Matematik kan ge färre fel i datorprogram

Datorer gör fel – det är något som vi har fått vänja oss vid! När en ordbehandlare som Microsoft Word ändrar stavningen på ett ord på ett felaktigt sätt är det mest lustigt. När PC-datorn kraschar mitt i ett arbete är det inte så roligt. När felen leder till skada på liv och egendom är det katastrofalt. Fel i datorsystem beror nästan uteslutande på att programmen som styr datorernas arbete är felaktigt konstruerade. Varför tar man då felaktiga program i bruk? Jo, det är i praktiken nästan omöjligt att undvika felaktigheter.

Så kan det gå när det är fel i programmet. Den europeiska Ariane 5-raketen havererar 40 sekunder efter uppskjutningen i juni 1996.

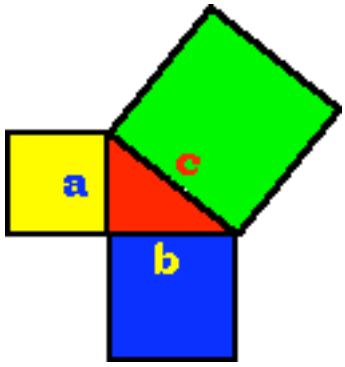
Programutveckling börjar med att man skriver en *kravspecifikation* som säger vad programmet skall göra. Många fel beror på att kravspecifikationen inte rätt beskriver programmets uppgift eller så är den är ofullständig eller tvetydig. Därefter skall en programmerare *koda* en stor mängd mycket detaljerade instruktioner – ett *program* – som beskriver hur datorn skall åstadkomma det som står i kravspecifikationen. Här kan fel uppstå genom att programmeraren missförstår kravspecifikationen. Dessa felkällor är i grunden kommunikationsproblem. Kravspecifikationen skrivs normalt i klartext på svenska eller annat mänskligt språk och kan därför – som alla texter – tolkas på olika sätt. Om programmeraren inte tolkar den som författaren avsåg blir programmet felaktigt.

Fel kan också uppstå genom att programmeraren konstruerar programmet fel trots att kravspecifikationen tolkats riktigt. Programmet är i praktiken så stort att man inte kan ha sådan överblick och metodik att programmet från början säkert konstrueras rätt. Program måste därför genomgå omfattande provning – *testning* – innan de kan tas i bruk. Under testningen hittar man felaktigheter i programmet, vilka spåras till en bestämd del av det

och sedan rättas. Sedan datorernas barndom har man lagt mycket arbete på att förbättra metodiken för programutveckling för att hålla nere antalet fel, men samtidigt har programmen blivit större och mer komplexa vilket lett till en utveckling i motsatt riktning.

En väsentlig svårighet med testning är att programmet kan utsättas för så många olika situationer att det i praktiken inte går att testa alla. Ett program som har i uppgift att addera två sexsiffriga tal har en miljon miljoner olika körningsfall. Testar man 1000 fall i sekunden tar det drygt 30 år att gå igenom alla. Istället testar man några utvalda tal och hoppas att programmet gör rätt i övriga fall. Därför kan – i praktiken – inget program utom de mest triviala visas vara felfria genom testning. Ställer man höga krav på att programmet gör rätt krävs mycket omfattande testning till stora kostnader. Detta har man insett ända sedan datorernas barndom och därför arbetat med forskning kring möjligheten att matematiskt bevisa att program är felfria – något som brukar kallas *formella metoder*.

Tester och bevis: *Pythagoras sats* säger att i en rätvinklig triangel är kvadraten på den långa sidans längd lika med summan av kvadraterna av de kortare sidornas längder. Detta har varit känt i många årtusenden, men endast så att man genom uträkningar kunnat se att det stämde. Med andra ord så testade man förhållandet mellan triangelns sidor. Först c:a 500 år f.kr. bevisade den grekiske matematikern Pythagoras att det måste gälla för *alla* rätvinkliga trianglar. På liknande sätt kan man bevisa att ett program gör rätt i alla situationer.



Pythagoras sats: $a^2+b^2 = c^2$.

Bevis av program kräver att kravspecifikationen uttrycks i ett *matematiskt språk*. Sådana är till sin natur precisa, så nästan alla problem med tolkning av kravspecifikationen försvinner. I praktiken leder det också till att en sådan *formell* kravspecifikation lättare blir mera fullständig än en som skrivits i naturligt språk. Eftersom fel, oklarheter och tolkningsutrymme hos kravspecifikationen är en väsentlig felkälla hos program får man redan här en minskning av antalet fel. Ännu bättre blir det när man genomför matematiska bevis – *formell verifiering* – av att programmen är riktiga. I princip kan man då få garanterat felfria program. I praktiken kan man dock inte nå så långt, bl.a. därför att man aldrig kan vara helt säker på att kravspecifikationen är riktig.

Under de senaste decennierna har formella metoder börjat användas i ökande utsträckning inom industriell programutveckling. Metoden är tekniskt komplicerad och kräver omfattande datorstöd. Dess kostnadseffektivitet är därför omtvistad, men den är klart att den kommit för att stanna i branscher där felaktigheter kan leda till svåra olyckor eller där det medför mycket stora kostnader att rätta felaktigheter. Exempel på sådana områden är styrsystem för flygplan och järnväg eller konstruktion av processorer för datorer vilka är dyra att utveckla och tillverkas i mycket stora serier.

Lars-Henrik Eriksson vid Uppsala Universitet, institutionen för Informationsteknologi, forskar kring metodiken vid användningen av formella metoder – alltså hur man utvecklar och använder formella specifikationer, hur man genomför formell verifiering och hur man skall tolka resultaten från den formella verifieringen. Skulle ett försök att bevisa programmet korrekt genom formell verifiering misslyckas är det viktigt att kunna tala om varför – annars är det svårt att lokalisera och rätta den del av programmet som är felaktigt utförd. Formell verifiering utförs i praktiken av speciella datorprogram, *teorembevisare* eller *modellcheckare*. De kan ofta automatiskt visa situationer där programmet gör fel, men dessa situationer innehåller mängder med irrelevanta detaljer och måste analyseras innan man kan tala om varför felet inträffade och därmed ha möjlighet att rätta det.

Det tillämpningsområde Eriksson främst har ägnat sig åt är signalsystem för järnvägar – ett område där kraven på korrekt funktion är mycket höga. I samarbete med konsultföretaget Industrilogik L4i AB som arbetar med formella metoder åt bl.a. Banverket studerar han hur sådana analyser utförs. Själva den formella verifieringen sker huvudsakligen automatiskt, medan detta ”efterarbete” är ett omfattande hantverk som utgör en stor del av svårigheten och därmed kostnaden vid praktisk användning av formell verifiering.

Lars-Henrik Eriksson

Institutionen för informationsteknologi, Uppsala Universitet