Låt Datorn Skriva Programmen!

Roland Olsson

Avdeling for informatikk, HiØ, Os Allé 11, 1757 Halden

Produktion av software har i dag minst lika stor ekonomisk och praktisk betydelse som till exempel biltillverkning, både som självständig industri och som stöd till annan verksamhet. Allt från tvättmaskiner, mobiltelefoner, bilmotorer och flygplan till web-browsers, ordbehandling och simuleringar styrs av program som år för år blir större och större och därmed svårare och dyrare att utveckla. Denna trend, som ibland kallas software-krisen, gäller alltså en stor del av all programmering. Krisen beror bland annat på att programmering är ett mycket kreativt arbete som i likhet med skönlitterärt författande ofta är mer konst än vetenskap eller välplanerat ingenjörsarbete.

Lösningen på software-krisen är automatisk programmering, vilket betyder att en dator självständigt genererar program som uppfyller krav formulerade av en människa. Dessa krav är en så kallad specifikation som säger vad ett program skall göra men kanske inte någonting om hur det fungerar. Automatisk programmering har tre typer av användare, nämligen Novisen, Proffset och Forskaren.

- Novisen kan inte programmera och är enbart intresserad av att lösa ganska enkla praktiska problem så snabbt och enkelt som möjligt med hjälp av en dator.
- Proffset har goda färdigheter i många programmeringsspråk och utvecklar ett komplext program för kommersiellt bruk. Trots att Proffsets insats är mycket kreativ, välplanerad och grundlig tar utvecklingen av detta komplexa program fem år och ger ett resultat som bara löser en liten del av det ursprungliga problemet och som är fullt av bugs (fel).
- Forskaren studerar metoder för utveckling av ”intelligenta program” med generell förmåga att lösa nya problem som är väsentligt annorlunda än problem som lösts tidigare. Forskaren anser att det sannolikt är omöjligt att själv skriva sådana program eftersom enorm programkomplexitet skulle krävas. Ett system för automatisk programmering kanske kan hantera denna komplexitet bättre än Forskaren, på samma sätt som program för numeriska beräkningar ofta är helt överlägsna matematiker med miniräknare.

Jag har utvecklat ett system för automatisk programmering, ADATE, vilket betyder Automatic Design of Algorithms through Evolution. Småningom kan ADATE användas av både Novisen, Proffset och Forskaren. I ADATE består en specifikation i huvudsak av ett antal exempel på input och krav som motsvarande output bör uppfylla. I enkla specifikationer kan dessa krav vara en specifik output för varje exempel på input, det vill säga input-output par. Här är några exempel på hur Novisen, Proffset och Forskaren kan använda ADATE.

Novisen

I stort sett alla användare av datorer måste ibland utföra enkla repetitiva uppgifter som en dator kan göra automatiskt om den bara programmeras riktigt. För att lösa sådana uppgifter finns det speciella så kallade scripting-språk, som Novisen dessvärre inte kan eller har tid att lära sig.

Ett exempel som är aktuellt inför milleniumskiftet om knappt två år är att byta ut alla tvåsiffriga års-angivelser i ett dokument mot fyrsiffriga. Till exempel skall ett datum 15/3 95 bytas ut mot 15/3 1995. Antag att Novisen redigerar dokumentet och själv ändrar cirka fem datum-angivelser. Före varje ändring trycker Novisen på en knapp som heter "Begin Example". När ändringen är färdig trycker Novisen på knappen "End Example". Efter att novisen har gjort detta fem gånger kan ADATE skriva ett program på ett par rader som automatiskt ändrar resten av dokumentet och dessutom alla andra dokument med tvåsiffriga årsangivelser.

Ett exempel som jag själv löste med ett tre-linjers script i språket PERL för några dagar sedan är att redigera en fil som innehåller en ordlista med 25000 engelska ord och ett ord per rad. Jag önskade att sätta citat-tecken runt varje ord och dessutom lägga till ett komma. Till exempel skall raden

```
acknowledge
```

förändras till

```
"acknowledge",
```

Om ADATE ges ungefär fem exempel på en rad före, respektive efter ändring, kan det automatiskt skriva ett program som förändrar resten av ordlistan. Även om Proffset skriver detta program på några minuter, representerar det ett till synes oöverstigligt hinder för Novisen.

Proffset

Varje gång som Proffset använder ADATE är det för ett mycket specialiserat ändamål som är svårt för utomstående att identifiera.

Anta till exempel att Proffset arbetar med nästa version av ett ordbehandlingsprogram och behöver en programdel för rättning av stavfel. Om till exempel användaren av ordbehandlingsprogrammet skriver "fortutous" skall detta rättas till "fortuitous". Proffset vill ha ett program (eller subrutin) som givet ett ord X och en ordlista finner det ord Y i ordlistan som mest liknar X, men det är inte uppenbart hur graden av likhet mellan X och Y skall mätas. Proffset ger därför en ordlista och ett antal exempel på (X,Y) till ADATE, som automatiskt producerar ett program som använder en metrik kallad LCS (Longest Common Subsequence) för att hitta bästa matchande ord i ordlistan. ADATE har aldrig hört talas om LCS, men producerar ändå automatiskt ett program som använder denna metrik. Proffset har heller aldrig hört talas om LCS och kan dessvärre inte förstå hur programmet producerat av ADATE fungerar, men är nöjd ändå eftersom programmet finner mycket bra matchande ord.

Ett annat exempel är ett Proffs som arbetar med program för styrning av tändning och insprutningsförlopp i bilmotorer vars tillverkare med huvudkontor i Göteborg skall förbli anonym. Proffsets problem är svängningar, kallade slag och såg, i drivlinan (motor och växellåda) som uppstår vid plötsliga gasändringar. Genom omfattande experiment har man klarlagt att det är orimligt att reducera svängningarna genom mekaniska ändringar. Man har två alternativ att välja bland, nämligen att isolera bort problemet så att föraren inte märker det eller att förbättra programmet som styr motorn. Emellertid har man en experimentellt framtagen komplex matematisk modell av drivlinan och kan med hjälp av denna snabbt se hur bra ett givet styrprogram är. Ett input-exempel till ADATE består av ett antal sensordata, bland annat temperatur, luftflöde, gaspådrag och "svängningshistorik" för til exempel de senaste två sekunderna. Motsvarande output är tändtidpunkt och insprutningsintervall, men optimala värden på dessa är inte kända och behöver inte vara det för att ADATE skall kunna användas. Den matematiska modellen av drivlinan kan emellertid betygsätta ett styrprogram producerat av ADATE, vilket räcker för att ADATE kan producera ett rimligt optimalt styrprogram.

Fords motorlabb i Detroit har redan använt en alternativ metod till ADATE, kallad Genetisk Programmering, för liknande tillämpningar.

Forskaren

Eftersom jag är Forskaren, skrivs detta avsnitt i jag-form.

Ett av mina intressen är förhållandet mellan programmeringsspråk, mänskliga språk och tänkande.

Kopplingen mellan matematik, logik, språk och tänkande har studerats i århundranden av lingvister, logiker och matematiker. Till exempel ansåg språkforskaren Esaias Tegner d.y., som var chef för Svenska akademins ordboksredaktion i början av 1900-talet, att det inte finns några tankar utom dem som kan uttryckas i språk. Hans verk "Språkets makt öfver tanken" är tyvärr inte citerat

av mer välkända amerikanska lingvister som senare arbetat inom samma ämne.

På 1930-talet studerades indianspråk av de amerikanska lingvisterna Benjamin Whorf och Edward Sapir som formulerade hypotesen att dessa språk saknar ingredienser nödvändiga för vetenskapligt tänkande. Den kontroversiella Whorf-Sapir hypotesen är att tänkande underlättas eller ibland till och med möjliggörs genom att välja ett lämpligt språk.

Jag menar att alla mänskliga språk är utomordentligt lämpade för tänkande jämfört med programmeringsspråk och andra formella språk som till exempel predikatlogik. Min uppfattning är alltså, i motsats till Whorfs, att exempelvis indianspråket hopi och svenska stöder tänkande lika bra. Whorf-Sapir hypotesen är sannolikt felaktig vid jämförelse mellan mänskliga språk, men korrekt när ett mänskligt språk jämförs med ett formellt.

Inspirerad av Whorf, konstruerade lingvisten och socialantropologen James Brown det mänskliga språket LOGic LANguage (LOGLAN) i slutet av 1950-talet och beskrev det i en artikel i Scientific American 1960. LOGLAN har en mycket enkel grammatik jämfört med naturliga språk, men påstås ändå vara bättre lämpat för mänskligt tänkande än dessa.

Det är viktigt att förstå skillnaden mellan ett formellt, matematiskt språk och ett mänskligt språk som LOGLAN. Vilken utsaga eller text som helst på svenska eller engelska kan enkelt översättas till LOGLAN, medan rimligt semantik-bevarande översättning till ett formellt språk ofta är i stort sett omöjlig.

Det finns många andra intressanta exempel på skalan från naturliga mänskliga språk till rena matematiska formalismer.

Ett exempel som ligger på den mänskliga delen av skalan kommer från lingvisten Noam Chomsky, som också haft stort inflytande över modern datavetenskap. Han har myntat begreppet djupstruktur, som grovt sett är en minsta möjliga semantikbevarande delmängd av ett naturligt språk. Ett liknande, men mindre känt arbete är Anna Wierzbickas "Lingua Mentalis".

På den matematiska delen av skalan finns det hundratals mer eller mindre begränsade logiska formalismer. Det mest ambitiösa försöket att bringa de naturliga språkens uttrycksförmåga till logikens värld gjordes under 1960-talet av Richard Montague, som definierade en logik med dussintals temporala, spatiala, kausala och andra kvantorer och operatorer. Montagues semantik, på grund av sin komplexitet ibland kallad en "Rubik cube machine", har bildat skola för 1980- och 1990-talens forskning i formell semantik.

Det förefaller nästan omöjligt att manuellt skriva ett program som fullständigt förstår ett mänskligt språk som LOGLAN och som självt kan "tänka" i detta språk. Emellertid är det möjligt att skriva en ADATE-specifikation av ett sådant program. Denna specifikation liknar en serie läroböcker som startar med en bok för babies och som gradvis innehåller mer och mer komplicerade språkliga kommunikationsproblem fram till böcker med problem som kräver självständigt tänkande och intelligens.

Det finns minst tre skäl för att ADATE i framtiden skall kunna använda en sådan specifikation för att producera ett program som har sunt förnuft, allmän

intelligens och generaliserande förmåga. Dessa tre skäl kallar jag existensindiciet, Moores lag och Turingmaskinstyrka.

Existensindiciet utgår från att naturlig evolution har producerat bland annat människans intellekt. ADATE är automatisk programmering genom simulerad evolution, vilken styrs av lagar som kanske är mer optimerade och välutvecklade än de till synes primitiva mekanismer som styr naturlig evolution. Om primitiv naturlig evolution har lyckats redan, varför är inte avancerad simulerad evolution lika framgångsrik?

Naturlig evolution använder enorma mängder datakraft bland annat genom extremt massiv parallellbearbetning. Simulerad evolution fram till den komplexitet som behövs för "intelligens" behöver sannolikt mycket mindre datakraft, men ändå så stora mängder att inte ens dagens snabbaste parallelldatorer kan utföra simuleringen på rimlig tid. Enligt Moore's lag dubblas hastigheten på en processor på mellan ett och två år. Om exempelvis 10 år kommer en processor enligt halvledarindustrins planer att vara cirka 100 gånger snabbare än en typisk processor av årsmodell 1998. En parallelldator med 1000 processorer kan alltså år 2008 ge 100000 gånger så mycket CPU-kraft som jag nu använder när jag kör ADATE på en PC.

Av stor vikt är att det så kallade funktionella språk som ADATE använder kan formulera program skrivna i alla andra språk. I datavetenskapen kallas denna egenskap för Turingmaskinstyrka, vilket också gäller alla vanliga programmeringsspråk som därmed alltså allihop kan formulera alla kända algoritmer, men med varierande lätthet naturligtvis. Den alternativa metoden Genetisk Programmering som omtalades ovan har inte Turingmaskinstyrka, utan används huvudsakligen för matematiska modeller som är mer begränsade än generella program.

Funktionell programmering kommer från forskning i matematisk logik utförd av Alonzo Church i början av 1940-talet innan elektroniska datorer existerade. Ett välskrivet funktionellt program är nästan alltid mindre och enklare än motsvarande program skrivna i andra programmeringsspråk. Ett funktionellt program är heller aldrig orimligt mycket långsammare. Ett funktionellt språk är sannolikt nästan optimalt med avseende på programmens enkelhet.

När är det en fördel att använda automatisk programmering?

ADATE-systemet är överlägset andra liknande system, men är fortfarande en forskningsprototyp som inte utan vidare kan användas till alla ovanstående tillämpningar. Det finns minst två hinder för att använda systemet, nämligen:

1. Specifikationen kan vara svår att skriva. För många användningsområden finns det redan effektiva standardalgoritmer, till exempel sortering och sökning i databaser. Inom överskådlig framtid är det nog enklare att

anpassa dessa standardalgoritmer till en relevant användning än att skriva en pedagogisk specifikation av dem.

2. Automatisk programmering genom simulerad evolution kräver enorma mängder datakraft.

ADATE kan alltså med fördel användas när en pedagogisk specifikation är lättare att skriva än ett motsvarande program och när mycket stora CPU-resurser finns tillgängliga, antingen i form av en parallelldator eller en CPU tillverkad efter år 2007.

En viktig egenskap hos ett system för automatisk programmering är så kallad skalerbarhet. God skalerbarhet innebär att varken svårigheten att skriva specifikationer eller inferenstiden ökar orimligt mycket när de syntetiserade programmen blir större och större. Observera att ett syntetiserat program kan ha exekveringstider på några millisekunder även om inferenstiden är en vecka, det vill säga även om det tar en vecka för ADATE att syntetisera programmet. Jag menar att ADATE kommer att ha god skalerbarhet till i stort sett obegränsad storlek hos syntetiserade program.