Forelesning 14: Automatisk bevissøk IV — matriser og koblingskalkyle

Christian Mahesh Hansen - 22. mai 2006

1 Automatisk bevissøk IV

1.1 Introduksjon

Bevissøk med koblinger

- Vi har til nå sett på forskjellige varianter av sekventkalkyle:
  - LK for *klassisk utsagnslogikk*,
  - LJ for *intuisjonistisk utsagnslogikk*,
  - ensidig sekventkalkyle,
  - grunn LK og
  - fri-variabel LK for *klassisk førsteordens logikk*.
- Alle disse kalkylene har to svakheter når de skal implementeres med en automatisk søkealgoritme:
  - *Redundans* – gjennom formelkopiering kan vi få mange forekomster av én og samme formel.
  - *Ingen relevanssjekk* – siden det kun tas hensyn til toppkonnektiv ved formelanalyse kan vi risikere å utvide formler som ikke er relevante for å lukke utledningen.
- Vi skal i denne forelesningen se på koblingskalkylen, som ikke har disse problemene.

Redundans i LK-utledninger

$$
\dfrac{
  \dfrac{\genfrac{}{}{0pt}{}{\times}{Q_2 \to R_1, P_2 \vdash R_2, P_1}}{Q_2 \to R_1 \vdash P_1, P_2 \to R_2}
  \qquad
  \dfrac{\genfrac{}{}{0pt}{}{\times}{Q_1 \vdash Q_2, P_2 \to R_2} \qquad \dfrac{\genfrac{}{}{0pt}{}{\times}{Q_1, R_1, P_2 \vdash R_2}}{Q_1, R_1 \vdash P_2 \to R_2}}{Q_2 \to R_1, Q_1 \vdash P_2 \to R_2}
}{P_1 \to Q_1, Q_2 \to R_1 \vdash P_2 \to R_2}
$$

- I rotsekventen forekommer både P, Q og R to ganger. Vi bruker $_1$ og $_2$ til å skille forekomstene fra hverandre.
- Siden P_2 og P_1 i venstre løvsekvent er forekomster av P, kan vi lukke med disse. Tilsvarende for de andre løvsekventene.
- En LK-utledning kan inneholde mange kopier av en (del)formel i rotsekventen. I utledningen over: 3 kopier av $Q_2 \to R_1$, 4 kopier av $P_2 \to R_2$, 2 kopier av P_1, 6 kopier av P_2, osv.
- Vi har derfor *redundans* i LK-utledninger.

Ingen relevanssjekk

$$\dfrac{\dfrac{\times}{(Q \to R) \lor (R \to Q), P \vdash P} \qquad \dfrac{\times}{(Q \to R) \lor (R \to Q), P \vdash P}}{(Q \to R) \lor (R \to Q), P \lor P \vdash P}$$

- I rotsekventen over er det to muligheter for utvidelse: $(Q \to R) \lor (R \to Q)$ eller $P \lor P$
  - Hvis vi velger $(Q \to R) \lor (R \to Q)$, vil vi gjøre (en eller flere) utvidelser som ikke bidrar til å lukke løvsekventene.
  - Velger vi derimot $P \lor P$, vil vi kunne lukke direkte.
- Det er de *atomære delformlene* til en formel som er med på å lukke løvsekventene.
- I sekventkalkyle ser vi imidlertid kun på *toppkonnektivene* for å velge hvilken formel vi skal utvide.
- Vi kan derfor risikere å utvide formler som er *irrelevante* m.h.p. å lukke utledningen.

Matriser og koblingskalkyle

- Bevisbarhet av en formel kan defineres som en egenskap ved formelen direkte, istedenfor via utledninger som i sekventkalkyle.
  - Vi kan representere en formel todimensjonalt ved en *matrise* bestående av de atomære delformlene.
  - Gyldighet defineres så som en egenskap ved stiene gjennom matrisen.
  - Hver sti må inneholde et komplementært par av atomer – kalt en *kobling*.
- *Koblingskalkyle* er basert på matrisekarakteristikken av gyldighet og utnytter at samme kobling kan forekomme på flere stier gjennom en matrise.
- Vi skal begrense oss til å se på koblingskalkyle for utsagnslogikk i disjunktiv normalform.
- Koblingskalkyle er imidlertid ikke begrenset til normalform, og finnes for mange forskjellige logikker – inkludert de vi har sett på i kurset.

1.2 Matriser

Disjunktiv normalform

- I ukeoppgavene har vi sett på normalformer.
- En *literal* er en atomær formel eller negasjonen av en atomær formel:
  - $P, Q, R, \ldots$ er *positive* literaler og $\neg P, \neg Q, \neg R, \ldots$ er *negative* literaler.
- En *generalisert konjunksjon* er en formel på formen $(\varphi_1 \land \ldots \land \varphi_n)$ der hver φ_i er en formel.
- En *generalisert disjunksjon* er en formel på formen $(\varphi_1 \lor \ldots \lor \varphi_n)$ der hver φ_i er en formel.

Definisjon 1.1 (Disjunktiv normalform)**.** *En formel er på* disjunktiv normalform *(DNF) hvis den er en (generalisert) disjunksjon av en eller flere (generaliserte) konjunksjoner av en eller flere literaler.*

Disjunktiv normalform

Hvilke formler er på DNF?

- $P \quad \surd$
- $P \vee Q \quad \surd$
- $(P \vee Q) \wedge R$ Nei, venstre konjunkt er en disjunksjon.
- $(\neg P \wedge Q) \vee (\neg Q \wedge P) \quad \surd$
- $(P \to Q) \vee Q \vee \neg P$ Nei, venstre konjunkt er en implikasjon.
- $(\neg P \wedge Q) \vee R \vee (\neg R \wedge P) \vee (\neg P \wedge Q \wedge \neg R) \quad \surd$
- $\neg P \wedge Q \wedge \neg R \quad \surd$
- Enhver literal er på DNF.
- Enhver disjunksjon av literaler er på DNF.
- Enhver konjunksjon av literaler er på DNF.

Transformasjon til DNF

- Vi har i ukeoppgavene sett at enhver utsagnslogisk formel kan transformeres til en *ekvivalent* formel på DNF.
- Husk: to formler er *ekvivalente* hvis de oppfylles av nøyaktig de samme valuasjonene/modellene.
- Eksempel:

$$\begin{aligned}
(P \wedge (P \to Q)) \to Q &\Leftrightarrow \neg(P \wedge (P \to Q)) \vee Q \\
&\Leftrightarrow \neg P \vee \neg(P \to Q) \vee Q \\
&\Leftrightarrow \neg P \vee \neg(\neg P \vee Q) \vee Q \\
&\Leftrightarrow \neg P \vee (\neg\neg P \wedge \neg Q) \vee Q \\
&\Leftrightarrow \neg P \vee (P \wedge \neg Q) \vee Q
\end{aligned}$$

Definisjon 1.2 (Matrise)**.**

- *En* klausul *er en endelig mengde literaler.*
- *En* matrise *er en endelig mengde klausuler.*

Eksempel (klausuler):

- $\{P\}$
- $\{P, \neg P, Q\}$
- $\{\neg Q, R, P\}$
- $\{\}$

Eksempel (matriser):

- $\{\{P\}, \{\neg P\}\}$
- $\{\{Q\}, \{\neg P, R\}, \{\neg R, P, \neg Q\}\}$
- $\{\}$
- $\{\{\}\}$

- En klausul er
  - *positiv* hvis den bare inneholder positive literaler, og
  - *negativ* hvis den bare inneholder negative literaler.

Grafisk matrisenotasjon

- Matriser blir lettere å lese med grafisk notasjon.
- Literalene i en klausul skrives vertikalt under hverandre.
- Klausulene skrives ved siden av hverandre horisontalt som søyler.
- Matrisen $\{\{\neg P\}, \{P, \neg Q\}, \{Q\}\}$ skrives

$$\left|\begin{array}{ccc} & P & Q \\ \neg P & \neg Q & \end{array}\right|$$

Definisjon 1.3 (Semantikk for matriser)**.** *La v være en boolsk valuasjon.*

- *For klausuler: $v \models \{L_1, \ldots, L_n\}$ hvis og bare hvis $v \models L_i$ for alle L_i.*
- *For matriser: $v \models \{K_1, \ldots, K_n\}$ hvis og bare hvis $v \models K_i$ for en K_i.*

Eksempel. *La v være slik at $v \models P$, $v \not\models Q$ og $v \models R$.*

- *$v \models \{\{P\}, \{\neg P\}\}$?* $\surd$
- *$v \models \{\{Q\}, \{\neg P, R\}, \{\neg R, P, \neg Q\}\}$? Nei, v oppfyller ingen klausuler.*
- *$v \models \{\}$ der $\{\}$ er en tom matrise? Nei, v oppfyller ingen klausuler i $\{\}$.*
- *$v \models \{\{\}\}$ (matrisen som kun inneholder en tom klausul)?* $\surd$
  ($v \models \{\} \in \{\{\}\}$ siden alle valuasjon oppfyller en tom klausul.)

Falsifiserbarhet av matriser

v	$M = \{K_1, \ldots, K_n\}$	$K = \{L_1, \ldots, L_m\}$
oppfyller	$v \models K_i$ for *en* K_i	$v \models L_i$ for *alle* L_i
falsifiserer	$v \not\models K_i$ for *alle* K_i	$v \not\models L_i$ for *en* L_i

- En boolsk valuasjon v *falsifiserer*
  - en klausul i M hvis v falsifiserer en av literalene i klausulen
  - alle klausulene i M hvis v falsifiserer en literal i hver klausul
- For hver klausul K_i har vi $|K_i|$ valg av literaler å falsifisere.
- Vi får maksimalt $|K_1| \times \ldots \times |K_n|$ måter å falsifisere M på.

DNF-formler som matriser

- En formel på DNF kan sees på som en matrise der klausulene tilsvarer disjunktene i formelen.
- Eksempel: formelen

  $$\neg P \lor (P \land \neg Q) \lor Q$$

  tilsvarer matrisen

  $$\{\{\neg P\}, \{P, \neg Q\}, \{Q\}\}.$$

Stier

Definisjon 1.4 (Sti)**.** *La M være en matrise.*

- *En* sti *gjennom M er en mengde som inneholder nøyaktig én literal fra hver klausul i M.*
- *En sti gjennom M er* partiell *hvis den mangler literaler fra én eller flere klausuler i M.*

Eksempel.

$$\left|\begin{array}{ccc} & P & Q \\ \neg P & \neg Q & \end{array}\right|$$

- *Stier: $\{\neg P, P, Q\}$ og $\{\neg P, \neg Q, Q\}$.*
- *Partieller stier: $\{\neg P\}$, $\{\neg P, P\}$, $\{\neg P, \neg Q\}$, $\{P\}$, $\{P, Q\}$, $\{Q\}$, $\{Q, \neg Q\}$ og $\{Q, \neg P\}$.*

Hver sti gjennom en matrise representerer en mulig falsifikasjon!

Koblinger

Definisjon 1.5 (Koblinger)**.** *La M være en matrise. En* kobling *i M er en partiell sti gjennom M på formen $\{A, \neg A\}$ der A er en atomær formel.*

Eksempel.

$$\left|\begin{array}{ccc} & P & Q \\ \neg P & \neg Q & \end{array}\right|$$

- *Koblinger: $\{\neg P, P\}$ og $\{\neg Q, Q\}$.*
- *Vi markerer sammenkoblede literaler med en bue i den grafiske matrisenotasjonen.*

Åpne og lukkede stier

- En kobling $\{P, \neg P\}$ er *ikke* falsifiserbar:
  - en boolsk valuasjon kan ikke falsifisere både P og $\neg P$ samtidig!
- Derfor vil en sti som inneholder en kobling, *ikke* være falsifiserbar.
- Dersom alle stiene gjennom en matrise inneholder en kobling, vil matrisen ikke være falsifiserbar – og dermed må formelen den representerer være gyldig.
- Vi sier at en (partiell) sti i en matrise er
  - *åpen* hvis den *ikke* inneholder noen kobling, og
  - *lukket* hvis den inneholder en kobling.

Matriser vs. LK**-utledninger**

- Stiene gjennom matrisen til en formel F tilsvarer løvsekventene vi får hvis vi gjør en *maksimal* LK-utledning for sekventen $\vdash F$:
  - Negative literaler er antecedentformler, og
  - positive literaler er succedentformler.

$$\left| \begin{array}{ccc} & P & Q \\ \neg P & \neg Q & \end{array} \right| \qquad\qquad \dfrac{\dfrac{P \vdash P, Q \qquad \dfrac{P, Q \vdash Q}{P \vdash \neg Q, Q}}{P \vdash P \land \neg Q, Q}}{\vdash \neg P, P \land \neg Q, Q}$$

- Lukkede stier gjennom matriser tilsvarer aksiomer i LK-utledninger.

Matrisekarakterisering av gyldighet

Teorem 1.1. *En formel F på DNF er gyldig hvis og bare hvis enhver sti gjennom matrisen til F inneholder en kobling.*

Eksempel.

- *Formelen $(P \land (P \to Q) \land (Q \to R)) \to R$ er gyldig.*
- *På DNF får vi formelen $\neg P \lor (P \land \neg Q) \lor (Q \land \neg R) \lor R$.*

$$\left| \begin{array}{cccc} & P & Q & R \\ \neg P & \neg Q & \neg R & \end{array} \right|$$

Koblinger: $\{\neg P, P\}$, $\{\neg Q, Q\}$ *og* $\{\neg R, R\}$.

Stier: $\{\neg P, P, Q, R\}$, $\{\neg P, P, \neg R, R\}$, $\{\neg P, \neg Q, Q, R\}$ *og* $\{\neg P, \neg Q, \neg R, R\}$.

- *Alle stiene inneholder en kobling.*

1.3 Koblingskalkyle

- Matrisekarakteriseringen av gyldighet gir oss muligheten til å avgjøre bevisbarhet ved å sjekke at alle stier inneholder en kobling.
- En første tilnærming vil være å liste opp alle stiene gjennom en matrise og sjekke hver av dem for koblinger.
- Det er imidlertid slik at én kobling kan forekomme på flere stier gjennom en matrise.

Eksempel.

$$\left|\begin{array}{cccc} M_1 & P & \neg P & M_2 \end{array}\right|$$

Uansett hvordan M_1 og M_2 ser ut vil enhver sti gå gjennom koblingen $\{P, \neg P\}$.

- Det er derfor en god idé å fokusere på koblinger istedenfor stier.
- Vi skal vise grunnidéene i koblingskalkylen ved et eksempel.

Startsteget

$$\left|\begin{array}{cccc} & \neg P & \neg Q & \\ P & Q & \neg P & R \\ {\scriptstyle\nearrow}\ {\scriptstyle\uparrow} & & & \end{array}\right|$$

- Vi starter med å velge en *startklausul*. Denne velges alltid blant de *positive* klausulene, som rettferdiggjøres av følgende lemma:

Lemma 1.1. *En matrise for en gyldig formel inneholder minst én positiv klausul.*

- Vi har to valg av startklausul: $\{P\}$ og $\{R\}$.
- Vi velger $\{P\}$ og markerer denne med $\uparrow$ under klausulen, og markerer alle literalene i startklausulen med $\nearrow$.
- Senere kommer vi tilbake til hva som skjer hvis vi velger $\{R\}$.

Utvidelsessteget I

$$\left|\begin{array}{cccc} & \neg P & \neg Q & \\ \boxed{\mathrm{P}} & Q & \neg P & R \\ {\scriptstyle\uparrow} & & & \end{array}\right|$$

- Vi kaller klausulen som er markert med $\uparrow$, for *aktiv klausul*.
- Hvis en literal i aktiv klausul er markert med $\nearrow$ må vi sjekke alle stier som inneholder literalen.
- I dette tilfellet har vi bare ett valg: P.
- Vi markerer P med en ramme for å indikere at literalen er en del av den stien vi for øyeblikket undersøker – den *aktive stien*.
- Samtidig fjerner vi $\nearrow$-symbolet fra P.

Utvidelsessteget II

$$\left|\begin{array}{cccc} & \neg P. & \neg Q & \\ \boxed{\mathrm{P}} & Q & \neg P & R \\ & {\nearrow}\,{\uparrow} & & \end{array}\right|$$

- Vi utvider den aktive stien ved å koble P med en komplementær literal i en av de andre klausulene.
- Vi har to valg: $\{\neg P, Q\}$ og $\{\neg Q, \neg P\}$.
- Vi velger den første klausulen og markerer de sammenkoblede literalene med en bue.
  - Det kan hende vi må tilbake for å sjekke det andre alternativet senere!
- Alle stier som springer ut fra den sammenkoblede $\neg P$ vil være lukkede på grunn av koblingen $\{P, \neg P\}$. Dette markeres med '.' etter $\neg P$.
- Den sammenkoblede klausulen settes aktiv og de resterende literalene på den markeres med $\nearrow$.

Utvidelsessteget – oppsummering

$$\left|\begin{array}{cccc} & \neg P & \neg Q & \\ P & Q & \neg P & R \\ {\nearrow}\,{\uparrow} & & & \end{array}\right| \rightsquigarrow \left|\begin{array}{cccc} & \neg P. & \neg Q & \\ \boxed{\mathrm{P}} & Q & \neg P & R \\ & {\nearrow}\,{\uparrow} & & \end{array}\right|$$

$\uparrow$ markerer aktiv klausul
$\boxed{L}$ markerer literaler på den aktive stien
. markerer lukkede partielle stier
$\nearrow$ markerer literaler i åpne partielle stier

1. Velg en literal L markert med $\nearrow$ i den aktive klausulen.
2. Bytt ut $\nearrow$ med en boks rundt L. Velg en L-kobling.
   - Hvis det er flere alternativer, ta vare på dem.
3. Marker den koblede literalen med '.'
4. Marker de resterende literalene i den koblede klausulen med $\nearrow$.
5. Flytt $\uparrow$ til den sammenkoblede klausulen.

Vi foretar nok et utvidelsessteg

$$\left|\begin{array}{cccc} & \neg P. & \neg Q & \\ \boxed{\mathrm{P}} & Q & \neg P & R \\ & {\nearrow}\,{\uparrow} & & \end{array}\right| \rightsquigarrow \left|\begin{array}{cccc} & \neg P. & \neg Q. & \\ \boxed{\mathrm{P}} & \boxed{\mathrm{Q}} & \neg P & R \\ & & {\nearrow}\,{\uparrow} & \end{array}\right|$$

$\uparrow$ markerer aktiv klausul
$\boxed{L}$ markerer literaler på den aktive stien
. markerer lukkede partielle stier
$\nearrow$ markerer literaler i åpne partielle stier

1. Vi velger literalen Q i den aktive klausulen.
2. Vi bytter ut $\nearrow$ med en boks rundt Q. Vi har bare ett alternativ til kobling: $\neg Q$.
3. Markerer den koblede literalen med '.'
4. Markerer de resterende literalene i den koblede klausulen med $\nearrow$.
5. Flytt $\uparrow$ til den sammenkoblede klausulen.

Reduksjonssteget

$\uparrow$ markerer aktiv klausul

. markerer lukkede partielle stier

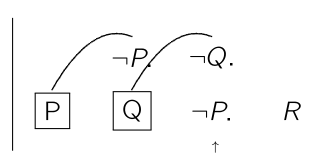
$\boxed{L}$ markerer literaler på den aktive stien

$\nearrow$ markerer literaler i åpne partielle stier

- I situasjonen til venstre finnes det ingen literaler å koble $\neg P$ med.
- Det finnes imidlertid en komplementær literal i den aktive stien: P.
- Vi kan derfor foreta et *reduksjonssteg*:

1. Blant literalene som er merket med $\nearrow$ i den aktive klausulen, velg en L som er komplementær med en literal på den aktive stien.
2. Fjern $\nearrow$ fra L og marker den med '.'

Fullført søk – suksess



- I situasjonen over er alle literaler i aktiv klausul markert med '.', dvs. at alle stier som fortsetter ut fra denne klausulen inneholder koblinger.
- I tillegg er ingen literaler i klausuler på den aktive stien merket med $\nearrow$, dvs. at vi ikke har noen partielle åpne stier igjen å sjekke.
- Tilstanden er et vitne på at søket er *fullført med suksess* – alle stier gjennom matrisen inneholder koblinger.

Kalkyle vs. søkealgoritme

- Koblingskalkylen består av reglene *start*, *utvidelse* og *reduksjon*.
- Reglene definerer et sett *stisjekkingstilstander*.
- I tillegg har vi en beskrivelse av hvilke tilstander som representerer *suksess* i stisjekkingen.
- En søkealgoritme for koblingskalkylen må spesifisere en *rekkefølge* å gjøre reglene i.
- Vi skal nøye oss med å presentere noen viktige poenger m.h.p. implementasjon av en søkealgoritme.
- Til slutt skal vi ta en titt på en Prolog-implementasjon av koblingskalkylen.

Sjekke alle åpne partielle stier

$$\left|\begin{array}{cccc} \boxed{P} & \neg P. & \neg Q. & \\ R & \boxed{Q} & \neg P. & R \\ \nearrow & & \uparrow & \end{array}\right|$$

- Vi ser på suksesstilstanden med matrisen fra eksempelet, men legger til R i første klausul. Den nye matrisen inneholder flere stier uten koblinger, f.eks. $\{R, Q, \neg P, R\}$.
- Tilstanden over er *ikke* en suksesstilstand, siden vi har en partiell åpen sti: den nye literalen R er merket med $\nearrow$.
- Vi må gå tilbake og se hva som skjer hvis vi velger R som del av den aktive stien istedenfor P i venstre klausul.
- Vi får en låst tilstand, siden R ikke kan kobles med noen literaler i matrisen.
- Vi må altså sjekke alle åpne partielle stier (literaler merket med $\nearrow$) før vi kan konludere med suksess.

Backtracking over valg av startklausul

$$\left|\begin{array}{cccc} & \neg P & \neg Q & \\ P & Q & \neg P & R \\ & & & \nearrow\ \uparrow \end{array}\right|$$

- La oss anta at vi velger $\{R\}$ som startklausul istedenfor $\{P\}$.
- De andre klausulene inneholder ikke literaler som er komplementære med R, så vi kan ikke utføre utvidelsessteg.
- Den aktive stien er tom, så vi kan ikke utføre reduksjonssteg.
- Vi har literaler merket med $\nearrow$, dvs. at vi fremdeles har åpne partielle stier igjen å sjekke.
- Eksempelet viser at vi må *backtracke over valg av startklausul* for å få en komplett søkealgoritme.

Backtracking over valg av kobling

$$\left|\begin{array}{cccc} & \neg P & \neg Q & \neg P. \\ \boxed{\mathsf{P}} & Q & \neg P & R \end{array}\right|$$

- La oss anta at vi legger til $\neg P$ i den høyre klausulen i matrisen fra eksempelet.
- Alle stier i den nye matrisen er lukkede. (Sjekk!)
- Vi gjør utvidelsessteg ved å koble P i startklausulen med den nye $\neg P$.
- Vi har igjen kommet til en låst tilstand: vi har åpne partielle stier, men hverken utvidelse eller reduksjon kan utføres siden R ikke kan kobles med noen annen literal i matrisen.
- Eksempelet viser at vi må *backtracke over valg av kobling* i utvidelsessteget for å få en komplett søkealgoritme.

Implementasjon i Prolog – leanCoP

```
prove(Mat) :-
  append(MatA,[Cla|MatB],Mat), append(MatA,MatB,Mat1),
  \+member(-_,Cla), prove(Cla,Mat1,[]).
prove([],_,_).
prove([Lit|Cla],Mat,Path) :-
  (-NegLit=Lit;-NegLit\=Lit,-Lit=NegLit),
   ( member(NegLit,Path); append(MatA,[Cla1|MatB],Mat),
     append(MatA,MatB,Mat1), append(ClaA,[NegLit|ClaB],Cla1),
     append(ClaA,ClaB,Cla3), prove(Cla3,Mat1,[Lit|Path])
   ), prove(Cla,Mat,Path).
```

- leanCoP er en elegant Prolog-implementasjon av koblingskalkylen for klassisk logikk i normalform.
- Utviklet av Jens Otten ved Universitetet i Potsdam utenfor Berlin.
- Utnytter Prologs innebygde unifikasjon og backtracking.
- For mer info: `http://www.leancop.de/`