MAT1030 – Forelesning 27

Trær

Dag Normann - 4. mai 2010

(Sist oppdatert: 2010-05-04 14:13)

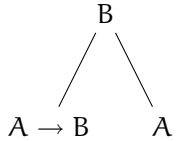
Forelesning 27

Oppsummering fra sist

- Forrige onsdag så vi på binære trær.
- For oss er syntakstrær viktige eksempler på binære trær.
- Vi så på forskjellige måter å traversere et binært tre på.
- Disse brukte vi til å representere termer via syntakstrær og de tre vanlige notasjonsformene for termer:
  1. *Infix* eller vanlig notasjon.
  2. *Prefix* eller polsk notasjon.
  3. *Postfix* eller baklengs polsk notasjon.
- Vi så på de binære trærne som en induktivt definert mengde, og på det tilhørende prinsippet for definisjoner ved rekursjon.
- Vi oppfattet prosedyrene som skriver ut de tre notasjonsformene fra syntakstreet som eksempler på trerekursjon.
- Det er meningen at dere skal kunne finne et syntakstre fra en formel eller en term, og at dere skal kunne skrive formelen eller termen med de tre notasjonsformene.
- Vi vil anvende dette tankegodset når vi kommer til unifiseringsalgoritmer.
- Først skal vi se på en annen type trær.

Bevistrær

- En annen type trær som spiller en stor rolle i logikk, men også i informatikk, er bevistrærne.
- Et bevistre er et merket tre hvor hver node er merket med en formel.
- Bladnodene vil være opplagt sanne, aksiomer i en eller annen formalisert teori, og merket til en foreldrenode vil følge logisk fra merkene til barna, ut fra visse prinsipper.
- En mulighet er at vi kan få lov til å ha deler av et bevistre som ser ut som



  som uttrykker at hvis vi har bevist A og $A \to B$, så kan vi konkludere B.
- Et slikt bevistre vil da være en garantist for at formelen som merker roten må være en konsekvens av de aksiomene som er brukt.

- Vi skal ikke la dette utvikle seg til et kurs i logikk, eller bevisteori, men som et eksempel på bruk av trær, skal vi se hvordan vi kan finne et bevis for et utsagnslogisk uttrykk, et tre som vil være en garantist for at uttrykket er en tautologi.
- Det er et poeng at hvis uttrykket er på *svak normalform*, så har vi en prosedyre for å omforme syntakstreet til et forsøksvis bevistre, og alle bladene blir aksiomer nøyaktig når utgangspunktet var en tautologi.
- Vi minner om at et utsagnslogisk uttrykk er på svak normalform hvis vi har følgende.
  - Kun bindeordene $\wedge$, $\vee$ og $\neg$ brukes.
  - $\neg$ kan kun stå rett foran en utsagnsvariabel.
- Som eksempler på rekursive konstruksjoner som går ut over rekursjon over $\mathbb{N}$ så vi på hvordan vi systematisk kan
  - fjerne forekomster av $\rightarrow$ og $\leftrightarrow$ i et utsagn slik at vi får et ekvivalent utsagn med bare $\wedge$, $\vee$ og $\neg$, og
  - som simultanrekursjon, skrive om utsagnene A og $\neg A$ til svak normalform.
- Disse konstruksjonene kan også formuleres ved hjelp av rekursjon på syntakstrær.
- Siden vi begrenser oss til utsagn på svak normalform, kan vi ikke bruke den regelen vi ga eksempel på, ettersom $\rightarrow$ ikke inngår i vokabularet.
- Vi skal tillate en type forgrening, eller slutningsregel.

$$\begin{array}{ccc} & A_1 \vee \cdots \vee A_n \vee (B \wedge C) & \\ \swarrow & & \searrow \\ A_1 \vee \cdots \vee A_n \vee B & & A_1 \vee \cdots \vee A_n \vee C \end{array}$$

- Vi skal ikke la rekkefølgen av delutsagnene i et $\vee$-utsagn bety noe.
- De eneste bladnodene vi vil akseptere er noder med merkene

$$A_1 \vee \cdots \vee A_n \vee p \vee \neg p$$

  hvor p er en utsagnsvariabel,

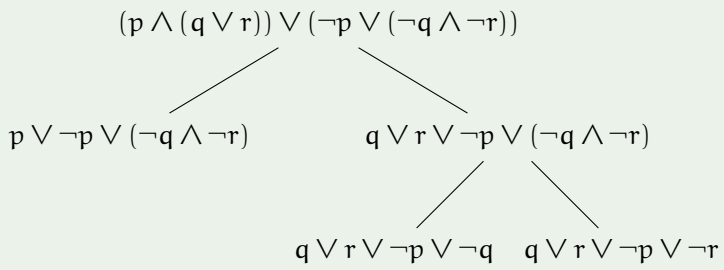
  altså disjunksjoner som inneholder både en utsagnsvariabel og negasjonen dens.
- Slike disjunksjoner er opplagt tautologier, og vil tjene som aksiomer.
- Vi skal se på et par eksempler hvor vi starter med et utsagn og utvikler et bevistre for dette utsagnet.
- Eksemplene forklares på tavla, om nødvendig.

Eksempel.

$$\begin{array}{ccc} & (p \wedge q) \vee (\neg p \vee \neg q) & \\ \swarrow & & \searrow \\ p \vee \neg p \vee \neg q & & q \vee \neg p \vee \neg q \end{array}$$

- Når vi bruker den distributive loven på $p \wedge q$ får vi en konjunksjon av to opplagte tautologier.
- Tautologier på denne formen har vi kalt aksiomer.
- Resultatet blir et bevistre.

Eksempel.



La oss nå se på prosedyren for å lage et forsøksvis bevistre fra et utsagnslogisk uttrykk.

Hvis vi har et utsagn A på svak normalform, kan A skrives på formen

$$A = A_1 \vee \cdots \vee A_n$$

hvor $n \geqslant 1$ og hver A_i enten er en konjunksjon, en utsagnsvariabel p eller negasjonen $\neg p$ av en utsagnsvariabel.

Hvis det finnes i og j slik at A_i er en utsagnsvariabel, og A_j er negasjonen av den samme, lar vi roten være bladnode merket med A, og vi har et bevistre.

Hvis alle A_i er utsagnsvariable eller negasjonen av slike, men vi er ikke i situasjonen over, lar vi roten være en bladnode, men konkluderer med at vi ikke har noe bevistre.

Ellers går vi til den minste i slik at $A_i = C \wedge D$.

Da lager vi to barn, hvor vi erstatter A_i med C når vi bygger treet videre under det ene barnet og vi erstatter A_i med D når vi bygger treet videre under det andre barnet.

Hvis begge disse deltrærne blir bevistrær, har vi konstruert et bevistre, ellers har vi ikke gjort det.

Vi illustrerer prosedyren på utsagnet

$$(\neg p \vee (p \wedge (p \vee q))) \wedge (p \vee \neg q \vee (q \wedge \neg p))$$

på tavla.

- Algoritmen for å finne et bevistre, om mulig, kan utvides til utsagn med kvantorer.
- Da er det ikke sikkert at vi alltid får et svar på om det finnes et bevis.
- Formelle bevis spiller en viss rolle om man ønsker å trekke ut numerisk informasjon rundt det beviste utsagnet.
- Formelle bevis spiller også en rolle i utviklingen av automatiserte korrekthetsbevis for programmer.
- Det neste temaet spiller også en rolle i anvendelser av logikk innen IT.

Unifisering

- Vi skal avslutte stoffet om trær med å se på en teknikk som kalles unifisering.
- Hvis vi har to termer hvor det forekommer variable, er det da mulig å erstatte disse variablene med andre termer slik at resultatene blir like?
- Vi skal begrense oss til et enkelt tilfelle, men den generelle unifiseringsalgoritmen spiller en stor rolle i logikkprogrammering og i automatisk bevissøk.

Eksempel.

La

$$t = (x + 0) \times ((0 + 0) \times x)$$
$$s = ((1 + 0) + 0) \times (y \times (1 + 0))$$

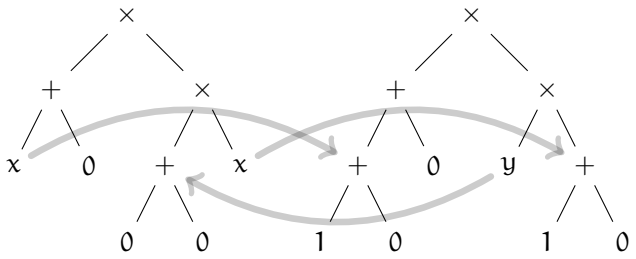
Er det mulig å erstatte x og y med termer slik at de to uttrykkene blir syntaktisk like?

I dette tilfellet kan vi sette inn $1 + 0$ for x og $0 + 0$ for y og begge uttrykkene blir

$$((1 + 0) + 0) \times ((0 + 0) \times (1 + 0)).$$

Dette ser vi lettest ved å betrakte syntakstrærne:

$$t = (x + 0) \times ((0 + 0) \times x)$$
$$s = ((1 + 0) + 0) \times (y \times (1 + 0))$$



- $x = (1 + 0)$
- $y = (0 + 0)$

Eksempel.

Nå lar vi

$$t = (x + 0) \times ((0 + 0) \times x)$$
$$s = ((1 + 0) + 0) \times (y \times (1 + y))$$

Hvis vi skal unifisere disse to uttrykkene, det vil si erstatte x og y med termer slik at s og t blir like, må vi få til at

$x + 0$ og $((1 + 0) + 0)$ blir like

$(0 + 0) \times x$ og $y \times (1 + y)$ blir like

samtidig.

Fra den første linjen ser vi at vi må sette inn $1 + 0$ for x, og hvis vi gjør det i den andre linjen, reduserer vi problemet vårt til å finne y slik at $y \times (1 + y)$ og $(0 + 0) \times (1 + 0)$ blir syntaktisk like.

Vi ser direkte at det er umulig.

Eksempel.

- Kan vi unifisere termene

$$(x \times (1 + 0)) + (((0 + 1) + z) \times (1 + x))$$

og

$$((0 + 1) \times (z + 0)) + ((y + 1) \times (1 + x))$$

det vil si, kan vi finne andre termer vi kan sette inn for x, y og z slik at de to uttrykkene blir like?

- Dette kan vi gjøre ved å sammenlikne termene systematisk og se om vi får fremtvunget hva vi skal erstatte x, y og z med for at resultatet skal bli vellykket.

Eksempel (Fortsatt).

- Først ser vi at begge termene har $+$ som hovedsymbol (dette ser vi lettere ut fra syntakstreet), og skal vi unifisere termene, må vi unifisere termene

  $(x \times (1 + 0))$ og $((0 + 1) \times (z + 0))$

  $(((0 + 1) + z) \times (1 + x))$ og $((y + 1) \times (1 + x))$

  simultant, det vil si vi må sette inn de samme termene for x, y og z i begge tilfellene.

Eksempel (Fortsatt).

Vi ser at begge termene i det første paret er produkttermer og det samme gjelder for begge termene i det andre paret. Oppgaven blir derfor å unifisere alle disse fire parene simultant:

1. x og $0 + 1$
2. $1 + 0$ og $z + 0$
3. $(0 + 1) + z$ og $y + 1$

4. $1 + x$ og $1 + x$

Vi ser at linje 1 forteller oss at vi må sette inn $0 + 1$ for x og i linje 4 har vi to like termer. De to andre linjene løser seg opp i fire nye linjer:

Eksempel (Fortsatt).

Vi må kunne unifisere

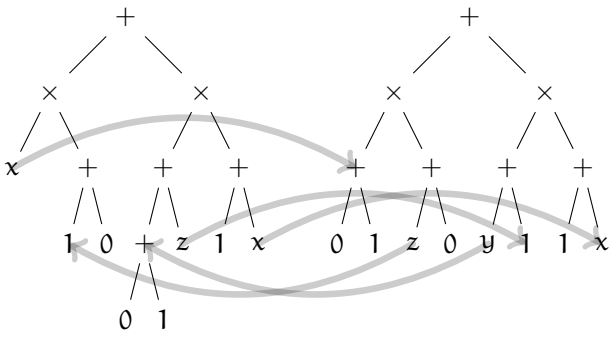
1. 1 og z
2. 0 og 0
3. $0 + 1$ og y
4. z og 1

simultant.

Nå har vi nådd bunnen av det som blir rekursjonen, og vi ser at hvis vi setter inn 1 for z og $0 + 1$ for y, i tillegg til at vi skulle sette inn $0 + 1$ for x, så får vi unifisering av de to første termene.

$$(x \times (1 + 0)) + (((0 + 1) + z) \times (1 + x))$$

$$((0 + 1) \times (z + 0)) + ((y + 1) \times (1 + x))$$



- $x = (0 + 1)$
- $z = 1$
- $y = (0 + 1)$

- Vi skal se på flere illustrerende eksempler før vi beskriver den endelige algoritmen for unifisering.
- Unifiseringen vil være mislykket hvis vi på et trinn må unifisere to forskjellige termer uten fri variable.
- Det er hele tiden viktig å tenke på at det er de syntaktiske uttrykkene som skal bli like, ikke bare de numeriske verdiene.

- For at en maskin skal kunne teste om en overgang er logisk korrekt, må uttrykk som spiller rollen som utsagnsvariable være syntaktisk like.
- Unifisering spiller en viktig rolle i logikkprogrammering, for eksempel i programmeringsspråk som PROLOG.

Eksempel.

- Unifiser $(1 + x) + (y + z)$ og $(1 + y) + (x + 0)$.
- Dette krever, i to trinn, at vi skal kunne unifisere
  1. 1 og 1
  2. x og y
  3. y og x
  4. z og 0

  simultant.
- Bare vi setter inn den samme termen for x, y og vi setter inn 0 for z, får vi en unifisering uansett.
- Vi vil derfor falle ned på den mest generelle unifiseringen, hvor vi beholder en av variablene x og y.
- Svaret blir derfor $(1 + x) + (x + 0)$

Eksempel.

- Vi skal unifisere $(1 + x) + (1 + y)$ og $(1 + 0) + (1 + (x + 1))$
- Dette reduseres til oppgaven å skulle unifisere
  1. $1 + x$ og $1 + 0$
  2. $1 + y$ og $1 + (x + 1)$

  simultant.

Eksempel (Fortsatt).

- Disse løser seg opp i fire oppgaver om å unifisere
  1. 1 og 1
  2. x og 0
  3. 1 og 1
  4. y og $x + 1$

  simultant.
- Linjene 1 og 3 er uproblematisk.
- I linje 2 ser vi at vi må sette inn 0 for x

- Det betyr at vi må omforme linje 4 til å skulle unifisere y og $0 + 1$, noe vi kan gjøre på direkten.

Eksempel.

- Anta at vi skal prøve å unifisere $x + (y + 0)$ og $(x + 1) + (1 + 0)$.
- Da må vi kunne unifisere
  1. x og $x + 1$
  2. $(y + 0)$ og $(1 + 0)$

  simultant.
- Andre linje er uproblematisk, men første linje er umulig, det finnes ingen term t slik at t og $t + 1$ er syntaktisk like.
- Hvis vi i forsøket på å unifisere par av termer må erstatte en variabel med en større term hvor variabelen forekommer, må vi konkludere med at unifisering er umulig.
- Ser vi på eksemplet $x + x$ og $y + (y + 1)$ reduseres det til samme type umulighet.

- Vi skal nå beskrive en rekursiv prosess som avgjør om det er mulig simultant å unifisere en endelig mengde par av termer i språket vårt.
- Etter at vi har gjort det, skal vi se på et eksempel på hvor vi kan få bruk for unifisering, og hvordan vi må tilpasse en konkret situasjon til en som kan håndteres av algoritmen vår.
- Vi vil starte med en endelig liste $x_1, \ldots, x_k$ av variable og to lister $t_1, \ldots, t_n$ og $s_1, \ldots, s_n$ av termer hvor disse variablene kan forekomme, og vi vil bestemme om det er mulig å erstatte variablene med termer $r_1, \ldots, r_k$ slik at hver t_i blir lik sin makker s_i.
- Får vi til det, sier vi at vi unifiserer parene simultant.
- Hvis t_1 er en variabel x_i og x_i forekommer i s_1, men s_1 er mer kompleks, gi opp unifiseringen. Det samme gjelder om situasjonen er omvendt.
- Hvis t_1 er variabelen x_i og x_i ikke forekommer i s_1, noterer vi at vi skal bruke sluttverdien av s_1 som termen r_i som skal erstatte x_i.

  Deretter erstatter vi alle andre forekomster av x_i med s_1 og fortsetter unifiseringsalgoritmen.

  I dette tilfellet har vi oppnådd å redusere antall variable med 1.

  Et korrekthetsbevis for algoritmen vil i første omgang være ved induksjon over antall variable.
- Hvis s_1 er en variabel, mens t_1 ikke er det, fortsetter vi på tilsvarende måte.
- Hvis s_1 og t_1 er samme term, bare stryker vi dette paret fra listen og fortsetter.
- Hvis ingen av tilfellene over gjelder har vi to muligheter:
  - s_1 og t_1 er åpenbart forskjellige, eksempelvis ved at den ene er en sum og den andre et produkt, den ene er et tall mens den andre er et funksjonsuttrykk eller de er forskjellige tall.

I dette tilfellet konkluderer vi med at unifisering er umulig.

– De er begge en sum eller de er begge et produkt.

Da erstatter vi paret s_1, t_1 med to par av mindre uttrykk, paret av de første addendene (faktorene) og paret av de andre addendene (faktorene) slik vi har sett eksempler på.

Deretter fortsetter vi algoritmen fra start.

- Hvis vi ikke finner ut underveis at unifisering er umulig, vil denne fremgangsmåten finne frem til de mest generelle termene $r_1, \ldots, r_k$ vi kan erstatte $x_1, \ldots, x_n$ med for å unifisere alle parene simultant.
- Korrekthetsbeviset er ved induksjon over antallet k av variable, med en underinduksjon over antall symboler sammenlagt i de to listene (dette antallet kan øke når vi kvitter oss med en variabel).
- Det finnes andre, og i praksis mer effektive, måter å gjennonføre unifisering på; hovedpoenget her var å vise prinsippene bak algoritmen.

  Det er ikke meningen at dere skal kunne gjengi denne algoritmen, men at dere skal kunne bestemme for hånd, i forholdsvis enkle tilfeller, om termer lar seg unifisere, og i tilfelle, gjennomføre det.

  Det kan i det minste være en fordel til eksamen å vite hva unifisering innebærer.

  Nå skal vi svare på spørsmålet om hva dette skal være godt for.

  Som tidligere nevnt spiller unifisering en rolle i automatisk bevissøk, eksempelvis i forbindelse med PROLOG.

  Vi skal se på et eksempel på hvordan vi systematisk kan søke etter et bevis for en påstand i et veldig enkelt logisk system.

  Eksemplet er så enkelt at vi ikke trenger hjelp av datamaskin til å finne et bevis, så det er mest til informasjon og motivasjon.

- Anta at vi har to aksiomer som vi kan bruke til å bevise at enkelte termer beskriver mindre tall enn andre termer:
  – $x < x + 1$
  – $x < y \wedge y < z \rightarrow x < z$.
- Så ønsker vi å søke etter et bevis for at

$$1 < ((1 + 1) + 1) + 1.$$

- Vi kan ikke finne en unifisering med aksiom 1, så håpet må være at vi har kommet frem til denne ulikheten som en anvendelse av aksiom 2.
- PROLOG vil da unifisere problemet vårt med konklusjonen i aksiom 2, slik at vi får $x = 1$ og $z = ((1 + 1) + 1) + 1$.
- Dette følger fra $1 < y \wedge y < ((1 + 1) + 1) + 1$, og PROLOG vil lete etter en verdi for y slik at begge delene av denne konjunksjonen kan bevises.
- $1 < y$ lar seg unifisere med aksiomet $x < x + 1$ ved å la $x = 1$ og $y = 1 + 1$.
- Prøver vi denne veien, må vi også prøve å bevise den andre delen av konjunksjonen for denne verdien av y, nemlig at $1 + 1 < ((1 + 1) + 1) + 1$.
- Igjen ser vi at dette ikke kommer direkte fra aksiom 1, så skal vi kunne bevise denne påstanden, må det være som en konsekvens av aksiom 2 og et bevis for en instans av

$$1 + 1 < y \wedge y < ((1 + 1) + 1) + 1.$$

- Setter vi inn $(1 + 1) + 1$ for y, i et forsøk på å la første del av denne konjunksjonen være en direkte konsekvens av aksiom 1, ser vi at også andre del blir en direkte konsekvens av aksiom 1.
- Vi har dermed systematisk lett oss frem til et bevis for den opprinnelige ulikheten.
- I virkelighetens verden kan vi trenge mer komplekse unifiseringer når vi prøver å finne bevis for påstander, og det å organisere søket på en slik måte at vi ofte raskt finner bevis der de finnes er et viktig teknologiskaspekt.
- Med dette gir vi oss med unifisering.
- Dette avslutter også innføringen i grafer og trær.