Tor Stålhane
RUNIT

LEKSIKALSK ANALYSE I MJUKE SYSTEM

0. INNLEDNING

Mjuke System er et RUNIT-prosjekt som blir finansiert av Norges Teknisk-Naturvitenskapelige Forskningsråd. Prosjektet har som mål å lette tilgange til EDB-systemer. Målgruppa består av to deler:

- folk som bare sporadisk bruker EDB
- folk som ofte bruker EDB, men som ofte bytter maskin. Dette gjelder særlig brukere som benytter datanett.

Hovedproblemet for begge disse brukergruppene er at de må huske en mengde nøkkelord og konvensjoner, så som plassering av punktum og komma o.l. Både nøkkelordene og konvensjonene varierer sterkt fra maskin til maskin.

Det er vår mening at dette problemet løses best ved å la brukerne benytte et subsett av norsk til kommunikasjon. Det systemet som skal stå mellom brukeren og EDB-systemet, har vi kalt et Mjukt System eller Mjukt Grensesnitt.

Dette systemet består av tre hoveddeler:

- leksikalanalysator
- syntaks- og semantikk-analysator
- system-grensesnitt

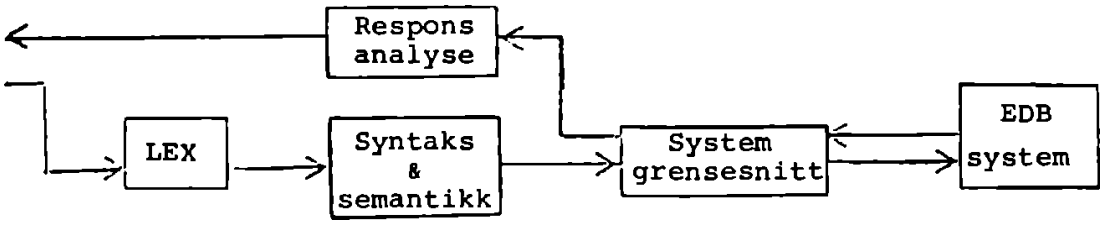


Fig. 1

Leksikalisk analyse i mjuke system
Tor Stålhane
Proceedings of NODALIDA 1981, pages 64-73

1. VOKABULAR

Systemet har for øyeblikket et relativt lite vokabular, bare ca. 150 ord. Ordutvalget er bestemt ut fra ei rekke protokollforsøk (1). Andre undersøkelser som er gjort har konkludert med at et generelt vokabular på ca. 400 ord pluss et fagvokabular på ca. 100 ord er tilstrekkelig for de fleste anvendelser (2).

1.1 Ordlister i Mjuke System

Vi har valgt følgende retningslinjer for ordliste-strukturen i prosjektet:

- notasjonen skal være enkel å forstå
- det skal være enkelt å legge inn nye ord og nye skrivemåter for ord som allerede er i ordlista
- endringer og tillegg skal kunne gjøres med en vanlig tekst-editor

Ut fra disse betingelsene har vi valgt følgende løsning:

Vi har en symbolsk fil som inneholder hele ordlista i direkte lesbar form. Lista inneholder to typer informasjon:

- generelle bøyningsmønster, som man kan referere til seinere i lista
- ord, med tilhørende bøyninger, eller med referanse til ei generell bøyning

Et generelt bøyningsmønster har forma

navn = <serie av bøyningsregler>.

Hver bøyningsregel har følgende form:

$$(\langle\text{ending}\rangle\{(\langle\text{utinfo}\rangle\langle\text{ordklasse}\rangle\langle\text{ordform}\rangle)\}_1^*)$$

Eksempel

```
SUBST_1=(- (- SUBST UBEST_SING))
        (EN(- SUBST BEST_SING))
        (ER(- SUBST UBEST_PL))
        (ENE(- SUBST BEST_PL)).
```

Dette bøyningsmønsteret kan nå knyttes til et ord i ordlista. F.eks. ordet BOKSTAV, som følger denne reglen

```
BOKSTAV SUBST_1.
```

Et ord defineres i ordlista på én av to måter:

- Ved bruk av de generelle bøyningsmønstrene, f.eks.

  BOKSTAV SUBST_1.

- Ved å definere et bøyningsmønster spesielt for dette ordet. Dette skjer ved å sette

  <ord> <serie av bøyningsregler>.

  F.eks.:

```
NAVN (-(- SUBST UBEST_SING)
        (- SUBST UBEST_PL))
     (ET(- SUBST BEST_SING))
     (ENE(- SUBST BEST_PL)).
```

Den sistnevnte metoden benyttes også for å behandle uregelmessige verb. Dette vises lettest ved et eksempel:

```
ER   (-(VÆR VB PRES)).
VÆR  (E(- VB INF)) (-(- VB IMPT))
     (T(- VB PERF)).
VAR  (-(VÆR VB IMPERF)).
```

Vi benytter her ut-info delen til å gi rett informasjon videre i systemet. Dette blir også brukt dersom vi ønsker å bytte ut et ord med et annet, f.eks.:

```
PRINTER (-(LINJESKRIVER SUBST UBEST_SING))....
```

1.2 Ordlistas lagerstruktur

Internt i leksikal-analysatoren blir hvert ord lagra på følgende måte:

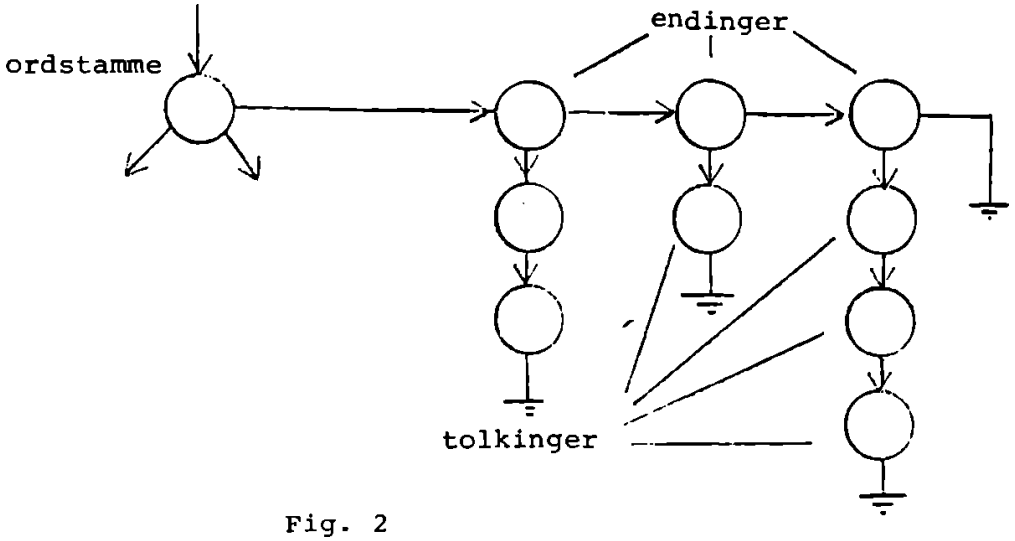


Fig. 2

Som ordstamme har vi valgt å bruker imperativsformen for verb og ubestemt form éntall for substantiver. Dette er valgt ut fra hensynet til en mest mulig kompakt struktur. Hadde vi f.eks. valgt infinitivsformen for verb ville vi vært nødt til ha ha ekstra innslag i lista for imperativsformen.

Eksempel på lagring:

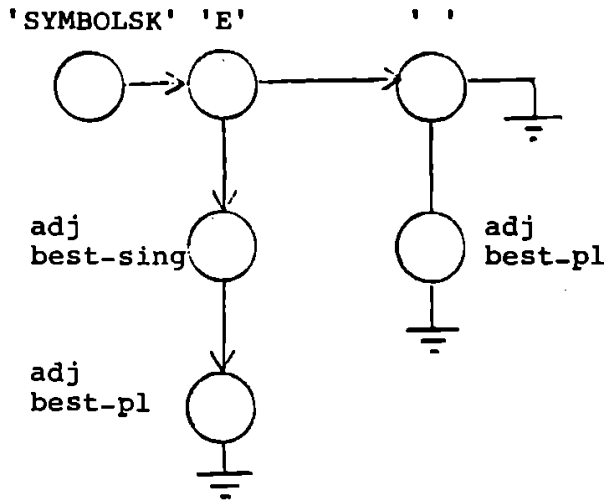


Fig. 3

Ordlista er delt opp i et sett av balanserte trær. Ett tre består av alle ord som begynner på 'A', ett tre består av alle ord som begynner på 'B' osv., til 'Å'.

Hvert tre er ordna etter tre kriterier:

- orda er sortert etter lengde før de settes inn i treet.
- hver ordstamme settes inn ved å leite fram et tomt lenkefelt på følgende måte:

  Viss ordet vi sammenligner med er lenger ute i alfabetet, følger vi høyre lenke, ellers følger vi den venstre. Dette fortsetter vi med til den lenka vi vil følge, er tom. Det nye ordet blir så hengt på der.

- treet er balansert for hver ordlengde (3)

Dette er gjort for å forenkle søkinga etter et ord og for å unngå å måtte gå igjennom treet flere ganger. Ved å sørge for at treet er balansert vil vi trenge gjennomsnittlig ca. $\frac{1}{3}\log_2(N)+1$ sammenligninger der N er antall ord i treet.

Uten balansering vil vi kunne risikere å måtte foreta N/2 sammenligninger. Det ferdige treet vil kunne se f.eks. slik ut

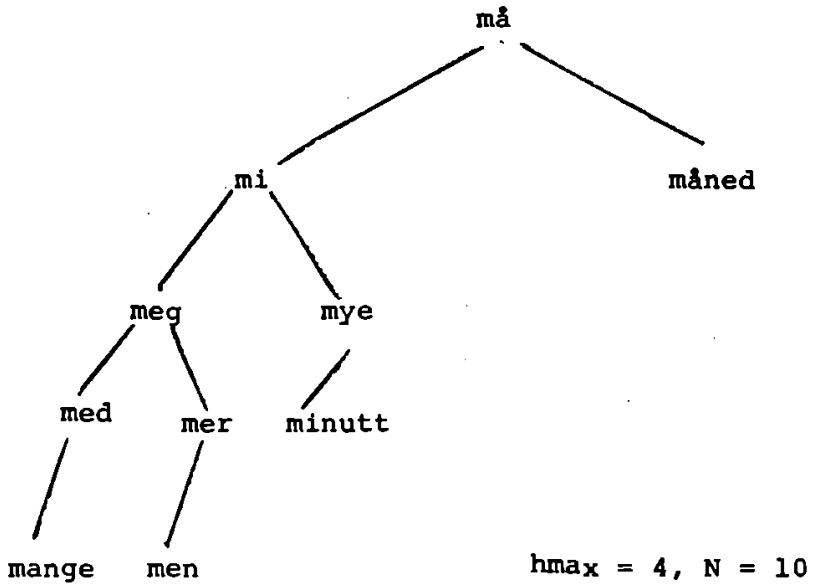


Fig. 4

1.3 Søkemetode

For å finne ut om et ord er i ordlista benytter vi følgende metode (4):

- Første bokstav viser hvilket tre vi skal søke i. Lengda av første ordstamme i treet viser hvor stor del av ordet vi må begynne med i søkinga. Resten av ordet blir behandla som ei ending.

- Deretter leter vi i treet til vi finner den aktuelle ordstammen eller vi finner en stamme som er lenger enn den vi har for øyeblikket.

  Dersom vi har kommet fram til en lengere ordstamme enn den vi søker med, forlenger vi stammen med neste bokstav i søkeordet. Endinga blir da forkorta tilsvarende.

  Viss det ikke er flere bokstaver igjen, finnes ikke søkeordet, ellers fortsetter vi søkinga til vi finner ordet, eller vi når slutten på treet.

- Når vi har funnet en ordstamme som passer, må vi sjekke om resten av ordet (endinga) finnes som ei lovlig ending til denne stammen. Dersom det er tilfelle, er alt ok og vi kan returnere den informasjonen vi har funnet.

  Dersom resten av ordet ikke er ei lovlig ending, må vi forlenge stammen og fortsette søkinga.

Et par enkle eksempler vil vise hvordan metoden fungerer i praksis:

1)

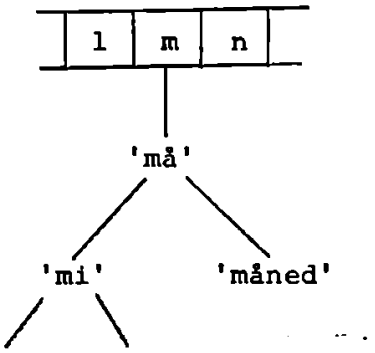


Fig. 5

Leter etter ordet 'måned'.

. begynner med 'må' - 'ned'

. finner 'må' først i treet, men denne stammen har ikke 'ned' som tillatt ending

. utvider til 'mån' - 'ed', men treet har ingen ordstammer langs denne greina med tre bokstaver. Det samme gjelder for 'måne'-- 'd'.

. først når vi utvider ordstammen til 'måned' - ' ' finner vi ordet og kan returnere

'måned subst ubest_sing'

2)

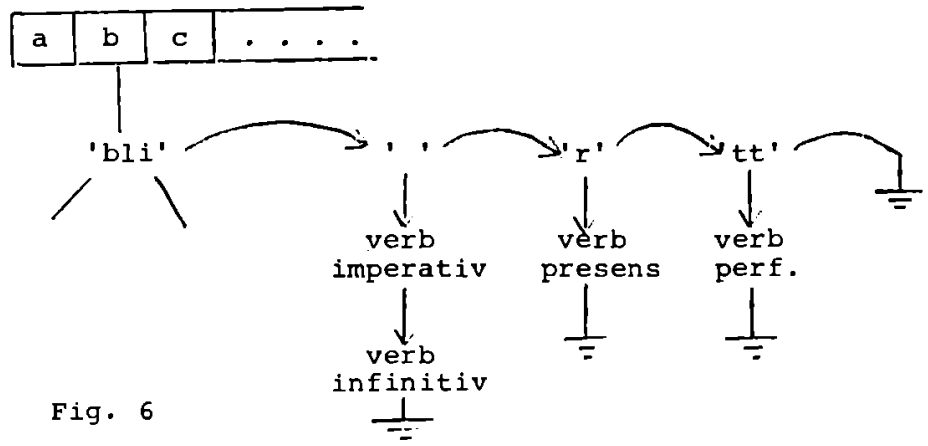


Fig. 6

Leter etter ordet 'blitt'.

Siden første ordstammen har tre bokstaver, starter vi med 'bli'-'tt'. Vi finner at 'bli' er en lovlig ordstamme og søker langs kjeda av lovlige endinger med 'tt'. Som vi ser i fig. 6, finner vi denne endinga sist i kjeda og returnerer derfor

'blitt bli verb perf.'

Som tidligere sagt under 1.1 blir uregelmessige verb og substantiver knytta til stammen via den delen vi kaller ut-info. Dersom vi f.eks. skulle leite eller 'blei' istedenfor 'blitt' ville vi ha funnet dette som et eget innslag i tabellen og fått returnert

'blei bli verb imperf'

2. HANDTERING AV STAVEFEIL

Eksperimenter og innsamla erfaringsdata indikerer at den alt overveiende delen av skrive/stave-feil tilhører en av følgende kategorier (5):

- én ekstra bokstav
- én manglende bokstav
- én feil bokstav

I tillegg har det vist seg at feilraten for alle disse feiltypene er langt større "inne i" ordet enn i starten og slutten. Disse forholda har gjort at vi har valgt følgende metode for handtering av stavefeil (6):

- for hvert ord som er lagra i ordlista blir det laga en bitvektor som forteller hvilke bostaver som finnes i ordet.

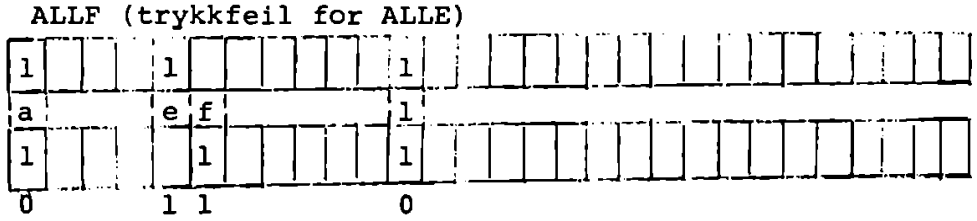
  Eks.

  ALLE

  I tillegg tar vi vare på ordlengden. Den samme informasjonen blir generert for et ord som ikke er blitt funnet i ordlista etter et vanlig gjennomløp.

- for hvert ord i det treet vi gjennomløper, sjekker vi så differansen mellom ordet i ordlista og det ordet vi tror er et feilstava ord. Denne differansen kommer fram ved å sammenligne de to bit-vektorene og summere alle bitene som ikke finnes i begge orda. Til dette tallet legger vi så absoluttverdien av forskjellen i ordlengdene.

  Eks.:



  Differansen blir 2, idet orda har samme lengde.

- for at et ord skal bli godtatt som en feilstaving av et annet, må differansen være mindre enn en forhandssatt terskel, dmax.

  Det er lett å forvisse seg om at de tre vanlige formene for skrivefeil alle vil gi en differanse på maksimum 2. Vi har derfor valgt dmax lik 2, unntatt når ordet har færre enn fire bokstaver. I de tilfellene benytter vi dmax lik 1.

  Metoden har så langt vist seg relativt tilfredsstillende. Det eneste problemet har vært at metoden ofte kommer opp med alt for mange tolkningsalternativer. Dette problemet vil kunne øke etter hvert som vi får flere ord i ordlista.

3. KOBLING TIL PROLOG

Den syntaktiske og semantiske analysen av inndata foregår i et PROLOG-program (7).

PROLOG benytter listestruktur til å representere inn- og ut-data. I Mjuke System kommer data inn som fri-format tekst. Leksikalanalysatoren deler opp teksten i ord, tall og skille-tegn og henger på en del informasjon om hver tekstenhet. Denne informasjonen har følgende format:

$$(\langle\text{inn-data}\rangle \left\{(\langle\text{indikator}\rangle\ \langle\text{stamme}\rangle\langle\text{tolking}\rangle)\right\}_1^*)$$

indikator kan være:

! : ordet er funnet i ordlista

? : ordet er ikke funnet i ordlista, men vi har funnet ei rimelig tolking ut fra feilstavingsalgoritma

tolking ::= <ordklasse> <bøyning>

Dersom bøyning ikke er relevant, blir den erstatta med '-'.

Noen eksempler vil vise hvordan denne blir brukt:

"Hvilke filer har jeg?"

vil gi:

```
((HVILKE (! HVILKE PRON   PL       ))
 (FILER  (! FIL    SUBST  UBEST-PL ))
 (HAR    (! HA     VB     PRES     ))
 (JEG    (! JEG    PRON     -      ))
 (?      (! "?"    TEGN   SPØRSMÅL ))
```

"Filler" (trykkfeil for Filer) vil gi:

```
 (FILLER (?  FILLER IDENT    -       )
         (?  FLER   ADJ     PL       )
         (?  FINN   VB      PRES     )
         (?  FEIL   SUBST   BEST-SING)
         (?  FIL    SUBST   BEST-PL  )
         (?  FIL    SUBST   UBEST-PL ))
```

4. OPPSUMMERING

Det er vår mening at den implementerte løsninga tilfredsstiller alle viktige krav til en leksikal-analysator for et naturlig språk:

. Det er lett å legge inn nye ord
. Det er lett å legge på nye tolkninger av ord som allerede finnes
. Det er lett å legge inn nye former og skrivemåter
. Systemet er rimelig tolerant overfor de vanligste skrivefeil

Det videre arbeidet med leksikalanalysen i Mjuke System vil konsentrere seg om to områder:

- utvidelse av vokabularet etter hvert som vi får mer bruker-erfaring

- forsøk på å dele vokabularet i én generell norsk-del som er brukeruavhengig, og en fagterm-del som må byttes ut når systemet skal handtere andre fagfelt/universer.

5. REFERANSER

(1) Amble, Tore
Mjuke System, Arbeidsnotat nr. 2, RUNIT 1981

(2) Kelly, Michael
Limited Vocabulary Natural Language Dialogue
Int. J. Man-Machine Studies, 1977 no. 9

(3) Wirth, Niklaus
Algorithms + Datastructures = Programs
Prentice-Hall, Englewood Cliffs NJ, 1976

(4) Stålhane, Tor
Mjuke System, Arbeidsnotat nr. 5, RUNIT 1981

(5) Morgan, H.L.
Spelling Error Correction in Systems Programs
Comm ACM 13, 1970

(6) Tenczar, P.
Spelling, Word and Concept Recognition
University of Illinois, Urbana, Ill. 1972

(7) Amble, Tore
Introduction to Logic Programming
RUNIT, 1981