Een methode en een programma voor grammatische inferentie.

door: Leo Konst, nov. 1977.

1. Inleiding:

Een van de meest interessante problemen die o.a. in psycholinguistisch onderzoek opdoemen is het probleem van de taalverwerving.
In zijn meest algemene vorm kan dit probleem als volgt vertolkt worden:
Hoe construeert een organisme (of machine) van een bepaalde structuur en gegeven een bepaalde vorm van taalaanbod uit de omgeving een eindige specificatie voor een eindige hoeveelheid taal, die dat organisme aangeboden krijgt.
Vertalen we dit probleem in meer algemene termen, dan luidt de definitie van dit grammatische inferentie probleem:
Gegeven een eindige steekproef van zinnen $S=\{s_1 \ldots\ldots s_n\}$ uit een mogelijk oneindige taal L ($S \subseteq L$), construeer een grammatica G, zodanig dat $L(G)=L$.
Vanuit deze definitie is het mogelijk het grammatische inferentie probleem op verscheidene manieren te beschouwen:

1. Het G.I. probleem in de compiler constructie voor programmeertalen;
2. Het G.I. probleem in de constructie van een systeem dat grammatica's voor natuurlijke talen afleidt;
3. Het G.I. probleem als model voor het taalverwervingsproces bij mensen;
4. Het G.I. probleem als onderdeel van het algemene inferentie (inductie)-probleem.

De belangrijkste variabelen die een rol spelen bij de beantwoording van het G.I. probleem zijn:

1. de hypotheseruimte.

Hieronder zullen we verstaan de hoeveelheid informatie over de eigenschappen van de gewenste grammatica, die van te voren al bekend is (vb. het type grammatica dat gezocht wordt).

2. de observatieruimte.

De observatieruimte kunnen we definiëren als de verzameling van eigenschappen van de aangeboden steekproef. We onderscheiden de volgende soorten steekproeven of informatierijen:

a) een positieve informatierij.
   De steekproef bevat alleen elementen van L. D.w.z. we krijgen alleen positieve informatie over L.
b) een gemengde informatierij.
   De steekproef bevat informatie over elementen die tot L behoren en geeft informatie over elementen die niet tot L behoren.
c) een complete/incomplete informatierij.
   Indien S=L spreken we van een complete positieve informatierij, ook wel tekstpresentatie genoemd. Indien $S \subset L$ spreken we van een incomplete positieve informatierij.

3. de acceptabiliteit van de gewenste grammatica.

De acceptabiliteit van een grammatica is een tot nu toe vaag gedefiniëerde variabele,die een functie is van de volgende eigenschappen van de grammatica:

a) de mate waarin $S \subset L(G)$
b) de complexiteit van de grammatica,die op haar beurt weer afhangt van de aarde van de produktieregels van de grammatica.

Voor een uitgebreidere behandeling verwijzen we naar Levelt (1973), die een goed overzicht geeft van de op dat moment bekende resultate van het onderzoek naar het G.I. probleem.

We zullen het komende stuk alsvolgt indelen:
Eerst geven we een eenvoudig algoritme,dat voor een contextvrije taal een grammatica construeert met slechts één nonterminaal symbool. Vervolgens breiden we dit uit voor grammatica's met meer dan een nonterminaal symbool. Het is de bedoeling dat het algoritme een contextvrije taal kan leren van een instructeur, in die zin,dat aan het eind van de instructiefase het programma

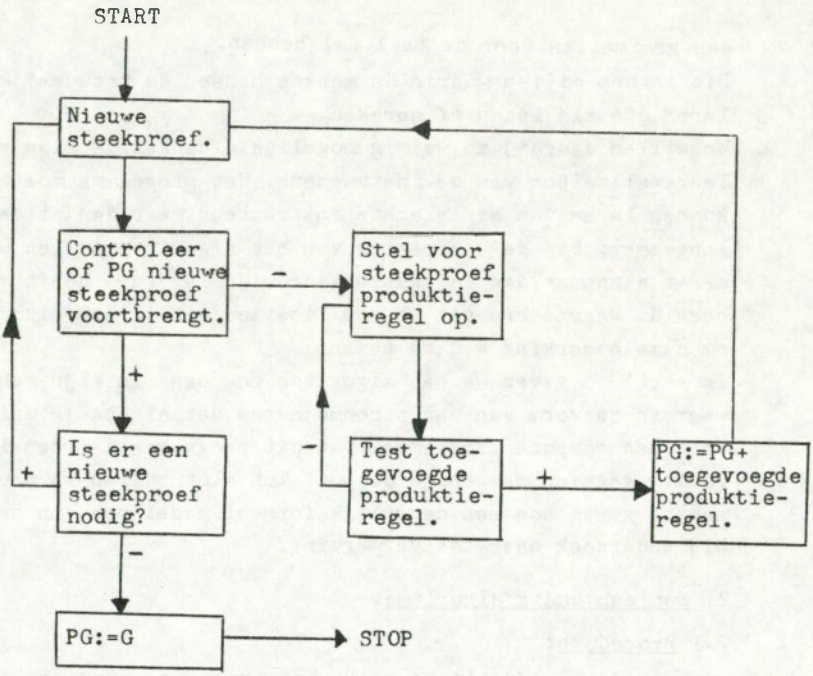
een grammatica voor de taal zal hebben.
Dit is het milieu waarin de meeste mensen de natuurlijke talen leren die zij lezen of spreken.
We willen daarbij zo weinig mogelijk afhankelijk zijn van de leerkwaliteiten van de instructeur. Het programma moet ook kunnen leren van een slechte instructeur,maar dan alleen wat langzamer. Bij de bespreking van het algoritme zullen we eerst aannemen,dat de instructeur zijn les goed heeft voorbereid. Daarna brengen we modificaties aan in het algoritme om deze beperking weg te nemen.
In sectie 5 geven we het algoritme nog eens in zijn geheel weer in de vorm van een stroomdiagram,dat als basis zal dienen voor een computerprogramma. Van dit programma zal een listing worden gegeven met enige output. Tot slot zullen we proberen aan te geven hoe een dergelijk formeel model ons kan helpen bij onderzoek naar taalverwerving.

2. Een eenvoudig algoritme:

2.1 Procedure:

Op ieder tijdstip beschikt het programma over een partiële grammatica,PG,voor de taal L. De instructeur geeft het programma een nieuwe steekproef $s_i \in L_{pg}$. Het programma checkt of s_i reeds door PG kan worden voortgebracht. Indien s_i kan worden voortgebracht overlegt het programma met de instructeur of een nieuwe steekproef s_{i+1} nodig is. Indien s_i niet kan worden voortgebracht voegt het programma aan de bestaande grammatica PG een produktieregel toe,die o.a. deze zin toevoegt aan de taal die PG kan genereren.
Het programma controleert of de toegevoegde produktieregel niet tot ongewenste (illegale) zinnen leidt en vraagt daarna aan de instructeur of een nieuwe steekproefzin nodig is.

In een blokdiagram weergegeven:



2.2 Het opstellen van nieuwe produktieregels:

Hoe zorgen we er nu voor,dat het programma op de meest economische manier een produktieregel aan de bestaande PG toevoegt?
Als we de grammatica compact willen houden en de instructiefase zo kort mogelijk,dan zal het programma de meest algemene regel toe moeten voegen.

Op deze manier wordt het mogelijk gemaakt een veel grotere klasse van zinnen toe te voegen aan de taal die reeds door PG gegenereerd kan worden,dan de enkele steekproef die aangeboden is. Tevens is de kans dan het grootst,dat die nieuwe grammatica PG'=PG+meest-algemene-prod.regel gefalsifieerd wordt als mogelijke kandidaat voor G.
Om de meest algemene correcte produktieregel te vinden, construeren we een lijst met kandidaat-produktieregels. Deze kandidaat-produktieregels worden gevormd door gedeelten uit de steekproefzin te vervangen door nonterminale symbolen uit de bestaande PG. Wanneer dit lukt,ontstaan "partiële parsings".

Voorbeeld 1: Stel dat we het programma de taal leren van expressies over het alfabet: I,+,*,(,).
Als de bestaande PG is:

<expr> ::= I
<expr> ::= <expr> + <expr>

en de nieuwe steekproefzin is:

I * I

dan verkrijgen we de volgende kandidaatprodukties:

(1) <expr> ::= <expr> * <expr>
(2) <expr> ::= <expr> * I
(3) <expr> ::= I * <expr>
(4) <expr> ::= I * I

Bij het definiëren van de relatieve algemeenheid van de produkties beperken we ons even tot de rechter-zijden van de regels,omdat binnen het huidige kader de linker-zijde van iedere kandidaat hetzelfde is (i.c. <expr>).
We kunnen nu de relatieve algemeenheid van produktieregels alsvolgt definiëren:

Een produktieregel $\alpha ::= \beta$ wordt algemener genoemd dan een regel $\delta ::= \gamma$ indien:

a) $|\beta| < |\gamma|$

b) de ratio van nonterminale/terminale symbolen in β groter is dan in γ.

c) $\alpha ::= \beta$ is een recursieve regel en $\delta ::= \gamma$ niet.

Volgens deze criteria is de eerste van de kandidaatprodukties de meest algemene. De regel <expr> ::= <expr> * <expr> wordt daarom toegevoegd aan PG.

Voorbeeld 2: Leer het programma de taal,die bestaat uit de verzameling welgevormde haakjesuitdrukkingen.
Wanneer we uitgaan van een PG die in het begin leeg is en we geven als eerste steekproefzin de haakjesuitdrukking ( ), dan verkrijgen we een PG met produktieregel:

<root> ::= ( )

Wanneer we als tweede steekproefzin de haakjesuitdrukking ( )( ) aanbieden,dan worden de volgende kandidaatregels opgesteld:

(1) <root> ::= <root> <root>
(2) <root> ::= <root> ( )
(3) <root> ::= ( ) <root>
(4) <root> ::= ( ) ( )

De eerste regel is voor alle kriteria de meest algemene regel en produceert geen illegale zinnen,zodat deze regel aan PG wordt toegevoegd. PG wordt dan:

<root> ::= ( )
<root> ::= <root> <root>

De haakjesuitdrukking (( )) als derde steekproefzin produceert tenslotte de kandidaatregels:

(1) <root> ::= (<root>)
(2) <root> ::= (( ))

De eerste regel is voor alle kriteria weer de meest algemene en produceert geen illegale zinnen,zodat PG nu wordt:

<root> ::= ( )
<root> ::= <root> <root>
<root> ::= (<root>)

Deze grammatica is nu een komplete grammatica voor de taal,die bestaat uit de verzameling welgevormde haakjesuitdrukkingen. Elke verdere steekproef is overbodig omdat er geen extra regels door worden toegevoegd. Merk op dat dit triviale voorbeeld een taal specificeert die niet regulier is.

3. Automatische introduktie van nonterminale symbolen:

De methode,die we tot dusver hebben beschreven,is in staat om grammatica's te construeren voor redelijk complexe talen, maar loopt vast bij sommige triviale reguliere talen,zoals de taal die bestaat uit willekeurig lange rijtjes van A's en willekeurig lange rijtjes van B's.
Dit komt omdat deze taal niet gegenereerd kan worden door een grammatica met slechts één nonterminaal symbool.
We zullen eerst zien hoe een PG zich ontwikkelt met behulp van de bestaande methode en daarna de nodige verbeteringen aanbrengen.

Voorbeeld 3: Leer het programma de taal die bestaat uit willekeurig lange rijtjes A's en willekeurig lange rijtjes B's.

Steekproefzin 1: A levert op:

<root> ::= A

Steekproefzin 2: B levert op:

<root> ::= B

Steekproefzin 3: AA levert de volgende kandidaatregels op:

(1) <root> ::= <root><root>
(2) <root> ::= <root> A
(3) <root> ::= A <root>
(4) <root> ::= AA

De eerste drie regels leveren ook illegale zinnen op,nl. AB,BA. Slechts regel (4) is toegestaan,maar op deze manier zal nooit een grammatica voor de taal kunnen worden ontwikkeld,omdat voor iedere A^n of B^n een corresponderende regel van de vorm

<root> ::= A^n
<root> ::= B^n of

zal moeten worden toegevoegd.

We construeren nu een algoritme dat deze moeilijkheid oplost en de methode voor het construeren van een grammatica in aanzienlijke mate versnelt.
Het algoritme bouwt voort op wat we al hebben ontwikkeld.
Het algoritme bestaat uit twee onderdelen, A1 en A2.
A1 zorgt ervoor dat automatisch nieuwe nonterminale symbolen worden geintroduceerd.
A2 is een methode om subrijtjes van produktieregels als mogelijke voorbeelden van dezelfde grammaticale constructie op te vatten. Als dit lukt,wordt een van de produktieregels vereenvoudigd.

3.1 Beschrijving van algoritme A1:

Om te beginnen is het ongemakkelijk om met produkties te werken,die terminale symbolen bevatten. Als een nieuwe produktieregel verschillende terminale symbolen $A_1,A_2,\ldots,A_n$ bevat, introduceren we daarom de hulpprodukties:

$\langle a_1 \rangle ::= A_1$
$\langle a_2 \rangle ::= A_2$
$\vdots$
$\langle a_n \rangle ::= A_n$

3.2 Beschrijving van algoritme A2:

Om subrijtjes te ontdekken in de bestaande produktieregels, die waarschijnlijk voorbeelden zijn van dezelfde grammaticale constructie,zullen we eerst het begrip "overeenkomst" tussen twee produktieregels definiëren.

We noemen twee produktieregels overeenkomstig indien:

(1) zij zijn te schrijven als
$X::=PQ_1R$ en
$X::=PQ_2R$ waarin
P,Q_1,Q_2 en R rijtjes zijn van nonterminale symbolen.
(2) of P of R of beide zijn niet-leeg. We zullen dit de geidentificeerde symbolen noemen.
(3) Q_1 en Q_2 zijn niet-leeg. We zullen dit de overeenkomstige rijtjes noemen.

We construeren het algoritme A_2 nu zodanig,dat het de paren van overeenkomstige rijtjes tussen de geidentificeerde symbolen voorlopig opvat als voorbeelden van éénzelfde grammaticale constructie. Daarom zal het algoritme dan ook trachten ieder van de overeenkomstige rijtjes te vervangen door hetzelfde nonterminale symbool. We onderscheiden twee voorkomende gevallen:

geval 1: Ieder van de twee overeenkomstige rijtjes bestaat uit een enkel nonterminaal symbool,zeg S en T. In dit geval kunnen we iedere T in de grammatica eenvoudig vervangen door S en de veranderde regels testen. Als de veranderde regels illegale zinnen produceren,maken we de verandering ongedaan. Zo niet,dan maken we de verandering permanent.

geval 2: Een van de twee overeenkomstige rijtjes is een enkel nonterminaal symbool S en de ander is een rij nonterminale symbolen $T_1,T_2,\ldots,T_m$.
In dit geval vervangen we de regel die de T's bevat door de regel $S::=T_1 \ldots T_m$.
Nadat deze verandering is gecheckt,kunnen we proberen andere gevallen van de rij $T_1 \ldots T_m$ te vervangen door S.

Over het algemeen kan een nieuwe produktieregel overeenkomen met verschillende andere produktieregels en bovendien kan er op verschillende manieren overeenkomst tussen produktieregels bestaan. Volgens ons basis-uitgangspunt zullen we in zulke gevallen weer de meest algemene mogelijkheid kiezen.

Voorbeeld 4: Met behulp van de algoritmes A1 en A2 zullen we nu nogmaals de taal proberen te specificeren,die bestaat uit willekeurig lange rijtjes A's en willekeurig lange rijtjes B's.

Steekproefzin 1: A levert op:

(1) <root> ::= $<a_1>$
(2) $<a_1>$ ::= A

Steekproefzin 2: B levert op:

(3) <root> ::= $<b_1>$
(4) $<b_1>$ ::= B

Steekproefzin 3: A A levert de volgende kandidaatregels op:

<root> ::= <root> <root>
<root> ::= <root> $<a_1>$
<root> ::= $<a_1>$ <root>
<root> ::= $<a_1>$ $<a_1>$
<root> ::= A A

De meest algemene produktieregel,die geen illegale zinnen genereert is:

(5) <root> ::= $<a_1>$ $<a_1>$

Steekproefzin 4: B B levert op:

(6) <root> ::= $<b_1>$ $<b_1>$

Steekproefzin 5: A A A zou nu toevoegen de regel:

(7') <root> ::= $<a_1>$ $<a_1>$ $<a_1>$

Maar er is een overeenkomst tussen regel (5) en (7'),wanneer we stellen dat:
$P=<a_1>$; $Q_1=<a_1>$; $Q_2=<a_1><a_1>$;R=leeg rijtje.
We kunnen nu geval-2-transformatie toepassen,nl:

(7) $<a_1>$ ::= $<a_1>$ $<a_1>$

Deze regel produceert geen illegale zinnen en vervangt daarom (7').

Steekproefzin 6: B B B levert via toepassing van dezelfde soort transformatie op:

(8) $<b_1>$ ::= $<b_1>$ $<b_1>$

Met de regels (1) t/m (8) hebben we nu de voltallige grammatica, die precies de gewenste taal zal genereren.

4. Reductie van de afhankelijkheid van aanbiedingsvolgorde:

Zoals ons algoritme er nu voor staat, werkt het redelijk goed als de steekproefzinnen in optimale volgorde worden aangeboden. Maar het algoritme is zeer afhankelijk van het ontdekken van deze volgorde door de instructeur.
"Onverstandige" instructeurs kunnen ervoor zorgen, dat de geconstrueerde grammatica alleen zinnen genereert voorzover ze in de steekproef gegeven zijn en geen andere.
We zullen dit illustreren aan de hand van het volgende voorbeeld:

Voorbeeld 5: Stel dat de instructeur uit voorbeeld 1 de volgende "onverstandig" gekozen steekproefvolgorde aanbiedt:

I + I, I * I, (I) en als laatste I.

Het programma zou dan met de volgende grammatica uit de bus komen:

<expr> ::= I + I
<expr> ::= I * I
<expr> ::= (I)
<expr> ::= I

d.w.z. een grammatica die alleen de steekproefzinnen genereert. De instructeur heeft n.l. verzuimd te beginnen met het rijtje I, waardoor het programma niet in de gelegenheid is gesteld, de grammatica op de juiste manier op te stellen.
Om de opbouw van de grammatica niet al te afhankelijk te laten zijn van een slechte instructeur, voeren we het volgende correctie-algoritme A3 in:

4.1 Beschrijving van algoritme A3:

Stel onze PG bestaat uit een aantal produktieregels:

(1) A ::= x_1 ... x_n
(2) B ::= b_1 ... b_m
⋮
(n) .....

en op grond van een nieuwe steekproef wordt een nieuwe produktieregel D ::= d_1 ... d_p toegevoegd.
Alle reeds bestaande regels 1,2,...n worden nu onderzocht op de mogelijkheid deze regels te parsen met behulp van D ::= d_1 ... d_p op de volgende wijze:

Stel we onderzoeken B ::= b_1 ... b_m.
Genereer nu alle partiële parsingen,waarin D::=d_1 ... d_p een rol speelt.
Vervang dan B::=b_1 ...b_m door B::= ... D ... (partiële parsing) en voer tests uit om illegale produkties te vermijden.

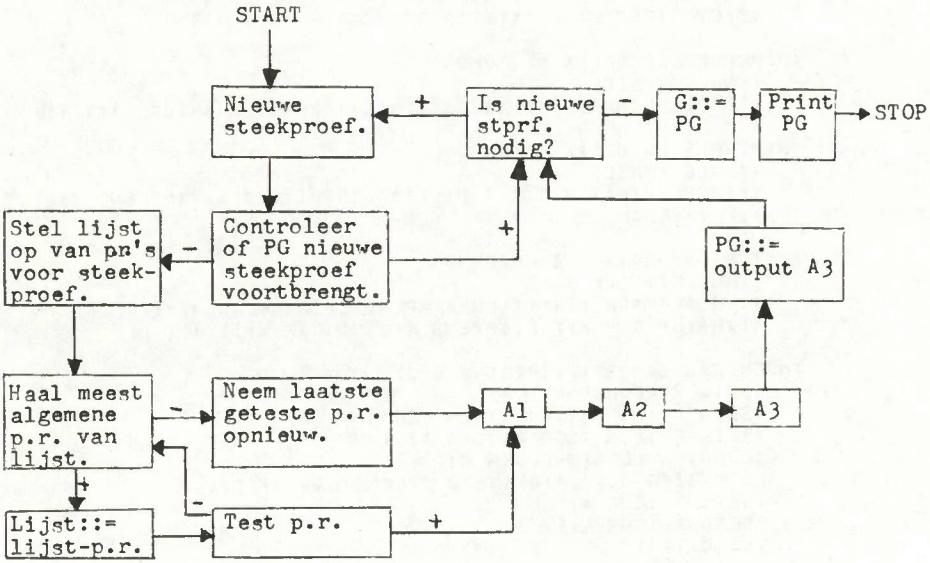
Voorbeeld 6: Zij gegeven een partiële grammatica:
(1) <S> ::= ( ) ( )
(2) <S> ::= (( ))
en de nieuwe produktieregel:
(3) <S> ::= ( )
Door met behulp van de nieuwe regel (3) de oude regels (1) en (2) te parsen vinden we:
(1) <S> ::= S S
(2) <S> ::= (S)
(3) <S> ::= ( )

5. Computerprogramma:

We hebben nu een programma geconstrueerd,dat op basis van een eindig aantal steekproefzinnen een contextvrije grammatica voor de taal,waaruit de steekproefzinnen genomen zijn,opbouwt. De methode die we voor de constructie van de contextvrije

grammatica gehanteerd hebben,is aangevuld met enige correctie-algoritmen,voldoende om op een redelijk efficiënte manier en tamelijk onafhankelijk van de volgorde van de aangeboden informatie,gebruik te maken van de steekproefzinnen.

In een stroomdiagram ziet de methode,aangevuld met correctie-algoritmen er alsvolgt uit:



We hebben bovenstaand stroomdiagram omgezet in een computer-programma,geschreven in de programmeertaal LISP 1.5 .
Deze taal is ontworpen door McCarthy (1962) en is daarna veel gebruikt op gebieden van psychologie,linguistiek en kunstmatige intelligentie.
Het programma is geschreven voor de U.T.-Lispcompiler en draait op een CYBER 73/26 computer. We zullen eerst een listing van het programma geven,om daarna enige outputresultaten te bespreken.

5.1 Programma-listing:

```
*EVAL!
? GRAMMATISCHE INFERENTIE ?
(SETQ //MODE #(EVALQUOTE . 2))
VALUE!
(EVALQUOTE . 2)
*EVALQUOTE!
DEFINE((

(PARSE(LAMBDA(X Y)(COND
   ((NULL X)NIL)
   ((ATOM(CAR X))(APPEND(LIST X)(MAPRULES X Y)))
   (T(CONS(PARSE(CAR X)Y)(PARSE(CDR X)Y))) )))

(MAPRULES(LAMBDA(X Y)(COND
   ((NULL Y)NIL)
   (T(APPEND(MAPINPUT X(CAR Y)NIL)(MAPRULES X(CDR Y))) ))) )

(MAPINPUT(LAMBDA(X Y Z)(COND
   ((NULL X)NIL)
   (T(APPEND(MATCH X Y Z NIL)(MAPINPUT(CDR X)Y(APPEND Z(LIST(CAR X)
   ))) )))

(MATCH(LAMBDA(X Y Z W)(COND
   ((NULL X)NIL)
   ((EQUAL(CADR Y)W)(PARSE(APPEND Z(CONS(CAR Y)X)) PG))
   (T(MATCH(CDR X)Y Z(APPEND W(LIST(CAR X))) ))) ))

(MGENERAL(LAMBDA(X)(PROG(A B C)
   (SETQ X(REDUCE X))
   (SETQ A (MAPCAR(FUNCTION LENGTH) X))
   (SETQ B (MIN (CDR A)(CAR A)))
  Q(COND((NULL A)(RETURN C))
        ((EQN(CAR A)B)(SETQ C(CONS(CAR X)C))) )
   (SETQ X (CDR X))
   (SETQ A (CDR A))
   (GO Q))) )

(REDUCE(LAMBDA(X)(COND
   ((NULL X)NIL)
   ((MEMBER(CAR X)(CDR X))(REDUCE(CDR X)))
   (T(CONS(CAR X)(REDUCE(CDR X))) ))) )

(MAPCAR(LAMBDA(FNEXPR X)(COND
   ((NULL X)NIL)
   (T(CONS(FNEXPR(CAR X))(MAPCAR FNEXPR(CDR X))) ))) )
```

```
(MIN(LAMBDA(X Y)(COND
   ((NULL X)Y)
   ((LESSP(CAR X)Y)(MIN(CDR X)(CAR X)))
   (T(MIN(CDR X)Y))) ))

(READINP(LAMBDA(X Y)(COND
   ((NULL X)(READINP(STARTREAD)Y))
   ((EQ X PERIOD)NIL)
   ((OR(EQ X BLANK)(EQ X EOR))(COND((NULL Y)(READINP(ADVANCE)Y))
          (T(CONS Y(READINP(ADVANCE)NIL))) ))
   (T(READINP(ADVANCE)(APPEND Y(LIST X))) )))

(MAPCAR1(LAMBDA(FNEXPR X)(COND
   ((NULL X)NIL)
   (T(APPEND(FNEXPR(CAR X))(MAPCAR1 FNEXPR(CDR X))) ))) )

(LEARN(LAMBDA(X)(PROG(A B C PR PG)
   (SETQ PG (GET #PG #APVAL))
   (SETQ A (LIST(EXPLODE X)))
 Q1 (COND((NULL A)(RETURN(BNF PG))))
   (SETQ PR(CONS(QUOTE S)(MGENERAL(PARSE(CAR A)PG))))
   (SETQ A (CDR A))
   (SETQ B (MAPCAR(FUNCTION CADR)PG))
   (PUT #PG #APVAL (LIST PR))
   (SETQ PG (GET #PG #APVAL))
   (SETQ PR (MAPCAR1 (FUNCTION MGENERAL)(PARSE B PG)))
Q2 (COND ((NULL PR)(GO Q1)))
   (SETQ C (CAR(MGENERAL PR)))
   (PUT #PG #APVAL (CONS(LIST #S C)PG))
   (SETQ PG (GET #PG #APVAL))
   (SETQ PR (EFFACE C PR))
   (GO Q2))))

(BNF(LAMBDA(X)(PROG(A)(PRIN1 #PG)(PRIN1 BLANK)(PRIN1 #IS)(PRIN1 COLON)
   (TERPRI) Y (COND((NULL X)(RETURN BLANK)))(SETQ A(CAR X))
   (PRIN1 LESS)(PRIN1(CAR A))(PRIN1 GREATER)(PRIN1 BLANK)(PRIN1 COLON)
   (PRIN1 COLON)(PRIN1 EQSIGN)(PRIN1 BLANK)(PRIN1(COMPRESS(CADR A)))
   (TERPRI)(SETQ X (CDR X))(GO Y))))

))


VALUE:
(PARSE MAPRULES MAPINPUT MATCH MGENERAL REDUCE MAPCAR MIN READINP

 MAPCAR1 LEARN BNF)
```

5.2 Output:

Stel we willen de machine de taal leren van zinnen die ieder bestaan uit een rij van A's en/of B's,gevolgd door het spiegelbeeld van die rij. Men noemt deze taal ook wel een spiegelbeeldtaal.
We geven de instructie eerst als een goede instructeur.
Aan het begin zetten we PG op NIL,het lisp-symbool voor de lege lijst.
Vervolgens roepen we het programma aan met de steekproefzin AA. Dit geeft als output een grammatica bestaande uit de regel: <S>::= AA .

```
31  PUT(PG APVAL NIL)
    VALUE:
33  PG
    *EVALQUOTE:
35  LEARN( AA )
    G IS:
37  <S> ::= AA
    *VALUE:
```

Vervolgens geven we de steekproefzin BB ,met als resultaat:

```
39  *EVALQUOTE:
    LEARN( BB )
41  G IS:
    <S> ::= AA
43  <S> ::= BB
    *VALUE:
45
```

De volgende steekproefzin, ABBA , kunnen we parsen met behulp van een reeds bestaande regel. Er wordt nu een recursieve herschrijfregel aan de grammatica toegevoegd.

```
47  *EVALQUOTE:
    LEARN( ABBA )
49  G IS:
    <S> ::= AA
    <S> ::= BB
    <S> ::= ASA
53  *VALUE:
```

De vierde steekproefzin, BAAB ,kunnen we weer parsen met behulp van een reeds bestaande regel. Er wordt weer een recursieve herschrijfregel aan de grammatica toegevoegd.

```
*EVALQUOTE:
LEARN( BAAB )
G IS:
<S> ::= AS A
<S> ::= AA
<S> ::= BB
<S> ::= BSB
*VALUE:
```

De grammatica is nu compleet. Elke verdere steekproef is overbodig,want er zullen geen extra regels door worden toegevoegd. Het zal duidelijk zijn,dat deze grammatica door de aanwezigheid van recursieve herschrijfregels een in principe oneindig aantal spiegelbeeld-zinnen kan genereren.

We zullen de machine dezelfde taal nog eens leren,maar geven de instructie dan als een slechte instructeur.
We beginnen weer met PG op NIL te zetten.
Daarna geven we de steekproefzinnen in deze volgorde: BAAB,ABBA,BB,AA. Bij deze instructie ziet de output er alsvolgt uit:

```
*EVALQUOTE:

PUT(PG APVAL NIL)
VALUE:
PG
*EVALQUOTE:
LEARN( BAAB )
G IS:
<S> ::= BAAB
*VALUE:

*EVALQUOTE:
LEARN( ABBA )
G IS:
<S> ::= BAAB
<S> ::= ABBA
*VALUE:
```

```
*EVALQUOTE:
LEARN( BB )
G IS:
<S> ::= BAAB
<S> ::= ASA
<S> ::= BB
*VALUE:

*EVALQUOTE:
LEARN( AA )
G IS:
<S> ::= BSB
<S> ::= ASA
<S> ::= BB
<S> ::= AA
*VALUE:

*EVALQUOTE:
FIN
```

6. Relatie met de taalverwervingstheorie:

Al in de eerste jaren van het psycholinguistisch onderzoek A.C. (after Chomsky),begon men met het onderzoek naar de verwerving van een grammatica door kinderen. Wanneer we een verband willen leggen tussen het hier ontwikkelde formele model en het conceptuele model van de taalverwervingstheorie dat lange tijd populair is geweest bij zowel taalkundigen als psycholinguisten, het zogenaamde LAD-model,dan moeten we nagaan in hoeverre de assumpties van beide modellen overeenkomen.
De assumpties van het conceptuele LAD-model komen grotendeels overeen met de assumpties,die we voor ons formele model hebben,namelijk:

1. een kind wordt geconfronteerd met een (eindige) hoeveelheid taaluitingen (zinnen over L)
2. op basis van deze taaluitingen moet het kind een grammatica voor een natuurlijke taal L opstellen
3. het kind wordt verondersteld te "weten" welke klasse van grammatica's onderzocht moet worden om een kandidaat voor L te vinden
4. het kind kan alleen gebruik maken van syntactische informatie uit de hem aangeboden zinnen.

Ruwweg karakterizeren deze uitgangspunten het onderzoek naar het Language Acquisition Device (LAD).
Schematisch kan het LAD-model zo voorgesteld worden:

$$\rightarrow S_n\ (x_1 \underset{\subset L}{\ldots} x_n) \rightarrow \text{LAD} \rightarrow G, L(G)=L$$

Het LAD ontvangt dus als input een reeks zinnen uit L en produceert als output een grammatica voor L.

We hebben niet de pretentie het hier ontwikkelde algoritme als een compleet formeel model van het LAD te postuleren, maar slechts een eerste stap in die richting geprobeerd.

De belangrijkste les die we uit dit onderzoek kunnen trekken is echter,dat een formeel model van een reële situatie of conceptueel model zoals hier taalverwerving,ons kan helpen het onderzoek te sturen en/of (radikaal) te wijzigen.
Dit geldt des te sterker als dit formele model uit een grondig onderzochte mathematische theorie stamt,waarin ook relaties gelegd kunnen worden met andere formele systemen.

Nemen we als voorbeeld dit onderzoeksprogramma:

1. Op basis van een aantal uitgangspunten wordt een onderzoek naar taalverwerving opgebouwd.
2. Het bestaan van een taalverwervend mechanisme (LAD) wordt gepostuleerd en een aantal assumpties gedaan over input en output van dit (conceptuele) model.
3. Op basis hiervan is een formeel model van dit conceptuele systeem te construeren.
4. Dit formele model maakt het mogelijk relaties te leggen met verschillende mathematische theorieën zoals de theorie van de formele talen,recursieve functie-theorie,abstracte automaten.
5. Met behulp van het formele model kunnen afgeleide stellingen (theorema's) terugvertaald(geïnterpreteerd) worden in het conceptuele model of systeem; als gevolg hiervan kunnen o.a.
   a) (assumpties van) het conceptuele model veranderd worden en/of b) toekomstige gedragingen van het reële systeem voorspeld worden als aan bepaalde voorwaarden is voldaan. en/of c) de relatie formeel/conceptueel model nader bekeken worden.
6. Omdat we in het LAD onderzoek in de eerste plaats een verklaring willen geven voor het feit,dat een kind een bepaalde natuurlijke taal leert,moeten op grond van de terugvertaalde resultaten uit de formele theorie wel veranderingen worden aangebracht in de assumtpies van het conceptuele model.
   Dat is dan ook het probleem waarvoor het LAD onderzoek zich ziet geplaatst.

Referenties:

McCarthy,J.,LISP 1.5 Programmer's Manual. MIT Press,1962.

Levelt,W.J.M.,Formele grammatica's in linguistiek en taalpsychologie. Van Lochem Slaterus,Deventer,1973.