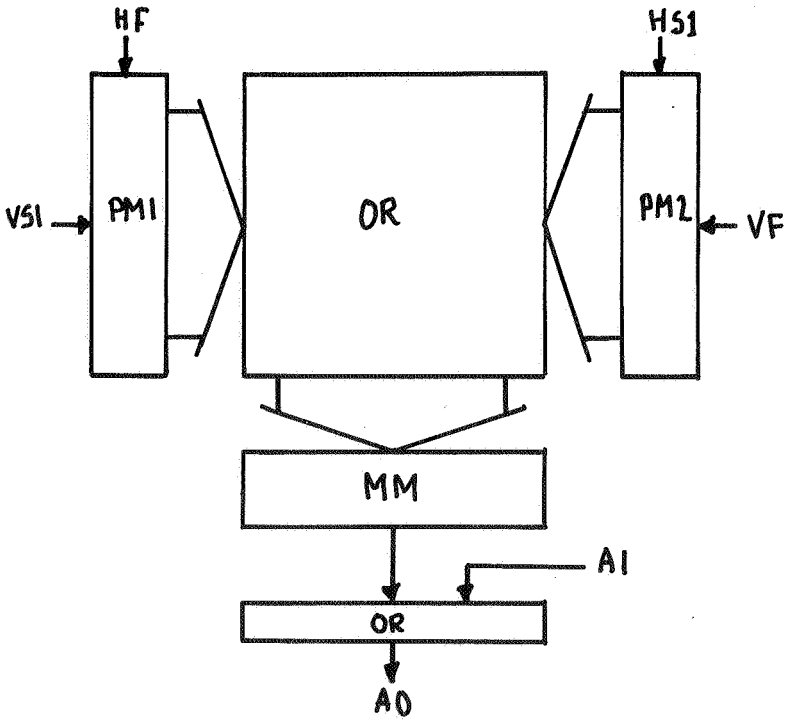

Fig.3 Functiemodule (FM)

Deze figuren zijn overgenomen van Chu en Fu [1].

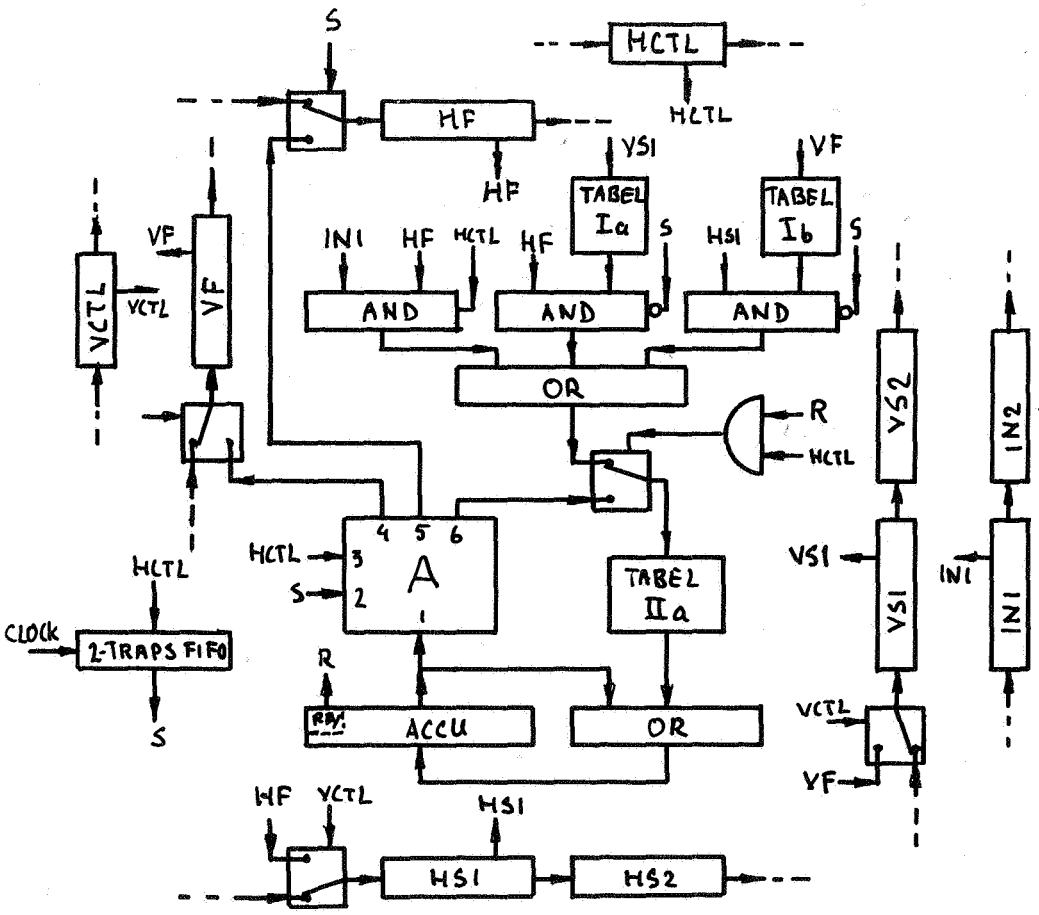

Fig.4 Ontwerp van een cel

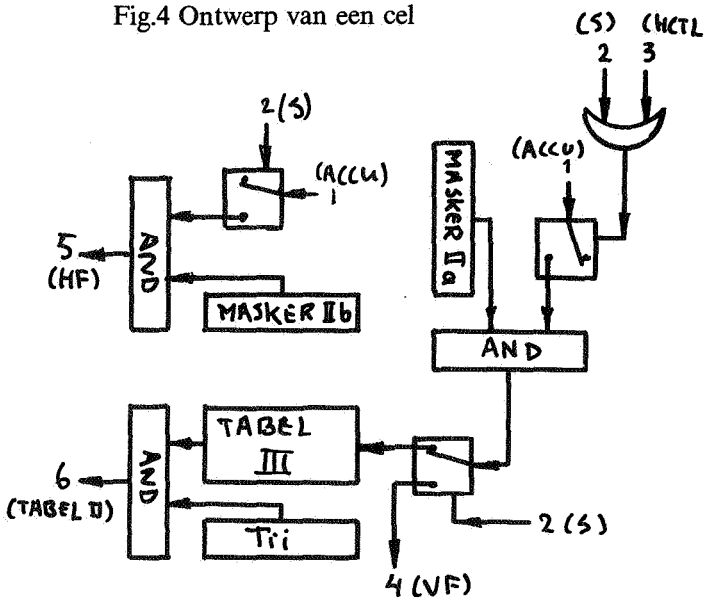

Fig.4a Ontwerp van A in fig.4