Gregers Koch
Datalogisk Institut, Københavns Universitet

EN PROBLEMORIENTERET PROGRAMMELUDVIKLINGSMETODE I LINGVISTISK DATABEHANDLING

1. Indledning

Fortolkning af prædikatkalkyle som et programmeringssprog udgør en ny og lovende datalogisk metode, som ofte kaldes logikprogrammering. Prædikatlogiske notationer kan betragtes som højniveau, menneskevenlige programmeringssprog som kan anvendes til praktisk programmering såvel som til teoretiske undersøgelser. Specielt i forbindelse med datalingvistiske problemstillinger synes metoden lovende. Dette kommer skriftet her nærmere ind på, og desuden diskuteres nogle forsøg på udvidelse af metoden.

Prædikatkalkyle synes at være af stadigt stigende interesse for datamatisk orienterede lingvister [Charniak og Wilks 76]. Samme tendens synes at gøre sig gældende inden for kunstig intelligens [Nilsson 80].

Med fremkomsten af logikprogrammeringssprog som Prolog [Bowen 79] kan man se nye perspektiver i denne udvikling, tildels på grund af muligheden for effektiv udførelse af inferenser som nødvendigvis knytter sig til sådanne systemer.

2. Metoden

Definitte klausuler (også kaldet Hornklausuler) [Colmerauer 78, Kowalski 74, 79, Mayoh 80] er formler af formen

$$C_0 \leftarrow C_1, C_2, \ldots, C_n$$

hvor alle Cerne er prædikatudtryk der indeholder

- variable $X, Y, Z, \ldots, X_1, Y_1, Z_1, \ldots$
- konstanter
- funktionsnavne.

Idet universel kvantificering er underforstået, kan en sådan formel betragtes som ækvivalent til prædikatkalkyleformlen

$$\forall X_1, X_2, \ldots, X_m: ((C_1 \wedge C_2 \wedge \ldots \wedge C_n) \rightarrow C_0)$$

hvor Xerne netop udgør sættet af samtlige variable i Cerne. Specielt kan betingelserne være tomme ($n = 0$) svarende til simple påstande, eller konklusionen C_0 kan mangle svarende til en negation (også kaldet en målklausul), eller begge dele

En problemorienteret programmeludviklingsmetode i lingvistisk databehandling
Gregers Koch
Proceedings of NODALIDA 1981, pages 47-63

svarende til den tomme klausul eller umulige påstand.

En kontekstfri grammatiks produktionsregler af formen

$$\text{Nonterminal} \rightarrow B_1\ B_2\ \ldots\ B_n$$

kan omformes til følgende formel fra første ordens prædikatkalkyle

$$\forall S_0,S_1,\ldots,S_n\colon ((B_1(S_0,S_1) \wedge B_2(S_1,S_2) \wedge \ldots \wedge B_n(S_{n-1},S_n)) \rightarrow \text{Nonterminal}\ (S_0,S_n))$$

med følgende mening:

- hele teksten fra position S_0 til position S_n kan fortolkes som et objekt tilhørende kategorien Nonterminal, såfremt
- teksten fra position S_0 til position S_1 kan fortolkes som et B_1-objekt, og
- teksten fra position S_1 til position S_2 kan fortolkes som et B_2-objekt, og

. . . . . .

- teksten fra position S_{n-1} til position S_n kan fortolkes som et B_n-objekt.

Et lille eksempel er følgende kontekstfrie grammatik:

Sentence	→	the Noun Verb	
Noun	→	woman	(1)
Verb	→	lives	
Verb	→	smells	

som kan omformes til følgende prædikatlogiske formler:

$$\forall S_0,S_1,S_2,S_3((C(\underline{\text{the}},S_0,S_1) \wedge \text{Noun}(S_1,S_2) \wedge \text{Verb}(S_2,S_3)) \rightarrow \text{Sentence}(S_0,S_3))$$

$$\forall S_0,S_1 : (C(\underline{\text{woman}},S_0,S_1) \rightarrow \text{Noun}(S_0,S_1))$$

$$\forall S_0,S_1 : (C(\underline{\text{lives}},S_0,S_1) \rightarrow \text{Verb}(S_0,S_1)) \quad (2)$$

$$\forall S_0,S_1 : (C(\underline{\text{smells}},S_0,S_1) \rightarrow \text{Verb}(S_0,S_1))$$

Hvis vi ønsker at se om

the	woman	smells
1	2	3 4

er en sætning fra den lille grammatik kan vi tilføje følgende påstande

$C(\underline{\text{the}},1,2)$

$C(\underline{\text{woman}},2,3)$

$C(\underline{\text{smells}},3,4)$.

Problemet er nu om systemet er konsistent, og om det er muligt at deducere Sentence (1,4) som et teorem inden for systemet.

Den lille grammatik (1) kan også udtrykkes som definitte klau-

suler i stil med [Pereira & Warren 80]:

$$\begin{aligned}
&\text{Sentence}(S_0,S_1) \leftarrow C(\underline{\text{the}},S_0,S_1),\ \text{Noun}(S_1,S_2),\ \text{Verb}(S_2,S_3)\\
&\text{Noun}(S_0,S_1) \leftarrow C(\underline{\text{woman}},S_0,S_1)\\
&\text{Verb}(S_0,S_1) \leftarrow C(\underline{\text{lives}},S_0,S_1)\\
&\text{Verb}(S_0,S_1) \leftarrow C(\underline{\text{smells}},S_0,S_1)\ .
\end{aligned} \tag{3}$$

Som et lidt større eksempel i samme retning kan vi kigge på syntaksanalyse i henhold til følgende lille grammatik

$$\begin{aligned}
&\text{S} \rightarrow \text{NP VP [ADVP]}\ .\\
&\text{NP} \rightarrow \text{[DET] ADJ* N}\ .\\
&\text{NP} \rightarrow \text{PRON}\ .\\
&\text{ADVP} \rightarrow \text{PREPP}\ .\\
&\text{ADVP} \rightarrow \text{ADV}\ .\\
&\text{PREPP} \rightarrow \text{PREP NP}\ .\\
&\text{VP} \rightarrow \text{V [NP] [ADVP]}\ .
\end{aligned} \tag{4}$$

Denne grammatik omformes til definitte klausuler ved simplificering (som her vil sige eliminering af valgfrie elementer [...] samt repetitive elementer ...*) samt ved tilføjelse af positionsangivelser (x,y,z,w):

$$\begin{aligned}
&\text{S}(x,y) \leftarrow \text{NP}(x,z),\ \text{VP}(z,w),\ \text{ADVP}(w,y)\\
&\text{S}(x,y) \leftarrow \text{NP}(x,z),\ \text{VP}(z,y)\\
&\text{NP}(x,y) \leftarrow \text{DET}(x,z),\ \text{ADJLIST}(z,w),\ \text{N}(w,y)\\
&\text{NP}(x,y) \leftarrow \text{ADJLIST}(x,z),\ \text{N}(z,y)\\
&\text{NP}(x,y) \leftarrow \text{PRON}(x,y)\\
&\text{ADJLIST}(x,y) \leftarrow\\
&\text{ADJLIST}(x,y) \leftarrow \text{ADJ}(x,z),\ \text{ADJLIST}(z,y)\\
&\text{ADVP}(x,y) \leftarrow \text{PREPP}(x,y)\\
&\text{ADVP}(x,y) \leftarrow \text{ADV}(x,y)\\
&\text{PREPP}(x,y) \leftarrow \text{PREP}(x,z),\ \text{NP}(z,y)\\
&\text{VP}(x,y) \leftarrow \text{V}(x,z),\ \text{NP}(z,w),\ \text{ADVP}(w,y)\\
&\text{VP}(x,y) \leftarrow \text{V}(x,z),\ \text{NP}(z,y)\\
&\text{VP}(x,y) \leftarrow \text{V}(x,z),\ \text{ADVP}(z,y)\\
&\text{VP}(x,y) \leftarrow \text{V}(x,y)\ .
\end{aligned} \tag{5}$$

En inddatastreng som "De kommer på skadestuen" kan analyseres ved tilføjelse af følgende leksikalinformation

```
PRON(x,y)   ← C(de,x,y)
V(x,y)      ← C(kommer,x,y)
PREP(x,y)   ← C(på,x,y)
N(x,y)      ← C(skadestuen,x,y)
C(de,1,2)   ←                                   (6)
C(kommer,2,3)    ←
C(på,3,4)        ←
C(skadestuen,4,5) ←
←S(1,5) .
```

Bemærk at vi intetsteds specificerer hvilken analysealgoritme der ønskes anvendt. Vi specificerer kun problemet, så finder systemet selv ud af, hvordan problemet skal håndteres.

3. Kasussystemer

Sagt ultrakort agiteres der her for en datalogisk metode som går ud på at udsætte datalingvistiske problemer for en datamatisk behandling som om de var logiske problemer, og der søges argumenteret for det fordelagtige i denne metode fra et datalogisk synspunkt.

Påstanden er således at så at sige enhver datalingvistisk teori eller strategi ville profitere af at benytte denne metode. Som eksempler har jeg beskæftiget mig med Schanks "Conceptual Dependency" [Schank 75] og Parker-Rhodes' "Inferential Semantics" [Parker-Rhodes 78, Jørgensen 80].

Fremstillingen her ligger nærmest Parker-Rhodes, medens fremstillingen i [Koch 80/16] har flere lighedspunkter med Schanks teorier.

Det må understreges at fremstillingen her kun skal ses som et eksempel der belyser mulighederne ved at anvende denne metode til realisering af givne datalingvistiske teorier. (Således udelades her flere aspekter bl.a. tempusangivelser og numerusangivelser).

Da begge forfattere vedkender sig en vis gæld til [Fillmore 68], kan disse to teorier med nogen ret betragtes som kasussystemer.

Lad os først behandle en række små eksempler fra [Schank 75]:

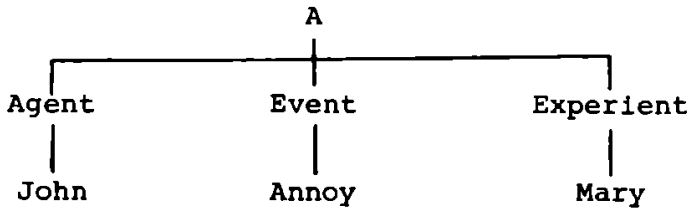
Eksempel 1

John annoyed Mary.

Den forudsatte leksikalske information kan være

(Annoy Agent (Experient)).

Et rimeligt syntakstræ kan være



I så fald kan det forventede resultat af syntaksanalysen være følgende listestruktur

[A [Agent John]
[Event Annoy]
[Experient Mary]].

Et rimeligt resultat af oversættelsen kan være

Event(A,Annoy)
Agent(A,John)
Experient(A,Mary). □

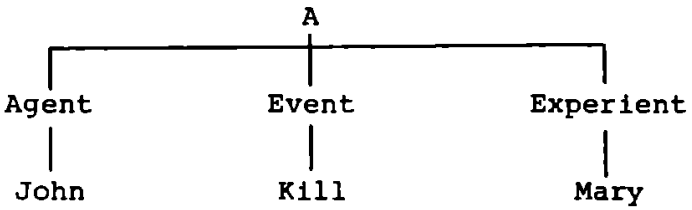
Eksempel 2

John killed Mary.

Den forudsatte leksikalinformation kan være

(Kill Agent (Experient) (by Instrument)) .

Et rimeligt syntakstræ kan være



Det forventede resultat af syntaksanalysen kan være

[A [Agent John]
[Event Kill]
[Experient Mary]].

Et rimeligt resultat af oversættelsen kan være

Event(A,Kill)
Agent(A,John)
Experient(A,Mary). □

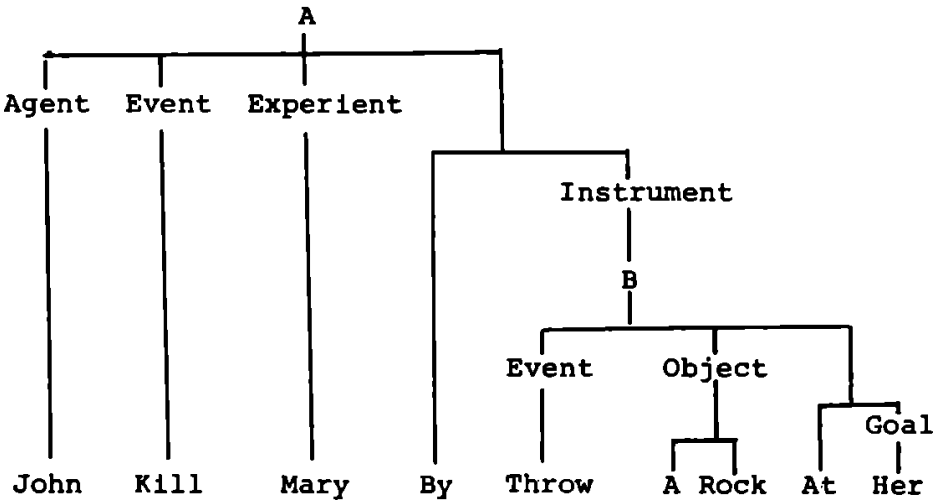
Eksempel 3

John killed Mary by throwing a rock at her.

Den forudsatte leksikalske information kan være

(Throw Agent (Object) (at Goal))
(Mary Person Female)
(John Person Male).

Et rimeligt syntakstræ kan være



Det forventede resultat af syntaksanalysen kan være

```
[A [Agent John]
   [Event Kill]
   [Experient Mary]
   [By [Instrument [B [Event Throw]
                      [Object [A Rock]]
                      [At [Goal Her]]]]]].
```

Et rimeligt resultat af oversættelsen kan være

```
Event(A,Kill)
Agent(A,John)
Experient(A,Mary)
Instrument(A,B)
Event(B,Throw)
Object(B,Rock)
Goal(B,Mary)
Agent(B,John) .
```

□

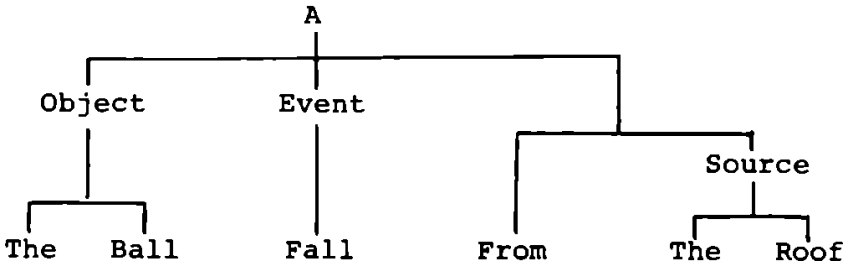
Eksempel 4

The ball fell from the roof.

Den forudsatte leksikalske information kan være

(Fall Object (from Source)).

Et rimeligt syntakstræ kan være



Et rimeligt resultat af oversættelsen kan være

Event(A,Fall)
Object(A,Ball)
Source(A,Roof).

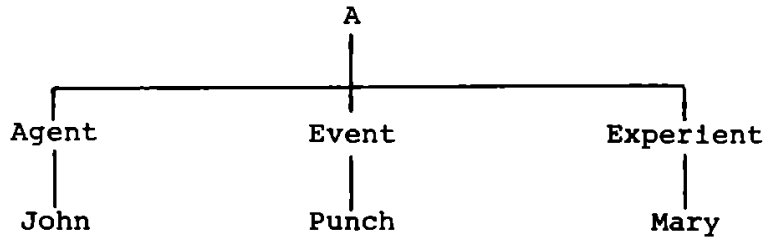
Eksempel 5

John punched Mary.

Den forudsatte leksikalske information kan være

(Punch Agent (Experient)).

Et rimeligt syntakstræ kan være



Et rimeligt resultat af oversættelsen kan være

Event(A,Punch)
Agent(A,John)
Experient(A,Mary).

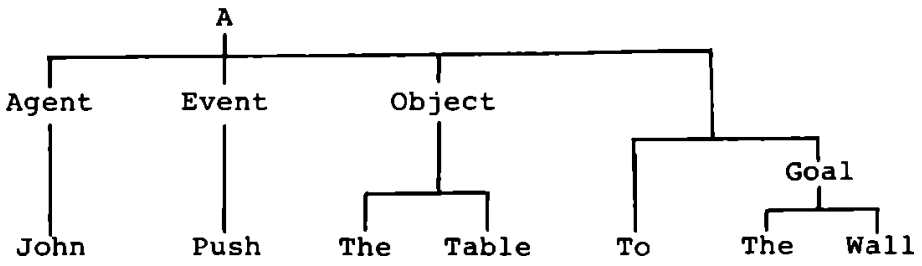
Eksempel 6

John pushed the table to the wall.

Den forudsatte leksikalske information kan være

(Push Agent (Object) (to Goal))

Det forventede resultat af syntaksanalysen kan være



Et rimeligt resultat af oversættelsen kan være

Event(A,Push)
Agent(A,John)
Object(A,Table)
Goal(A,Wall). □

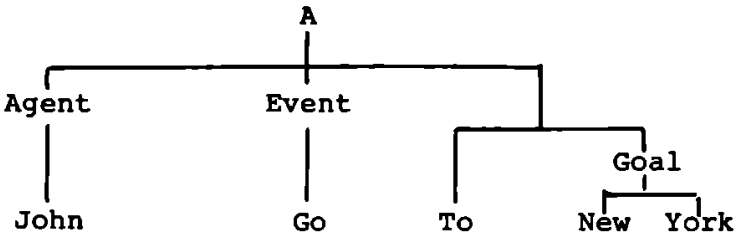
Eksempel 7

John went to New York.

Den forudsatte leksikalske information kan være

(Go Agent (from Source) (to Goal)) .

Et rimeligt syntakstræ kan være



Et rimeligt resultat af oversættelsen kan være

Event(A,Go)
Agent(A,John)
Goal(A,New York) . □

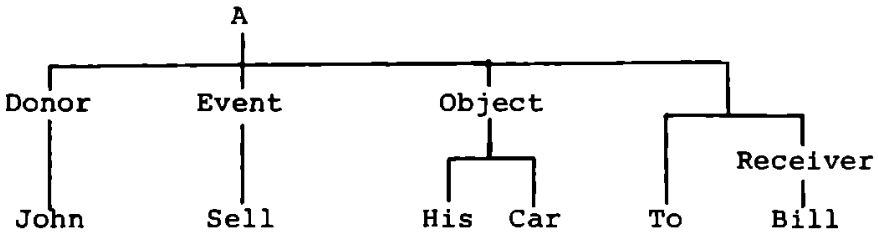
Eksempel 8

John sold his car to Bill.

Den forudsatte leksikalske information kan være

(Sell { ((Receiver) Object) / (Object) (to Receiver) }) .

Et rimeligt syntakstræ kan være



Et rimeligt resultat af oversættelsen kan være

Event(A,Sell)
Donor(A,John)
Object(A,Car)
Receiver(A,Bill). □

På baggrund af sådanne eksempler søger vi nu at realisere oversættelsen. Generering af syntakstræ er ganske nemt ud fra metoden i det foregående afsnit med brug af en grammatik som beskriver eksemplerne. Denne kan for eksempel udtrykkes noget i denne retning:

Event → { Agent (Timeadv) Event\A | NP is Adj | NP have NP | Donor (Timeadv) Event\D | Object (Timeadv) Event\O }

Event\A → { EventAE (Experient) | EventAEBYI (Experient) (by Instrument) | EventAOATG (Object) ({ at | to } Goal) | EventAFSTG (from Source) (to Goal) | EventAO (Object) | EventA }

Event\D → { EventDRO ((Receiver) Object) | EventDRO (Object) (to Receiver) | EventDRD (Receiver) (Datum) | EventDRD (Datum) (to Receiver) }

Event\O → EventOFS (from Source)

{ Experient | Object | Source } → NP

Adj → short

Timeadv → often

etc.

Systemet (3) gav kun svaret ja/nej. For at få et resultat af analysen, må man tilføje ét eller flere ekstra argumenter og i disse resultatargumenter angive hvordan konklusionens resultatarguments værdi skal være.

Tilbage står at realisere oversættelsen fra syntakstræ på listestrukturform til sættet af prædikater. Dette hverv kan udføres omtrent således:

```
Make((n . (x . y),r)← M((x . y),n,r)
M(([u w] . z),n, ([u "<" n "," w ">"] . z1))
     ← Member(u,Cases), M(z,n,z1)
M ([p [u [(n1 . w)]]] . z),n,
       ([u "<" n "," n1 ">"] . z1))
     ← Member(p,Prepositions), Member(u,Cases), M(z,n1,z1).
```

Lad os nu se på endnu et par eksempler (denne gang taget fra [Parker-Rhodes 78]):

Eksempel 9

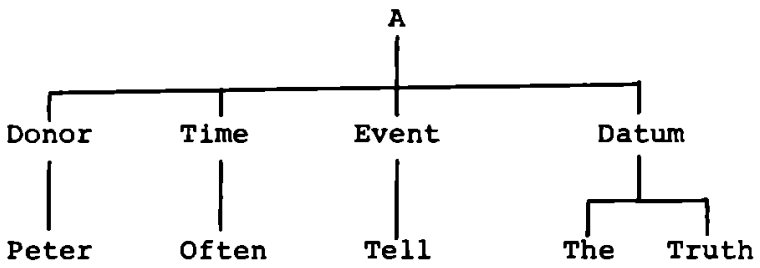
Peter often tells the truth.

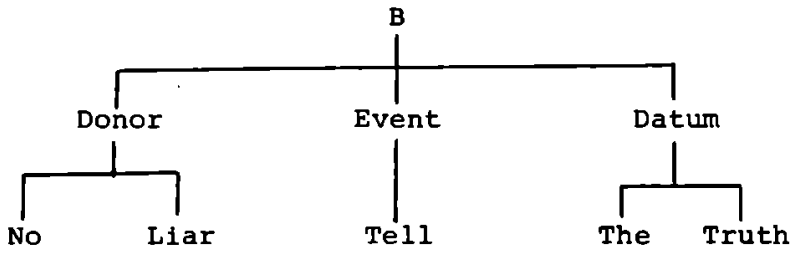
No liars tell the truth.

Den forudsatte leksikalske information kan være noget i denne retning

(Tell Donor { (Datum) (to Receiver) / (Receiver) Datum })

Syntakstræerne kan være





Resultatet ved algoritmen skulle så blive noget i denne retning

Event(A,Tell)

Donor(A,Peter)

Time(A.Often)

Datum(A,Truth)

← Event(y,Tell),Donor(y,x),Datum(y,Truth),Isa(x,Liar).

□

Eksempel 10

Peter eats garlic.

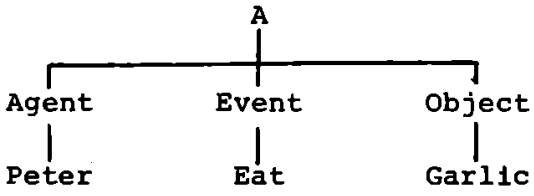
Everyone who eats garlic smells.

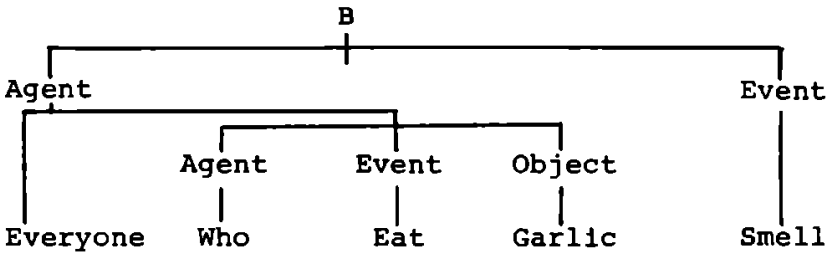
Den forudsatte leksikalske information kan være

(Eat Agent (Object))

(Smell Agent).

Syntakstræerne kan være





Resultatet bliver så noget i denne retning

Event(A,Eat)

Agent(A,Peter)

Object(A,Garlic)

Event(F(x,y),Smell) ← Event(y,Eat),Object(y,Garlic),
Agent(y,x)

Agent(F(x,y),x) ← Event(Y,Touch),Object(y,Garlic),
Agent(y,x).

□

Fordelene fra et datalogisk synspunkt er blandt andre af følgende art:

- Kompleksitetsteoretisk: Simple grammatikker (for eksempel af type LL1) giver effektiv (lineær) analyse, og grammatikker som "næsten" har disse egenskaber giver også forholdsvis effektiv analyse.
- Brugervenlighed: Den lingvistiske bruger skal kun udtrykke egentlige datalingvistiske relationer, endda i en notation som ligger tæt på de normalt anvendte.
- Systemkonstruktionsmæssigt: Som datastyret programmel benyttes den samme algoritme (inferensalgoritmen) hver gang. Man kan sige at der kun kræves en (ganske vist temmelig omstændelig) problemspecifikation. Såsnart problemet er logisk entydigt, kan programmellet overtage behandlingen. I denne forstand kan et sådant system betragtes som et problemorienteret system, og denne metode at konstruere systemer på kan betragtes som en problemorienteret programmeludviklingsmetode.

Vi arbejder på at benytte metoden her i forbindelse med datamatformidlet undervisning.

Vi er også ved at udvide systemer af denne art til at omfatte nogle intensionelle logiske systemer à la [Montague 74a] og [Koch 79].

4. Databaseforespørgsler

En metode til håndtering af databaseforespørgsler i humansproglige vendinger går ud på undervejs at oversætte forespørgslen til definitte klausuler.

Med samtlige oplysninger fra databasen formuleret som definitte klausuler ville svaret kunne genereres inden for det deduktive system i kraft af den indbyggede deduktionsmekanisme.

Fra et databasesynspunkt ville denne fremgangsmåde imidlertid være utilfredsstillende af effektivitetshensyn. Langt bedre ville det være at oversætte de definitte klausuler til et egentligt forespørgselssprog for et databasesystem. Som en realistisk mulighed har vi især undersøgt sproget QUEL tilhørende systemet INGRES. [Stonebraker et al 76].

Her eksemplificeres med en simpel forespørgsel til QUEL. En simpel database for "The Happy Valley Food Cooperative" består af tre databaserelationer

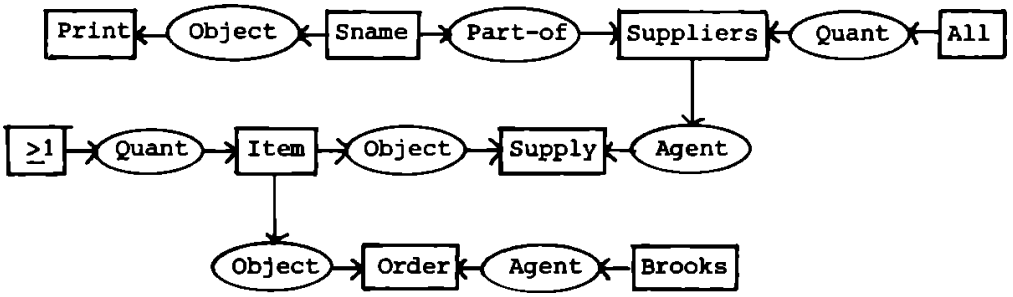
MEMBERS(NAME, ADDRESS, BALANCE)

ORDERS(NAME, ITEM, QUANTITY)

SUPPLIERS(SNAME, SADDRESS, ITEM, PRICE)

således at alle medlemmer har en adresse og en saldo, nogle medlemmer har bestilt forskellige varer i bestemte mængder, og nogle varer kan leveres af leverandører fra deres forretningsadresse til bestemte priser [Ullman 80].

Forespørgslen "Udskriv navnene på alle leverandører, som leverer mindst een vare bestilt af Brooks" kan i et logikprogrammeringssystem analyseres til en konceptuel graf [Sowa 76, 79, Pedersen 78] af følgende form



som igen af en relativt simpel algoritme kan transformeres videre til et logisk program af følgende form

```
Event(A(y),Print)       ← Isa(y,Suppliers),
                          Event(E(y),Order),
                          Agent(E(y),Brooks),
                          Object(E(y),Item(y))
Object(A(y),Sname(y)) ← Isa(y,Suppliers),
                          Event(E(y),Order),
                          Agent(E(y),Brooks),
                          Object(E(y),Item(y)).
```

Dette logiske program kan automatisk oversættes videre til et program i et egentligt databaseforespørgselssprog som QUEL med følgende resultat

```
RANGE OF y IS Suppliers
RANGE OF z IS Orders
RETRIEVE y . Sname
WHERE z . Name = Brooks  ∧
      z . Item = y . Item
```

som er et udmærket program til at besvare det stillede spørgsmål. [Jørgensen & Koch 81].

5. Udvidelser

Det simple databaseeksempel i foregående afsnit gik godt med brug af definitte klausuler. Men vi kunne også kigge på et eksempel som går ud på at undersøge om der findes medlemmer af Alpinistklubben som er bjergbestigere men ikke skisportsmænd, idet følgende vides:

Tony, Mike og John er i Alpinistklubben. Ethvert medlem af Alpinistklubben er skisportsmand eller bjergbestiger. Ingen bjergbestiger kan lide regn, og alle skisportsmænd elsker sne. Mike hader alt det Tony holder af og holder af alt det Tony hader. Tony kan godt lide regn og sne.

I sædvanlig prædikatkalkyle kan vi skrive:

$$\begin{aligned}
&\text{Tony} \in \text{Alpinists}\\
&\text{Mike} \in \text{Alpinists}\\
&\text{John} \in \text{Alpinists}\\
&\forall x \in \text{Alpinists}\,[\text{Skier}(x) \vee \text{Climber}(x)]\\
&\forall x\,[\text{Climber}(x) \rightarrow \text{Dislikes}(x,\text{Rain})]\\
&\forall x\,[\text{Skier}(x) \rightarrow \text{Likes}(x,\text{Snow})]
\end{aligned} \qquad (7)$$

$$\begin{aligned}
&\forall y\,[\text{Likes}(\text{Tony},y) \rightarrow \text{Dislikes}(\text{Mike},y)]\\
&\forall y\,[\text{Dislikes}(\text{Tony},y) \rightarrow \text{Likes}(\text{Mike},y)]\\
&\text{Likes}(\text{Tony},\text{Rain}) \wedge \text{Likes}(\text{Tony},\text{Snow})\\
&\forall x \in \text{Alpinists}\,[\text{Climber}(x) \wedge \neg\,\text{Skier}(x) \rightarrow \text{Print}(x)]
\end{aligned} \qquad (8)$$

(7) og (8) giver os problemer fordi negation er nødvendig.

De vanskeligheder vi her er løbet ind i beror væsentligt på at kun definitte klausuler er tilladte. Definitte klausuler er åbenbart for restriktiv en notation, og specielt at negation mangler synes at volde problemer. Så metoden her skulle helst generaliseres ud over definitte klausuler.

Den mest oplagte udvidelse er nok følgende:

Vi kan supplere hvert prædikatnavn P med et tilsvarende navn NP som symboliserer benægtelsen af prædikatet P. Altså vi forsøger så at sige at indføre negationen bag om ryggen på systemet (at programmere den ind i systemet). For at kunne udnytte sammenhængen mellem P og NP bliver vi så også nødt til at mangedoble hver regel

$$A_1(\underline{x}), \ldots, A_n(\underline{x}) \leftarrow B(\underline{x}), \ldots, B_m(\underline{x})$$

ved omskrivningen

$$A_1(\underline{x}) \vee \ldots \vee A_n(\underline{x}) \vee \neg B_1(\underline{x}) \vee \ldots \vee \neg B_m(\underline{x})$$

eller

$$\neg NA_1(\underline{x}) \vee \ldots \vee \neg NA_{i-1}(\underline{x}) \vee A_i(\underline{x}) \vee \neg NA_{i+1}(\underline{x}) \vee \ldots \vee \neg NA_n(\underline{x}) \vee \neg B_1(\underline{x}) \vee \ldots \vee \neg B_m(\underline{x})$$

eller

$$A_i(\underline{x}) \leftarrow B_1(\underline{x}), \ldots, B_m(\underline{x}), NA_1(\underline{x}), \ldots, NA_{i-1}(\underline{x}), NA_{i+1}(\underline{x}), \ldots, NA_n(\underline{x})$$

for hvert $i \in \{1, \ldots, n\}$.

Tilsvarende laves reglen

$$NB_j(\underline{x}) \leftarrow B_1(\underline{x}),\ldots,B_{j-1}(\underline{x}),B_{j+1}(\underline{x}),\ldots,B_m(\underline{x}),NA_1(\underline{x}),\ldots,NA_n(\underline{x})$$

for hvert $j \in \{1,\ldots, m\}$. Vi kan sige at vi udnævner hvert af de indgående prædikater til konklusion i en definit klausul. Endelig tilføjes reglen

$$\leftarrow p(\underline{x}),\ Np(\underline{x})$$

for hvert prædikat p .

Bruger vi denne metode på Alpinist-eksemplet fås følgende system som løser problemet:

```
Alpinist(Tony).
Alpinist(Mike).
Alpinist(John).
Dislikes(x,Rain) ← Climber(x).
Likes(x,Snow) ← Skier(x).        Nskier(x) ← Dislikes(x,Snow).
Dislikes(Mike,y) ← Likes(Tony,y).
Likes(Mike,y) ← Dislikes(Tony,y).
Likes(Tony,Rain).
Likes(Tony,Snow).
Skier(x) ← Alpinist(x), NClimber(x).
Climber(x) ← Alpinist(x), NSkier(x).
← Climber(x), NClimber(x).
← Skier(x), NSkier(x).
Print(x) ← Climber(x), NSkier(x).
```

Ulempen er at vi ender med at simulere traditionelle resolutionsstrategier med den deri liggende fare for ineffektivitet af såvel pladsmæssig som tidsmæssig art.

Altså den oplagte metode med at inkludere negationer i prædikatnavnene fører til en forholdsvis ineffektiv variant af resolutionsmetoden. I et kommende skrift [Koch 81] søges udviklet nogle alternative og mere begrænsede udvidelser af definitte klausuler, hvor disses effektive udførelse i det væsentlige synes bevaret.

6. Litteraturhenvisninger

Bowen, K.A. [1979]. Prolog, Proc. of the 1979 Annual Conf. ACM, Detroit, Michigan.

Charniak, E., Wilks, Y. (eds.) [1976]. Computational Semantics, pub. North-Holland.

Colmerauer, A. [1978]. Metamorphosis Grammars, in L. Bolc (ed.) Natural Language Communications with Computers, Springer, Berlin.

Fillmore, C. [1968]. The Case for Case, in Bach and Harms (eds.) Universals in Linguistics Theory, Holt, Rinehart and Winston, New York.

Jørgensen, P.H. [1980]. Inference and Semantics of Natural Language, master thesis, Institute of Datalogy, Copenhagen University.

Jørgensen, P.H., Koch, G. [1981]. Two New Methods of Natural Language Database Queries (in Danish), Proc. Norddata Conf., Copenhagen 1981, 227 - 232.

Koch, G. [1979]. Experimental Formalization of Danish. DIKU report 79/19 (in Danish), Institute of Datalogy, Copenhagen University.

Koch, G. [1980]. A Prolog Way of Representing Natural Language Fragments, DIKU report 80/16, Institute of Datalogy, Copenhagen University.

Koch, G. [1981]. Ulemper ved og udvidelser af definitte klausuler. Forthcoming DIKU report, Institute of Datalogy, Copenhagen University.

Kowalski, R. [1974]. Predicate Logic As Programming Language, Proc. IFIP 74, Stockholm.

Kowalski, R. [1979]. Logic for Problem Solving, New York, North-Holland, New York.

Mayoh, B.H. [1980]. The Meaning of Logical Programs, DAIMI PB-126, Aarhus University.

Montague, R. [1974a]. The Proper Treatment of Quantification in Ordinary English. [In Montague 74b].

Montague, R. [1974b]. Formal Philosophy, Yale University Press.

Nilsson, N.J. [1980]. Principles of Artificial Intelligence, Tioga Publ. Comp., California.

Parker-Rhodes, F. [1978]. Inferential Semantics, Harvester, Sussex, England.

Pedersen, G.S. [1978]. Conceptual Graphs I. DIKU report 78/9, Institute of Datalogy, Copenhagen University.

Pereira, F.C.N., Warren, D.H.D. [1980]. Definite Clause Grammars for Language Analysis - a Survey of the Formalism And a Comparison with Augmented Transition Networks. Artif. Intell. 13,3,231 - 278.

Schank, R. (ed.) [1975]. Conceptual Information Processing, North-Holland, Amsterdam.

Sowa, J.F. [1976]. Conceptual Graphs for a Database Interface. IBM Journ. Research. Devel. 20, 336 - 357.

Sowa, J.F. [1979]. Definitional Mechanism of Conceptual Graphs, in V. Claus et al. (eds.) Graph-grammars And Their Application to Computer Science And Biology, Springer, Berlin.

Ullman, J.D. [1980]. Principles of Database Systems. London.

Stonebraker, M. et al. [1976]. The Design and Implementation of INGRES. ACM Trans. on Database Systems 1,3, 189-222.