Determinisme og syntaktisk flertydighet

Torbjørn Nordgård
Universitetet i Bergen

1. Innledning

Man forstår gjerne en *deterministisk* innretning som et redskap som finner raskeste vei til en, og bare en, løsning av en oppgave, eksempelvis analysen av en setning. *Syntaktisk flertydighet*, på den annen side, behandles oftest ved at grammatikken prediserer et *sett* av syntaktiske representasjoner til en ordsekvens. I det som følger skal jeg vise hvordan determinismebegrepet kan modifiseres på en måte som bevarer determinismens fordelaktige sider, samtidig som syntaktiske flertydigheter kan analyseres korrekt.

2. Syntaktisk flertydighet

Den lingvistiske litteraturen inneholder utallige eksempler på syntaktiske flertydigheter. Typisk er analysen av den syntaktiske rekkevidden til preposisjonsfraser, som i (1), men også varianter som i (2) og (3):

(1) Jens vil se mannen med kikkerten (2 lesninger)

a. Jens vil [VP se [NP mannen [PP med kikkerten]]]
"Jens vil se mannen, og mannen har kikkerten"

b. Jens vil [VP se [NP mannen] [PP med kikkerten]]
"Jens vil se mannen, og Jens vil bruke kikkerten"

(2) Hvem elsker Marit? (2 lesninger)

a. Hvem [VP elsker [NP Marit]]
"For hvilke individer x forholder det seg slik at x er elsker av Marit"

b. [CP Hvem $_i$ [C' elsker $_j$ [S Marit [VP ej e_i]]]]^1
"For hvilke individer x forholder det seg slik at Marit er elsker av x"

(3) Jens så mannen med kikkerten (4 lesninger)

a. [S Jens [VP så [NP mannen] [PP med kikkerten]]]
"Mannen ble sett av Jens, og Jens brukte kikkerten"

b. [S Jens [VP så [NP mannen [PP med kikkerten]]]]
"Mannen som hadde kikkerten ble sett av Jens"

c. [CP Jens$_i$ [C'så$_j$ [S [NP mannen [PP med kikkerten]] [VP ej e_i]]]]
"Jens ble sett av mannen som hadde kikkerten"

Proceedings of NODALIDA 1991, pages 153-164

d. [CP Jens$_i$ [C'såj [S [NP mannen] [VP e$_j$ e$_i$ [PP med kikkerten]]]]]
"Jens ble sett av mannen, og mannen brukte kikkerten"

3. Syntaktisk flertydighet og automatisk setningsanalyse

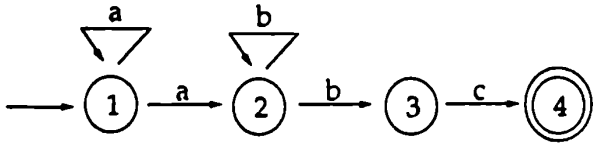
En maskin som foretar setningsanalyse, dvs. en *parser*, bør finne frem til alle syntaktiske lesningene av et gitt input. Merk den noe beskjedne modaliteten; den kommer av at det kan gis argumenter som motiverer parsere som bare finner *en* syntaksrepresentasjon for hvert input. Jeg tenker her på parsere for formelle språk, f.eks. programmeringsspråk, der man aldri tillater noen form for flertydigheter (en datamaskin må ha entydige instruksjoner). Det kan også argumenteres for at parsere for naturlige språk ikke skal returnere mer enn en strukturell analyse. Dette gjelder parsere som bare skal finne den mest plausible lesningen, gitt at den også har tilgang til intonasjonsmessig relevant informasjon, diskursinformasjon, osv. De sistnevnte forutsetningene er problematiske når man stiller seg oppgaven å implementere en parser idag. Intonasjonsinformasjon forutsetter enten at talegjenkjenningsproblemet i vid forstand er løst, hvilket det ikke er, eller at intonasjonsinformasjon på en eller annen måte kodes sammen med parserens input. Dertil trenger vi en troverdig og generell teori om diskursrepresentasjon, og til tross for lovende ansatser finnes ikke en slik teori idag. Et siste argument for parsere som bare kommer ut med ett resultat, kan være antagelser om at der finnes et rent syntaktisk definert hierarki mellom syntaktisk flertydige lesninger som gir korrekte preferanser. Abney (1987) er et relevant arbeid i denne sammenheng, men jeg må innrømme at jeg synes det er vanskelig å se at man kan skille mellom lesningene av setning (2) på et rent syntaktisk grunnlag.

For å summere opp; dersom man ønsker å lage en parser for naturlige språk kommer man ikke utenom at den av og til må kunne returnere mer enn ett resultat. Dermed distanserer vi oss fra en "naiv" deterministisk parser som bare finner en analyse av et input.

4. Determinisme og setningsgjenkjenning

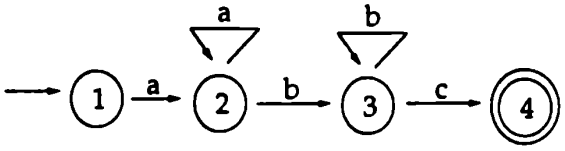
La oss først klargjøre hva man normalt forstår med termen "determinisme". Vi går da til automatteori og skillet mellom deterministiske og ikke-deterministiske automater (en automat er en slags "abstrakt" maskin som kan utføre nærmere definerte oppgaver). La oss for enkelhets skyld se nærmere på en deterministisk og en ikke-deterministisk automat som gjenkjenner språket $a^n b^m c$ for n,m > 0; altså et (kunstig) språk der setningene alltid består av et vilkårlig antall *a*'er etterfulgt av et vilkårlig antall *b*'er samt en *c* til slutt. Automat A er ikke-deterministisk og automat B er deterministisk:

(4) **Automat A:**	Initiale tilstander:	{1}
	Finale tilstander:	{4}
	Instruksjoner:	Fra 1 til 1 ved symbol a
		Fra 1 til 2 ved symbol a
		Fra 2 til 2 ved symbol b
		Fra 2 til 3 ved symbol b
		Fra 3 til 4 ved symbol c



Automaten fremstilt som transisjonsdiagram:

Automat B:	Initiale tilstander:	{1}
	Finale tilstander:	{4}
	Instruksjoner:	Fra 1 til 2 ved symbol **a**
		Fra 2 til 2 ved symbol **a**
		Fra 2 til 3 ved symbol **b**
		Fra 3 til 3 ved symbol **b**
		Fra 3 til 4 ved symbol **c**



Automaten fremstilt som diagram:

Den deterministiske automaten trenger en enkel algoritme til å fortolke og benytte den i en analyse:

1. Finn en instruksjon der første symbol i inputstrengen er lik det symbol som er nevnt i instruksjonen, og der instruksjonens "fra"-tilstand er en initial tilstand (en tilstand som er element i mengden av initiale tilstander).
2. Når en slik instruksjon *i* er funnet, les inputstrengens neste symbol. Let etter en instruksjon *ii* som er slik at dette symbolet er lik symbolet som er nevnt i instruksjon *ii*, og *is* "til"-tilstand er den samme som *iis* "fra"-tilstand. Gjenta denne prosessen inntil hele inputstrengen er lest eller det ikke finnes matchende instruksjoner.
3. Når step 1 og 2 er ferdig, avgjør om følgende kriterier er sanne: (i) instruksjon *iis* "til"-tilstanden er element i mengden av finale tilstander, og (ii) alle symbolene i inputstrengen er lest. Hvis begge kriteriene er oppfylte, er setningen akseptert. Hvis ikke er den ikke en setning i språket slik det er definert av automaten.

5. Ikke-determinisme og setningsgjenkjenning

Den ikke-deterministiske automaten (Automat A), derimot, krever en noe mer sofistikert fortolkningsalgoritme fordi den må utstyres med en *hukommelse* som gjør det mulig å gå tilbake og prøve alternative muligheter fra en gitt analysesituasjon. Det finnes flere velkjente teknikker for å oppnå dette; den kanskje mest kjente og konseptuelt enkleste å forstå er *backtracking*. Den deterministiske algoritmen modifiseres på følgende måte når den gjøres til en (noe forenklet) backtracker:

1. Finn alle instruksjoner der første symbol i inputstrengen er lik det symbol som er nevnt i instruksjonen, og der instruksjonens "fra"-tilstand er en initial tilstand (en tilstand som er element i mengden av initiale tilstander).
2. Når de er funnet, **velg** en av dem. Kall den *i*. De andre tas vare på sammen med informasjon om hvilket symbol som var i ferd med å bli lest. Les inputstrengens neste symbol. Finn alle instruksjoner *ii* som er slik at dette symbolet er lik symbolet som er nevnt i instruksjon *ii*, og *is* "til"-tilstand er den samme som *iis* "fra"-tilstand. Gjenta denne prosessen inntil inputstrengen er tom eller det ikke finnes matchende instruksjoner.
3. a. Når step 1 og 2 er ferdig, avgjør om følgende kriterier er sanne: (i) det finnes minst en instruksjon *ii* hvis "til"-tilstand er element i mengden av finale tilstander, og (ii) alle symbolene i inputstrengen er lest. Hvis (i) og (ii) er oppfylte, er setningen akseptert.
   b. Undersøk om det finnes noen alternativer som ikke er forsøkt ennå. Hvis alternativer finnes, velg det som sist ble tatt vare på, og fortsett analysen fra punkt 2.

c. Hvis kriteriene (i) og (ii) under 3.a. aldri er blitt tilfredsstilt under analysen, samt at det ikke finnes noen uprøvede alternativer, er strengen ikke en setning i språket slik det er definert av automaten.

Merk at denne algoritmen finner frem til *alle* mulige veier frem til korrekte analyser av en streng. Det er presis en slik egenskap vi er ute etter for å kunne gripe syntaktisk flertydighet i *parsing* av setninger i naturlige språk. Forskjellen mellom en gjenkjenner og en parser er at gjenkjenneren kun svarer benektende eller bekreftende på om en foreslått streng er grammatisk eller ikke, mens parseren produserer strukturelle representasjoner av velformede setninger. Dersom mengden av representasjoner etter parsingen er tom, er setningen ugrammatisk. Dersom den har ett element, er setningen entydig, to elementer viser tvetydighet, osv.

Den "deterministiske" algoritmen bør ikke appliseres på ikke-deterministiske automater fordi den ikke er istand til å ta vare på alternative instruksjoner i en gitt situasjon. Derfor klarer den ikke å fortolke Automat A på en måte som gjør at språket $a^n b^m c$ gjenkjennes. Den fortolker Automat B som vi ønsker nettopp fordi automaten er deterministisk og det aldri er mer enn en mulighet videre fra et punkt i analysen.

Backtrackeren oppviser altså den egenskap å kunne håndtere ikke-deterministiske automater. Den kan også benyttes i fortolkningen av mer velkjente PS-grammatikker for naturlige språk. Alternativer til backtrackere finnes; i datalingvistikken er særlig datastrukturer som representerer et *chart* etterhvert blitt populære, særlig fordi chartet organiserer informasjon på en mer effektiv måte enn en standard backtracker gjør, samt at det gir bedre muligheter til å studere selve analyseprosessen.

Likevel, ikke-deterministiske parsere har enkelte negative egenskaper. En har å gjøre med effektivitet; når mengden av PS-regler blir stor og setningene som skal analyseres er lange, kan slike parsere bli trege, selv om eksponensialitet unngås, jf. det kompleksitetsteoretiske uttrykket $|G|^2 \cdot n^3$ der $|G|$ er grammatikkstørrelsen (mengden av symboler i grammatikken) og n er setningslengden. Det bør innskytes at å *verifisere* om en foreslått analyse er korrekt oftest er fort gjort, mens det å *finne* den/de korrekte analysene er et adskillig hardere problem. Dersom man går utover det rene PS-regelformatet og introduserer metaregler, ID-regler, LP-restriksjoner, trekkperkolering, unifikasjon, Kasusmarkering osv., kan man få parsere med ukjente kompleksitetsegenskaper. De kan bli noe raskere, men de kan også bli markant tregere. En slik kompleksitetsdiskusjon er ikke emne for denne artikkelen og jeg lar den derfor ligge, men den grunnleggende *årsaken* til kompleksiteten må påpekes, og den har å gjøre med *mengden av hypoteser som må konsulteres, eller rettere, kombineres, under analysen*; derfor den treffende termen "blind combinatorial search". Dersom en formalisme eller et analyseredskap (en parser) ikke setter noen grense for mengden av slike hypoteser, er den *potensielt* ineffektiv, selv om den ikke trenger å være det i praktiske applikasjoner.

6. Determinisme-egenskaper og deterministiske parsere

Med dette utgangspunktet synes det rimelig å hevde at dersom muligheten til kombinatorisk søking begrenses, vil årsakene til komputasjonell ineffektivitet langt på vei elimineres, kanskje elimineres helt. Determinisme er en nærliggende mulighet, men som vi husker av diskusjonen ovenfor, har determinismen den uheldige egenskap at den gjør det umulig å frembringe mer enn en analyse av et input. Man kan da spørre om det er mulig å operere med en alternativ definisjon av determinisme som både ivaretar ønsket lingvistisk dekningsgrad og som unngår potensielle kompleksitetsproblemer. La oss forsøke å komme frem til et slikt determinismebegrep.

En deterministisk maskin har følgende essensielle egenskaper:

A: Den kan ikke "glemme" eller "ødelegge" hypoteser på veien mot en løsning av en oppgave eller et problem.

B: Bare én regel (eller instruksjon) kan slå til når parseren (eller automaten) er i en gitt tilstand.

Alle valg den foretar i løpet av analysen er ugjenkallelige. I automatteori er denne egenskapen ivaretatt ved transisjonsfunksjonen, dvs. beskrivelsen av hvordan instruksjonene ser ut. For en finitt automat (fa) går den fra mengden av tilstander og inputalfabetet til mengden av tilstander (fra S x A til S der *S* er en finitt mengde tilstander og *A* er alfabetet). Siden man her snakker om en *funksjon* vil det for ethvert par av tilstander og symbol fra alfabetet finnes en unik tilstand. Enhver deterministisk fa må tilfredsstille dette kravet. Determinismekravet til en pushdown-automat (pda) må også ta pushdown-stacken i betraktning. En pda er deterministisk dersom det for enhver kombinasjon av tilstand, inputsymbol og stackelement kun finnes ett par av tilstander og stack-symboler, dvs. en mapping fra S x A x Γ til S x Γ der *S* er en finitt mengde tilstander, *A* et finitt sett symboler som utgjør alfabetet og Γ et finitt sett av stack-symboler. (For enkelhets skyld ser vi bort fra ε-transisjoner).

De automatene vi kjenner fra automatteori har i tillegg et par andre underforståtte og viktige egenskaper som skiller dem fra deterministiske Turingmaskiner (dvs. vanlige "serielle" datamaskiner). For det første kan de ikke benytte en vilkårlig mengde "kladdetaper" til å beregne hjelpehypoteser underveis. Bare pda'er, lineært avgrensede automater og stack-automater tillates å benytte slike ekstrataper, men de bruker kun én. Videre kan automatene ikke skrive nye symboler på eller vilkårlig fjerne symboler fra inputtapen. For det tredje kan de ikke bevege skrivehodet vilkårlig frem og tilbake på tapen; de kan bare se det symbol som til enhver tid står lengst til venstre på tapen. Et fjerde poeng er at de ikke tillates å scanne gjennom hele strengen først, ta vare på det de ser, og deretter starte analysen fra begynnelsen.

Dersom vi beveger oss over til parsingslitteraturen finner vi at de fleste av disse egenskapene etterleves, men ofte med visse modifikasjoner. Såkalte LR(k) parsere kan "se" *k* symboler videre inn i inputstrengen. En LR(k,t) parser kan også se *k* symboler inn i inputstrengen, men den kan i tillegg utsette tilordningen av *t* ferdig analyserte delkonstituenter (f.eks. nominalfraser og adjektivfraser). Parseren til Marcus (1980) er påstått å være LR(2,2)[2]. LR(k) og LR(k,t) parsere gjenkjenner deterministisk kontekstfrie språk. I tillegg kommer LR* parserne som kan se ubegrenset langt fremover i inputstrengen.

Alle disse parserne er deterministiske, men ingen av dem er istand til å håndtere syntaktisk flertydighet av den art som ble beskrevet innledningsvis.

7. Determinisme og syntaktiske flertydigheter

7.1. To typer determinisme

La oss anta to typer deterministiske innretninger; "løse" ("sloppy") og "strenge" deterministiske maskiner. En streng deterministisk maskin forstår vi heretter som en "klassisk" deterministisk maskin som aldri returnerer mer enn ett analyseresultat. En løs deterministisk maskin kan derimot returnere mer enn et resultat, men den er underlagt restriksjoner som vi skal presisere i det som følger.

Vi trenger et term for å omtale mengden av korrekte analyser for et gitt input. Derfor skal vi snakke om mengden av gyldige analyser for streng s i henhold til grammatikk *G*, forkortet som MGA(s,G). Et eksempel; MGA(s,G) for s lik Jens så mannen med kikkerten og *G* lik grammatikk (5) skal være mengden av frasemarkører i (6):

(5) S → NP VP
NP → N | N PP
VP → V | V NP | V NP PP
PP → P NP

(6) {[S [NP [N Jens] [VP [V så [NP [N mannen]] [PP [P [NP [N kikkerten]]]]]]]
[S [NP [N Jens] [VP [V så [NP [N mannen] [PP [P [NP [N kikkerten]]]]]]]]}

La oss sette termen MGA i relasjon til standard automatteori før den appliseres på parsing. Vi er interessert i en beskrivelse av hvordan automaten *fortolkes* slik at den kommer frem til MGA. For enkelhets skyld holder vi oss i første omgang til finitte tilstandsautomater. Vi beskriver egenskapene til en backtracker-fortolker av en fa: Før analysen starter, skrives et sett av sett av ordnede par av initial tilstand og ukonsumert input, f.eks. slik:

{{(1, aaabbc)}}

Kall dette settet for *mengden av analysestier*, forkortet til Σ. Analysestiene består av en mengde ordnede par av tilstander og ukonsumert input. Denne mengden er i sin tur ordnet etter lengden på ukonsumert input.

Backtrackeren prøver stiene i tur og orden. Etter hver utført instruksjon legger den til et nytt ordnet par bestående av ny tilstand og ukonsumert input minus symbolet lengst til venstre i inputstrengen. Dersom instruksjonen gir opphav til mer enn en ny tilstand, lages det kopier av stien frem til "nåværende situasjon" slik at to eller flere distinkte stier oppstår. Et eksempel:

{{(1, aaabbc) (1, aabbc)}
{{(1, aaabbc) (2, aabbc)}}

I dette tilfellet er det understrekede paret i den siste stien en kopi fra den første stien.

Etterhvert som Σ øker, konsulteres og analyseres hver sti på samme måte. Enhver sti har en endestasjon, dvs. et par som kan eller ikke kan ekspanderes videre. Dersom endestasjonen i en sti er et par bestående av en final tilstand og Ø, kaller vi den en terminert endestasjon. Enhver sti med en terminert endestasjon er et element i MGA, og MGA inneholder kun slike terminerte endestasjoner. Dersom Σ ikke inneholder stier med terminerte endestasjoner, er ikke inputstrengen velformet. Inneholder den mer enn èn sti med terminert endestasjon, er den flertydig.

Vi har nå beskrevet egenskapene til en backtracker, og vi illustrerer med følgende eksempel (vellykkede stier er markert med fete typer):

La Automat 4.B gjenkjenne strengen aaabbc. Automaten er deterministisk og produserer bare en sti, altså kun ett medlem av Σ:

Σ= { {(1,aaabbc)(2,aabbc)(2,abbc)(2,bbc)(3,bc)(3,c)(4,Ø)} }

Denne stien er også element i MGA(aaabbc, Automat B). La Automat 4.A gjenkjenne samme streng. Resultatet blir

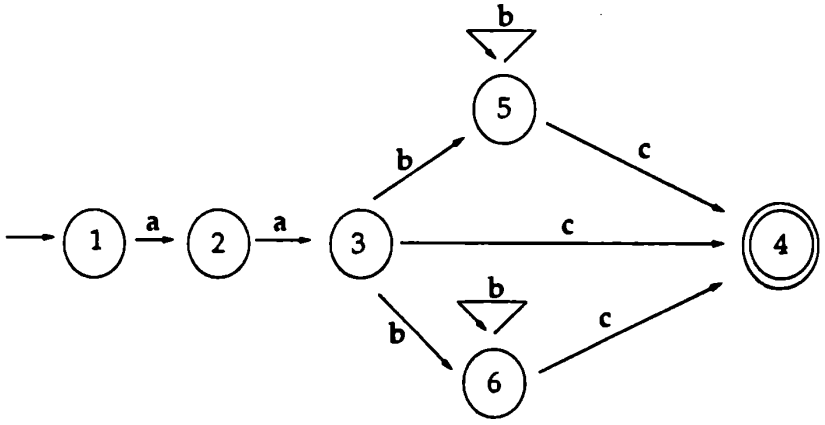
Σ= {	(1,aaabbc) (1,aabbc) (1,abbc) (1, bbc)}	Feil
	{(1,aaabbc) (1,aabbc) (1,abbc) (2,bbc) (2,bc) (2,c)}	Feil
	{(1,aaabbc) (1,aabbc) (1,abbc) (2,bbc) (2,bc) (3,c) (4,Ø)}	**Suksess**
	{(1,aaabbc) (1,aabbc) (1,abbc) (2,bbc) (3,bc)}	Feil
	{(1,aaabbc) (1,aabbc) (2,abbc)}	Feil
	{(1,aaabbc) (2,aabbc)}	Feil
	}	

I dette tilfellet er bare stien {(1,aaabbc) (1,aabbc) (1,abbc) (2,bbc) (2,bc) (3,c) (4,Ø)} element i MGA(aaabbc, Automat A). Alle de øvrige stiene er blindveier. Dette resultatet er typisk for ikke-deterministiske automater (og parsere), dvs. at MGA bare er et lite subsett av Σ.

Så langt har vi ikke sagt noe nytt, men la oss nå se på en løs deterministisk automat, som vi kaller Automat C:

(7) Automat C: Initiale tilstander: {1}
Finale tilstander: {4}
Instruksjoner:

1. Fra 1 til 2 ved a.
2. Fra 2 til 3 ved a.
3. Fra 3 til 5 ved b.
4. Fra 3 til 4 ved c.
5. Fra 3 til 6 ved b.
6. Fra 5 til 5 ved b.
7. Fra 5 til 4 ved c.
8. Fra 6 til 6 ved b.
9. Fra 6 til 4 ved c.



Automaten fremstilt som transisjonsdiagram:

Denne automaten gjenkjenner språket aab^nc for $b \geq 0$, og den gir to analyser av enhver streng i aab^nc for $b > 0$, som er et subsett av språket aab^nc for $b \geq 0$. Strengen aac, derimot, får kun en analyse. Vi lar automaten analysere *aabbc*. Resultatet blir

Σ= { {(1,aabbc)(2,abbc)(3,bbc)(5,bc)(5,c)(4,Ø)}
{(1,aabbc)(2,abbc)(3,bbc)(6,bc)(6,c)(4,Ø)} }

I dette tilfellet inneholder Σ to stier, men merk at *begge* er elementer i MGA(aaabbc, Automat C). Altså, Σ = MGA(aaabbc, Automat C).

Vi er nå istand til å definere et determinisme-hierarki: For en vanlig deterministisk automat gjelder at Σ aldri har kardinalitet > 1, mens en løs deterministisk automat tillater at Σ har kardinalitet > 1. For begge determinismetyper gjelder at automaten aksepterer input bare når hvert element av Σ også er element i MGA. For vanlige ikke-deterministiske automater er disse restriksjonene irrelevante.

Dette betyr at en løs deterministisk maskin alltid må være sikker på at enhver instruksjon som anvendes produserer et tilstand/input par som er medlem av en en sti som er element i MGA. Det er presis denne intuisjonen som er uttrykt i Nordgård (1991) der det kreves at parseren må være sikker på at inputstrengen er strukturelt flertydig før tilstandskopiering finner sted.

Et par kommentarer er på sin plass. Illustrasjonene ovenfor viser at løs determinisme ikke er det samme som blind kombinatorisk søking siden alle elementer i Σ er korrekte analyser. Blind kombinatorisk søking vil typisk føre til haugevis av mislykkede stier i Σ, og desto flere mislykkede stier, desto tregere analyse. Et annet poeng er at løs determinisme slik den er fremstilt her benytter samme prosesseringsmaskineri som blind kombinatorisk søking, selv om maskineriets muligheter bare utnyttes i begrenset omfang. Ideelt sett ønsker vi et maskineri som håndterer løs determinisme, men som ikke kan brukes til blind kombinatorisk søking.

7.2. Parseren beskrevet i Nordgård (1991)

I Nordgård (1991) diskuteres en parser som har de essensielle determinisme-egenskapene nevnt ovenfor, dvs. at den ikke kan fjerne eller glemme strukturer den har bygget underveis, og bare en regel slår til i en gitt tilstand. Men ulikt LR-parserne kan den også analysere strukturelle flertydigheter. Den har likheter med andre deterministiske analyseredskaper:

(a) Den kan ikke gå frem og tilbake i inputstrengen, flytte eller legge til symboler i inputstrengen.
(b) Den minner om en LR* parser siden den til enhver tid kan se hele inputstrengen som et regulært uttrykk, altså en større uttrykkskraft enn LR(k), LR(k,t) og Marcus-parserne.
(c) I likhet med en stack-automat kan den se hele "stack'en" den har bygd (dvs. de konstituentene i trestrukturen som dominerer den konstituenten som er i ferd med å bygges), altså mer enn en pda og Marcus-parseren.

Til forskjell fra en stack-automat kan parseren modifisere innholdet på stack'en (her forstått som node i trestrukturen som dominerer den noden som er i ferd med å bygges), f.eks. ved å legge til bindingsrelevant informasjon ("node x binder node y").

I tillegg har den et par andre egenskaper som gjør den spesiell. For det første kan den bygge et "templat" som den kan etterfylle tomme plasser i. Den kan således bevege seg inne i stack'en og modifisere dens innhold uten hele tiden å måtte forholde seg til "øverste" stackelement, jf. Marcus-parseren og pda'er. Heller ikke dette affekterer determinismen, men det kan angå generativ kraft (dvs. hvilke formelle språk den er istand til å gjenkjenne), noe som kan være verdt å diskutere ytterligere. Den andre særegenheten er at den kan søke oppover og nedover i treet vha. deterministiske fa-teknikker. Heller ikke dette angår determinismen, men det fortjener å settes i relasjon til effektivitet. Ingen av disse egenskapene rører ved den essensielle determinisme-egenskapen, men de er relevante for parserens eventuelle psykologisk plausibilitet og effektivitet.[3]

7.3 En løs deterministisk analyse av syntaktiske flertydigheter

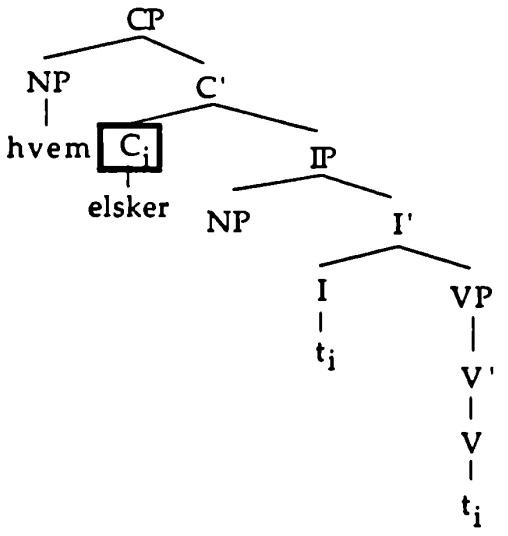
La se hvordan parseren beskrevet i Nordgård (1991) håndterer syntaktiske flertydigheter vha. løs determinisme.

Vi ser nærmere på setning (2) igjen:

(2) Hvem elsker Marit? (2 lesninger)

Parseren vil, når den er i følgende tilstand, konsultere en heuristisk regel, kall den regel #1:[4]

(8)



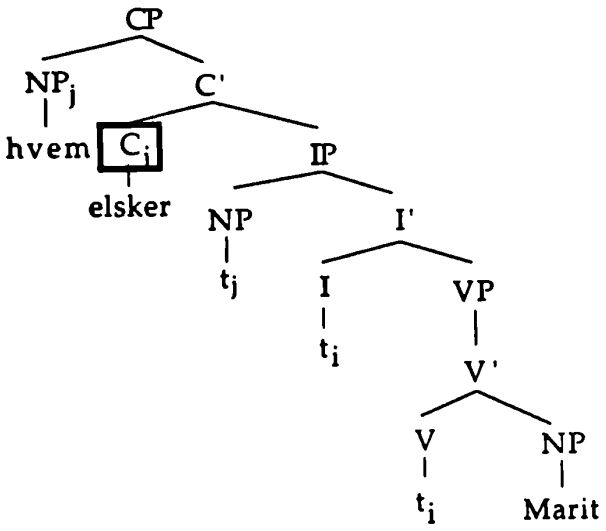
Input: # Marit #

(9) Heuristisk regel #1:

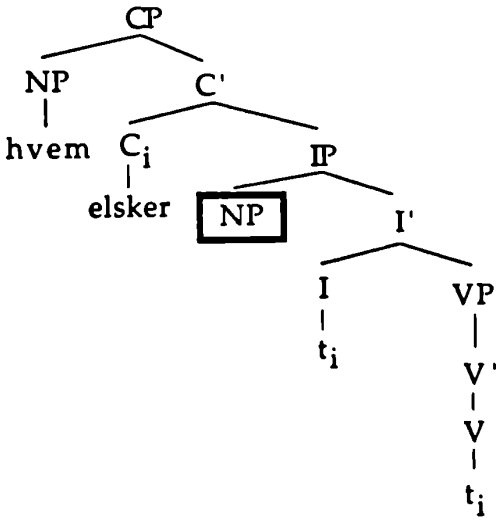
Tree Condition:	↑ : [Head CP], (Transitive) = yes. ∧ [Spec (category C)], (category) = N, (barlevel) = 2, (binder-of) = { }.
String Condition:	From !1 to !1 by Adv From !1 to 0! by N From !1 to 0! by Det From !1 to 0! by A
Action:	CopyState (PopQWP) EC (∧ [Spec (category C)] (∨ [Spec, (category I)]

Regelen krever at parserens oppmerksomhet er rettet mot posisjonen [Head, CP] og at den inneholder et transitivt verb ("oppmerksomhet" er symbolisert ved "↑"). Dessuten skal [Spec,CP] inneholde en NP som ikke binder noe.[5] Videre skal inputstrengen inneholde et nomen eller en determinativ eller et adjektiv eller et adverb etterfulgt av N, Det eller A.[6] Dersom disse kravene holder, skal tilstanden kopieres sammen med en instruksjon om at oppmerksomhen skal flyttes til neste ventende posisjon ([Spec,IP]). I den originale strukturen skal det plasseres et spor i [Spec,IP] bundet fra [Spec,CP]. Resultatet av den første parsingen blir som i (10), mens kopien som starter opp fra tilstand (11) returnerer (12):

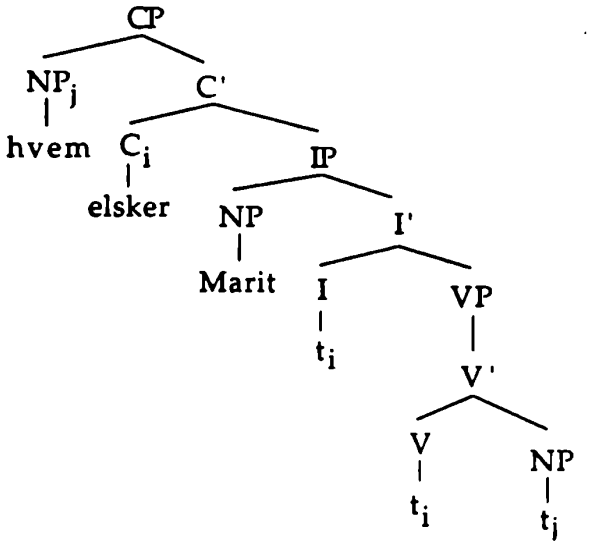
(10)


CP
NP$_j$
C'
hvem
C$_j$
IP
elsker
NP
I'
t$_j$
I
VP
t$_i$
V'
V
NP
t$_i$
Marit

(11)


CP
NP
C'
hvem
C$_i$
IP
elsker
NP
I'
I
VP
t$_i$
V'
V
t$_i$

(12)



Detaljene i parsingen går vi ikke inn på her; leseren henvises til Nordgård (1991, 1992). Det sentrale poeng er imidlertid at parseren ikke har laget mer struktur under parsingen enn nettopp de to ønskede analysene. Derfor er determinismekravet tilfredsstilt, og parseren har bygget to representasjoner vha. løs determinisme.

8. Løs determinisme og ikke-deterministiske språk

I formell språkteori relateres gjerne distinksjonen determinisme/ikke-determinisme til distinksjonen deterministiske/ikke-deterministiske språk. Et ikke-deterministisk språk synes å kreve ikke-deterministiske analysealgoritmer. Språket karakterisert som mengden av speilbildestrenger over {a, b} kan ikke gjenkjennes av en deterministisk pda, mens en ikke-deterministisk pda kan gjøre den jobben. Men det er et velkjent problem at det er vanskelig å avgjøre om et språk generert av en ikke-deterministisk pda kan genereres av en deterministisk pda. Situasjonen er den samme for deterministiske og ikke-deterministiske lineært avgrensede automater.

Et naturlig spørsmål for vår diskusjon er om alle ikke-deterministiske kontekstfrie språk kan kan genereres vha. løs determinisme. Svaret er negativt siden det er vanskelig å se hvordan speilbildespråket over {a,b}* kan gjenkjennes vha. en løs deterministisk pda.[7]

Løs determinisme synes ikke å være spesielt relevant for diskusjoner av generativ kraft. Begrepet er derimot egnet til å karakterisere et subsett av ikke-deterministiske maskiner eller grammatikker som som *ikke* utnytter ikke-determinismens muligheter til å forfølge feilaktige hypoteser. Denne klassen av grammatikker er interessante både i et komputasjonelt og psykolingvistisk perspektiv. Den komputasjonelle fordelen ligger i at det ikke blir utført mer arbeid enn nødvendig under prosesseringen. Hva psykolingvistikken angår burde man vha. løs determinisme kunne lage mer troverdige modeller der også syntaktiske flertydigheter gis adekvate analyser.

9. Oppsummering

Jeg har i det foregående diskutert ulike sider ved determinisme, og jeg har introdusert begrepet løs determinisme. Det er vist hvordan løs determinisme forstås i relasjon til finitte automater og hvordan begrepet kan benyttes i parsing. Et gjenværende spørsmål er hvorvidt potensiell ineffektivitet lurer i bakgrunnen i prosesseringen av løst deterministiske grammatikker. Svaret er både ja og nei. Negativt fordi parseren aldri kan plukke opp alle kompatible utveier fra en gitt tilstand, altså at flere regler matcher en parsingstilstand. Positivt fordi prosesseringen kan føre til mislykkede resultat, enten ved kopien eller den "originale" strukturen. Men uansett er ett viktig aspekt ved blind kombinatorisk søking eliminert. Det andre kan bare ivaretas ved at grammatikkreglene er etterlever determinismekravet, altså et spørsmål om grammatikkskriverens dyktighet.

Noter

1 Dette er en GB-analyse der hv-frasen **hvem** har flyttet fra objektsposisjonen e_i og verbet har flyttet fra hodeposisjonen e_j i verbfrasen.

2 Se Berwick (1985).

3 For en diskusjon av generativ kraft og effektivitet til parseren beskrevet i Nordgård (1991), se Nordgård (1992b).

4 Plassen tillater ikke en fremstilling av hvordan struktur (8) er produsert. Leseren henvises til Nordgård (1991) eller Nordgård (1992).

5 Operatoren "$\wedge$" betyr at det skal søkes oppover i treet fra "oppmerksomhetsposisjonen". "$\vee$" fortolkes som søking nedover i treet.

6 En forklaring av notasjonskonvensjonene til strengautomaten er påkrevet: !n betyr initial tilstand og n! betyr final tilstand. !n! er både initial og final tilstand.

7 Et annet spørsmål er om speilbildespråket over {a,b} kan gjenkjennes av en løs deterministisk pda med ubegrenset lookahead. Se Nordgård (1992b) for en diskusjon av dette og enkelte andre emner relatert til løs determinisme.

Referanser

Abney, S. (1987) "Licensing and Parsing", i *Proceedings of NELS 17, Volume 1*. Dept. of Linguistics, University of Massachusetts, Amherst.

Berwick, R. (1985) *The Acquisition of Syntactic Knowledge*. MIT Press.

Nordgård, T. (1991) A GB-Related Parser for Norwegian. Upublisert doktoravhandling, Institutt for fonetikk og lingvistikk, Universitetet i Bergen.

Nordgård, T. (1992) *A GB-Related Parser for Norwegian*. Det historisk-filosofiske fakultets publikasjonsserie, Universitetet i Bergen.

Nordgård, T. (1992b) "On the efficiency of E-Parser", upublisert manuskript, Institutt for fonetikk og lingvistikk, Universitetet i Bergen.

Torbjørn Nordgård
Institutt for fonetikk og lingvistikk
Universitetet i Bergen
Sydnesplass 9
N-5007 Bergen
E-mail: nordgaard@hf.uib.no