Avgörbara problem

Dagens föreläsning

- Motivation och bakgrund
- Acceptansproblemet olika beräkningsmodeller
- Tomhetsproblemet för DFA
- Ekvivalensproblemet för DFA
- Acceptansproblemet för CFG
- Tomhetsproblemet för CFG
- Ekvivalensproblemet för CFG
- Kontextfria språks avgörbarhet

Motivation och bakgrund

Turingmaskinen (TM) är trots sin enkelhet extremt kraftfull. Men även den har gränser! Det finns problem som inte kan lösas av en TM.

Fråga: om ett problem inte kan lösas av en enkel TM, kan vi då lösa det med en kraftfull modern dator?

Motivation och bakgrund, forts.

Nej, faktiskt inte (Church-Turing-tesen)! Faktum är att datorn inte ens är lika kraftfull som en TM. Varför?

Dagens föreläsning kommer att behandla bevis för att sådant vi studerat hittills (tack och lov) är avgörbart för Turingmaskiner såväl som vanliga datorer.

Acceptansproblemet

Det första problemet vi studerar är acceptansproblemet, alltså huruvida vår beräkningsmodell accepterar en viss sträng. Vi använder oss av en Turingmaskin för att simulera beräkningsmodellen (tänk att det är ungefär likvärdigt med att programmera en dator).

Formellt definierar vi problemet som ett språk (med viss modifikation beroende på beräkningsmodell) på följande sätt:

$$A_B = \{\langle B, w\rangle \mid \text{beräkningsmodellen } B \text{ accepterar inputsträngen } w\}$$

Strängen $\langle B, w\rangle$ är en kodning av beräkningsmodellen och inputsträngen som är dess indata.

Acceptansproblemet för DFA

Vi vill visa att $A_{DFA} = \{\langle B, w\rangle \mid B \text{ är en DFA som accepterar strängen } w\}$ är avgörbart.

Om vi kan skapa följande Turingmaskin M är beviset klart:

M = "Vid indata $\langle B, w\rangle$, där B är en DFA och w är en sträng:

1. Simulera B med indata w.
2. Om simuleringen slutar i ett accepterande tillstånd, *acceptera*. Om simuleringen slutar i ett icke-accepterande tillstånd, *refusera*.

Acceptansproblemet för DFA, forts.

Vi gräver inte ner oss i beviset – det borde gå att övertyga sig om att det är möjligt att programmera en dator så att den simulerar en DFA!

Har ni tid över är det en inte helt omöjlig uppgift i Java. ☺

Acceptansproblemet för NFA

Vi vill visa att $A_{NFA} = \{\langle B, w\rangle \mid B \text{ är en NFA som accepterar strängen } w\}$ är avgörbart (**teorem 4.1**).

Hur kan vi på enklast möjliga sätt göra detta?

Acceptansproblemet för NFA, forts.

När vi skapar ett nytt bevis, måste vi se till att det bygger på redan känd och bevisad kunskap. Allt blir då en logisk följd som inte kan sägas emot (förutsatt att vi inte gjort fel).

Vi skapar följande TM som bevisar att A_{NFA} är avgörbart (**teorem 4.2**).

N = "Vid indata $\langle B, w \rangle$, där B är en NFA och w är en sträng:

1. Konvertera B till en ekvivalent DFA C enligt proceduren i bokens teorem 1.39.
2. Kör M från teorem 4.1 på indatan $\langle C, w \rangle$.
3. Om M accepterar, *acceptera*, annars *refusera*.

Acceptansproblemet för reguljära uttryck

På samma sätt som i förra problemet använder vi oss av redan känd fakta för att skapa ett enkelt bevis. Beviset bygger på att använda teorem 1.54 ur boken för att konvertera uttrycket till en NFA. Eftersom vi vet att A_{NFA} är avgörbart är således även A_{REX} avgörbart.

Tomhetsproblemet för DFA

Ett lite intressantare och lite mindre intuitivt problem än det förra – kan vi undersöka om vår beräkningsmodell accepterar något alls?

Eftersom vi ser att det räcker att vi bevisar att en DFA klarar uppgiften för att NFA och reguljära uttryck skall vara logiska följder, koncenterar vi oss bara på följande problem:

$$E_{DFA} = \{\langle A \rangle \mid A \text{ är en DFA och } L(A) = \emptyset\}$$

Hur ska vi göra? Vad vet vi och hur kan vi utnyttja det? Hur beskriver vi det formellt?

Tomhetsproblemet för DFA, forts.

Vi vet tack vare definitionen att en DFA accepterar en sträng *omm* vi från starttillståndet kan följa någon sekvens av transitioner till ett accepterande tillstånd.

Följande TM kan avgöra problemet (**teorem 4.4**):

T = "Vid indata $\langle A \rangle$, där A är en DFA:

1. Markera starttillståndet i A.
2. Upprepa tills inga nya tillstånd markeras:
   2.1. Markera varje tillstånd som har en inkommande transition från något av de redan markerade tillstånden.
3. Om inget accepterande tillstånd markerats, *acceptera* annars *refusera*.

Ekvivalensproblemet för DFA

Kan vi avgöra huruvida två automater accepterar samma språk, alltså är ekvivalenta? **Teorem 4.5** säger att så är fallet. Yttryckt formellt:

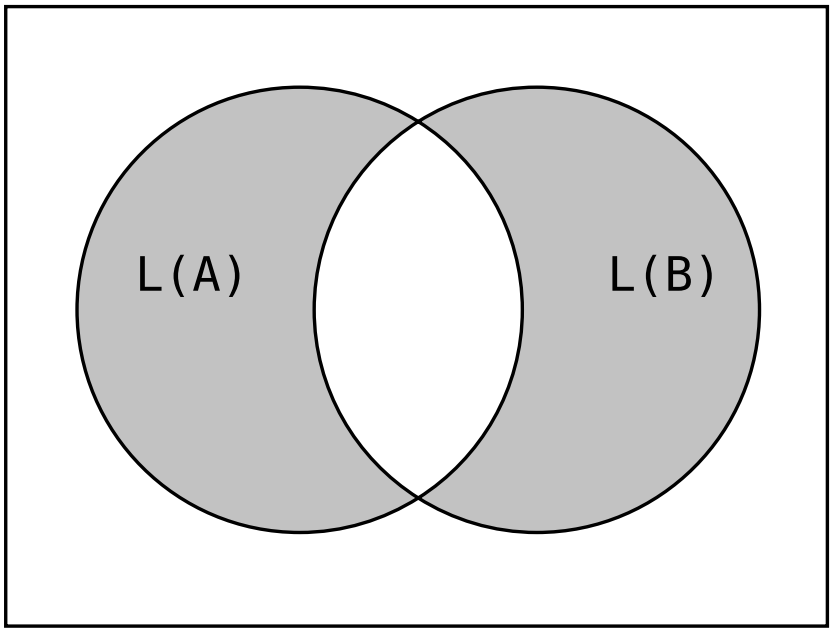
$$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ och } B \text{ är DFA:er och } L(A) = L(B)\}$$

Det här beviset är snyggt! Vi använder oss av lite mängdlära, samt att vi vet att vi kan bevisa att E_{DFA} är avgörbart.

Symmetrisk differens

En praktisk mängdoperation är att skapa den så kallade *symmetriska differensen* mellan två mängder $L(A)$ och $L(B)$. På svenska säger vi "de element som ingår i antingen $L(A)$ eller $L(B)$, men inte i båda".

Hur skriver vi symmetriska differensen matematiskt? Vad blir symmetriska differensen för två mängder som är lika?



Ekvivalensproblemet för DFA, forts.

Symmetrisk differens mellan två mängder som är lika är tomma mängden! Vi har nu något vi kan använda oss av: om vi skapar ett nytt språk som är symmetriska differensen mellan $L(A)$ och $L(B)$ och testar om det nya språket är tomt, så vet vi att $L(A) = L(B)$!

Låt oss skapa en ny DFA C där vi låter språket $L(C)$ vara:

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$

Vi låter $\overline{L(A)}$ betyda komplementet till $L(A)$.

Ekvivalensproblemet för DFA, forts.

För att avgöra om de två automaterna A och B är ekvivalenta räcker det således med att testa om $L(C) = \emptyset$. Teorem 4.4 garanterar oss att detta är möjligt.

Anledningen till att vi får göra detta är att det är bevisat att de reguljära språken är slutna under komplement, union och snitt – vilket är precis vad vi behöver för att kunna skapa det symmetriska snittet!

Acceptansproblemet för CFG

På föreläsningen om CFG nämndes att programmeringsspråk använder sig av CFG:er för att definiera språkets syntax. Vad en del av kompilatorn då gör, på sitt sätt, är att fråga språkets grammatik om programmet vi har skrivits kan genereras av grammatiken. Datorn arbetar således följande problem:

$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ är en CFG som genererar } w\}$$

Teorem 4.7 säger att A_{CFG} är avgörbart, men hur bevisar vi det? Kan vi testa oss fram?

Acceptansproblemet för CFG, forts.

Det är rätt frestande att testa alla möjliga deriveringar, men dessvärre är de oändligt många. Det är inte så lämpligt, eftersom vi vill ha svar inom ändlig tid.

Den som har gjort uppgift 2.26 i boken (rekommenderas för den flitige) vet att det går att bevisa att en derivering av en sträng med längden n går på $2n - 1$ steg *OM* grammatiken är på Chomsky-normalform. I och med detta har vi bara ett begränsat antal deriveringar att testa!

Acceptansproblemet för CFG, forts.

Beviset är således att vi skapar en TM som konverterar grammatiken G till en likvärdig grammatik som är på Chomsky-normalform (möjligt enligt teorem 2.9) och låter den nya grammatiken prova alla deriveringar av längden $2n - 1$ (eller längden 1 om $n = 0$). Om någon av dessa genererar strängen w skall TM:en *acceptera* annars *refusera*.

Tomhetsproblemet för CFG

Kan vi avgöra om en CFG inte genererar några strängar alls?

$$E_{CFG} = \{\langle G \rangle \mid G \text{ är en CFG och } L(G) = \emptyset\}$$

Vad innebär det att en grammatik genererar en sträng? Hur skall vi angripa det här problemet?

Tomhetsproblemet för CFG, forts.

Att en grammatik genererar en sträng innebär att vi från startvariablen kan göra ett antal deriveringssteg och få en sträng bestående av bara terminaler.

En idé är således följande: vi försöker kontrollera varje variabel, och se om den kan generera något som resulterar i en sträng bestående av terminaler. Om vi har lyckats se att en viss variabel A kan göra detta, så kan vi då undersöka om andra variabler kan generera en sträng bestående av terminaler eller A.

Upprepar vi detta tillräckligt länge, kommer vi till slut att veta om startvariablen kan generera en sträng bestående av terminaler eller inte.

Tomhetsproblemet för CFG, exempel

Följande grammatik känner vi igen från föreläsningen om CFG.

$$\begin{aligned} \langle \text{EXPR} \rangle &\rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle \\ \langle \text{TERM} \rangle &\rightarrow \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle \\ \langle \text{FACTOR} \rangle &\rightarrow (\langle \text{EXPR} \rangle) \mid a \end{aligned}$$

Använd metoden från förra sidan till att verifiera att vi från startvariabeln $\langle \text{EXPR} \rangle$ kan nå till terminalen a.

Tomhetsproblemet för CFG, forts.

Teorem 4.8 säger att E_{CFG} är avgörbart. Beviset går ut på att vi skapar följande TM:

T = "Vid indata $\langle G \rangle$, där G är en CFG:

1. Markera samtliga terminaler i G.
2. Upprepa tills inga nya symboler markeras:
   2.1. Markera varje variabel A, där G innehåller en regel $A \rightarrow U_1 U_2 \ldots U_k$ och varje symbol $U_1, \ldots, U_k$ är markerad.
3. Om startvariablen inte har markerats, *acceptera* annars *refusera*."

Ekvivalensproblemet för CFG

Kan vi avgöra om två CFG:er genererar samma språk, alltså följande problem:

$$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ och } H \text{ är CFG:er och } L(G) = L(H)\}$$

Hur skulle vi kunna börja, vad kan vår lösning bygga på?

Ekvivalensproblemet för CFG, forts.

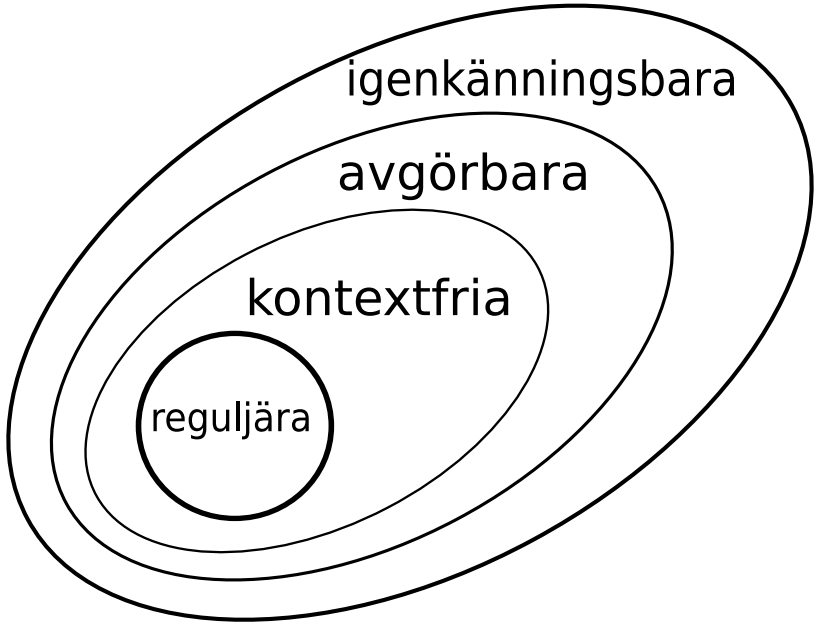
Att kolla om grammatikerna "ser likadana ut" (frånse från vad variablerna heter, men se om vi har likvärdiga regler) hjälper oss inte, tyvärr. Det ger bara en enkel lösning, som i stort sett är att kolla om två grafer är isomorfa.

Hur vi än försöker vrida och vända på problemet och hitta på lösbara specialfall så kommer vi att fastna: EQ_{CFG} är **oavgörbart**!

Kontextfria språks avgörbarhet

Teorem 4.9 säger att alla kontextfria språk är avgörbara. Beviset bygger på att vi kan lösa acceptansproblemet A_{CFG}.

Detta är viktigt för teorin bakom formella språk. Vi vet således att följande samband råder:



Bonus: Flertydiga grammatiker, återkomsten

En naturlig fråga för den intresserade gällande grammatiker är följande:

$$AMBIG_{CFG} = \{\langle G \rangle \mid G \text{ är en flertydig CFG}\}$$

Detta problem är oavgörbart (trots att vi ju har en så fin definition av flertydighet)! I framtida föreläsningar kommer metoder som behövs att beskrivas, men man kan visa att *om* detta problem skulle vara avgörbart så skulle även alla andra oavgörbara problem vara det. Titta på övningsuppgift 5.26 när det ligger rätt i tiden!