Computer-ondersteund redeneren: de boekhouder steunt de denker

Rede uitgesproken bij de aanvaarding van het ambt van hoogleraar Computerondersteund redeneren aan de Faculteit der Natuurwetenschappen, Wiskunde en Informatica van de Radboud Universiteit Nijmegen op vrijdag 9 maart 2007

Prof.dr. Herman Geuvers

9 maart 2007

Mijnheer de Rector Magnificus,
zeer gewaardeerde toehoorders,

Computers maken fouten. Soms zijn dat vervelende fouten waardoor gebruikers zich ergeren of wat werk verliezen, maar soms zijn dat grote fouten, waardoor grote sommen geld verloren gaan of zelfs mensen het leven zouden kunnen verliezen.

Vervelende fouten kennen we allemaal, bijvoorbeeld als ons Windows-besturingssysteem vastloopt. Gebruikers zijn hieraan gewend en slaan voortdurend hun werk op en maken backups. Van een andere orde is de fout die gemaakt werd in het besturingssysteem van de Ariane 5 raket, waardoor deze op 4 juni 1996 veertig seconden na lancering plotseling uit de koers raakte en zichzelf opblies. Dat de raket zichzelf opblies, was een 'feature': een bewust aangebrachte veiligheidsnoodklep om te voorkomen dat de raket zich ergens in de grond boort. Dat de raket zo uit koers raakte was een software 'bug'. Het probleem bleek dat de conversie van een 64-bit floating point getal naar een 16-bit geheel getal waarde niet goed werkte voor grote getallen. Bij de Ariane 4, waar deze software ook gebruikt werd, traden dit soort grote waarden niet op, maar de Ariane 5 was een snellere raket. Voor wie niet weet wat

een 64-bit floating point getal is: het gaat er vooral om u te realiseren dat het hier een heel klein stukje programmacode betreft waardoor een raket van 500 miljoen dollar verloren ging, waaraan ESA zo'n tien jaar had gewerkt en waarin zeven miljard dollar aan ontwikkelkosten was geïnvesteerd.

Van een soortgelijke orde is de fout die Intel in 1994 maakte in de nieuwe Pentium chip. De deeloperatie was fout geïmplementeerd op de chip, iets wat een gemiddeld gebruiker nooit zou merken, maar een Amerikaans wiskundige wel. In dit geval zat de fout ook in enkele kleine instructies. Die waren wel al in hardware gegoten – op een chip – en miljoenen keren verspreid. Hoewel in dit geval zowel de fout zelf als de gevolgen klein waren, waren de publicitaire gevolgen groot. Intel probeerde de 'bug' af te doen als een 'flaw' – een vergissinkje – maar via het internet verspreidde het verhaal zich snel met grote reputatieschade en dalende aandelenkoersen voor Intel als gevolg.

Hoe voorkomen we deze fouten? Door de software en hardware op een gedegen manier te ontwikkelen en de gewenste eigenschappen te verifiëren. Dat is een ingewikkeld en omvangrijk werk, want we moeten de software zelf – vele duizenden regels code – analyseren, maar ook de omgeving modelleren waarin deze software geacht wordt te opereren. De besturingssoftware van de Ariane 5 bijvoorbeeld deed het goed in de Ariane 4, maar niet in de veel snellere Ariane 5.

Er zijn veel methoden om software en hardware te verifiëren, bijvoorbeeld door te testen. Testen is het gestructureerd zoeken naar fouten door bij bepaalde invoer te kijken of de uitvoer klopt. Hiermee kunnen we fouten vinden, maar nooit alle, en als we geen fouten vinden hebben we geen garantie dat die er ook niet zijn. De ultieme vorm van verificatie is een correctheidsbewijs: een wiskundig bewijs dat een bepaald stuk computer code aan bepaalde eigenschappen voldoet.

Bewijzen dat software of hardware correct is, is een ingewikkelde materie, omdat de systemen groot zijn. Daarom gebruiken we computer programma's om ons daarbij te helpen, de stellingbewijzers of bewijsassistenten. Zelf gebruik ik het liefst het woord 'bewijsassistent' omdat 'stellingbewijzer' suggereert dat het systeem automatisch stellingen voor ons bewijst. Dat is echter niet zo. De bewijsassistent helpt bij het nagaan of de definities goed zijn, houdt de bewijstoestand bij en kan bewijsstappen suggereren, maar het bewijs moeten we als gebruiker toch zelf leveren.

Het gebruik van bewijsassistenten heeft de laatste jaren in de informatica op verschillende plekken ingang gevonden. Zo zei Bill Gates in 2002:

> 'Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.'

Gates heeft het hier over de bij Microsoft ontwikkelde Static Driver Verifier . Het blijkt dat veel van de fouten in Windows niet worden veroorzaakt door het besturingssysteem zelf, maar door slecht geschreven drivers voor randapparatuur die interfereren met het besturingssysteem.

Ook NASA gebruikt bewijsassistenten om software voor de luchtverkeersleiding te verifiëren. Intel gebruikt bewijsassistenten bij het verifiëren van de nieuwe chips .

Mijn onderzoek houdt zich bezig met bewijzen op de computer. Dat betekent dat de bewijzen in een heel precieze formele taal zijn gegoten, zodat we ze op de computer kunnen opslaan en ze met behulp van een computerprogramma kunnen manipuleren. Het voordeel van geformaliseerde bewijzen is dat ze eenvoudig gecheckt kunnen worden. Daarvoor is het natuurlijk wel nodig dat ze zeer gedetailleerd zijn opgeschreven.

Bewijzen

Een bewijs is volgens Van Dales woordenboek 'een feit of redenering waaruit de juistheid van een bewering onweerlegbaar blijkt'. In de wiskunde is een bewijs absoluut. De correctheid van een bewijs kan door iedereen nagegaan en vastgesteld worden. Dat is mogelijk doordat een wiskundig bewijs gereduceerd kan worden tot een serie van heel kleine stapjes die stuk voor stuk eenvoudig en onomstotelijk te verifiëren zijn. Deze stapjes zijn zo klein dat geen wiskundige dit daadwerkelijk gaat doen, maar het is een algemeen geaccepteerde aanname dat de wiskundige bewijzen die we vinden in boeken en tijdschriftartikelen volledig in alle detail uitgespeld zouden kunnen worden.

Het komt voor dat een wiskundige stelling fout blijkt te zijn. In dat geval zijn niet alle stappen tot in volledig detail uitgespeld en nagegaan en het is dan ook altijd mogelijk om in het bewijs een stap (soms meer) aan te wijzen die niet te verifiëren valt. Het zal duidelijk zijn dat we bij het heel precies maken van de kleine bewijsstappen en het nagaan of al deze kleine bewijsstappen samen een correct bewijs vormen goed een computer kunnen gebruiken. Dit is het boekhouderswerk dat we aan een machine

kunnen overlaten. Eerst nog iets over de rol van het wiskundig bewijs in de wiskunde zelf. Een bewijs speelt twee rollen:

A Een bewijs *overtuigt* de lezer van de correctheid van het gestelde.

B Een bewijs *legt uit* waarom het gestelde geldt.

Bij het eerste punt gaat het dus vooral om het precies nagaan van alle redeneerstappen en inzien dat deze inderdaad kloppen. Dit is een vrij boekhoudkundige activiteit: je hoeft niet naar het grotere plaatje te kijken, maar alleen na te gaan of iedere volgende stap een juiste is. Bij het tweede punt gaat het vooral om het geven van intuïtie over de stelling: waarom is het zo 'natuurlijk' dat dit geldt en hoe zijn we op het idee gekomen om het zo te doen?

Het QED-manifest

Als we kijken naar de twee rollen die een bewijs speelt, dan zien we dat een computer rol (A) prima kan overnemen. Zodra we een bewijs formeel opgeschreven hebben, kan een computerprogramma dit bewijs *checken*. Dit is de boekhouder uit de titel van de oratie, die één voor één de stapjes nagaat en kijkt of het klopt. De Nederlandse wiskundige De Bruijn wist dit eind 60-er jaren al en in het door hem geleide Automath-project werden de eerste *bewijscheckers* ontwikkeld. Bij de Automath-systemen typte de gebruiker een bewijs in, in de speciale syntaxis van het systeem, en de Automath-bewijschecker checkte dan of het bewijs klopte. De huidige systemen werken volgens hetzelfde idee, maar nu helpt het systeem ook om het bewijs te maken en daarom spreken we nu van een *bewijsassistent*.

Zijn we dan nu zover dat we met de huidige bewijsassistenten een flink deel van de wiskunde – definities, bewijzen, berekeningen, ... – kunnen formaliseren? Het formaliseren van alle wiskunde is als doel beschreven in het in 1994 op de CADE conferentie verschenen 'QED Manifesto':

> QED is the very tentative title of a project to build a computer system that effectively represents all important mathematical knowledge and techniques. The QED system will conform to the highest standards of mathematical rigor, including the use of strict formality in the internal representation of knowledge and the use of mechanical methods to check proofs of the correctness of all entries in the system.

In het QED-manifest worden negen redenen opgesomd als motivatie voor het QED project. De eerste twee daarvan zijn in mijn ogen het meest relevant.

1. De wiskunde heeft zo'n grote omvang gekregen dat het niet meer mogelijk is om een overzicht te hebben van alle relevante wiskunde. Een geformaliseerde bibliotheek moet het mogelijk maken snel te zoeken naar relevante resultaten.

2. Bij het ontwerpen van nieuwe hoog-technologische systemen, zoals software voor een automatische piloot, gebruikt men gecompliceerde wiskundige modellen. Het QED systeem kan een belangrijke component zijn voor het modelleren, ontwikkelen en verifiëren van zulke systemen.

De doelen van QED zijn ambitieus en er is sinds 1994 wel enigevooruitgang geboekt, vooral op het tweede punt, maar niet zoveel. Wat het eerste punt betreft: er zijn zeer indrukwekkende formalisaties gedaan en het volume van geformaliseerde wiskunde is zeer sterk uitgebreid, maar we kunnen niet spreken van één coherente formele bibliotheek.

Is het dan te ambitieus? Ja, op dit moment wel. Alle wiskunde formaliseren is sowieso niet realistisch. Een gemotiveerde berekening van Wiedijk komt uit op 140 manjaar die nodig zijn om het standaard curriculum van een wiskunde studie te formaliseren. Dat is veel en gaat de onderzoeksbudgetten van onze universiteiten ver te boven. Maar we moeten ook erkennen dat de huidige bewijsassistenten absoluut niet goed genoeg zijn om vlot een stuk wiskunde te formaliseren. In dit kader is het instructief te lezen wat de schrijvers van het QED-manifest meenden dat er gedaan moest worden. Allereerst zou een groep enthousiaste wetenschappers zich moeten verzamelen en met elkaar vaststellen welke delen van de wiskunde geformaliseerd moeten worden, in welke volgorde en met welke verbanden. De manifestschrijvers gaan er daarbij vanuit dat deze fase misschien jaren zal vergen en dat er misschien een reorganisatie van de wiskunde zelf voor nodig is alvorens echt met het formalisatie werk begonnen kan worden. Andere punten in dit 'to do'-lijstje zijn van een soortgelijke top-down, organisatorische aard.

In mijn optiek is dit een foute benadering van de problematiek. Ontwikkelingen als Wikipedia laten zien dat juist een 'bottom up'-gedistribueerde benadering werkt, met een zeer lichtgewichte basistechnologie. Men zou kunnen vermoeden dat zo'n benadering voor het formaliseren van wiskunde niet werkt, maar dat vermoedde men van Wikipedia ook: Wikipedia is typisch

iets dat in theorie niet werkt, maar in de praktijk wel. Het probleem is dat we de lichtgewicht basistechnologie voor het formaliseren van wiskunde nog niet hebben. Het is niet zo dat iedereen met een browser van waar ook ter wereld eenvoudig kan bijdragen aan een gezamenlijke geformaliseerde wiskunde bibliotheek.

Om deze technologie mogelijk te maken is er in onze groep een web interface voor de bewijsassistent Coq gemaakt. In principe kan iedereen met een internet-verbinding eenvoudig – zonder een bewijsassistent te installeren – bijdragen aan één gezamenlijke bibliotheek die op onze server in Nijmegen staat. Deze web interface wordt momenteel uitgebreid tot een zogenaamde 'MathWiki'. Ook op andere universiteiten wordt gewerkt aan zo'n Wikipedia voor de wiskunde.

Bewijsverificatie

Wiskundige bewijzen worden steeds complexer. Dat is onvermijdelijk, want er zijn altijd korte stellingen met lange bewijzen. Dat kun je zelfs bewijzen: er is geen bovengrens aan de verhouding

$$\frac{\text{lengte van het kortste bewijs van } A}{\text{lengte van } A}$$

Het zou natuurlijk kunnen dat korte stellingen met heel lange bewijzen allemaal oninteressant zijn, maar er is op voorhand geen reden dat aan te nemen, en bovendien is 'interessant' een kwestie van smaak.

Onlangs zijn er wiskundige bewijzen gegeven die ook daadwerkelijk zo groot zijn dat ze niet eenvoudig door een mens geverifieerd kunnen worden. Het bekendste voorbeeld daarvan is Hales' bewijs van het vermoeden van Kepler uit 1611. Het vermoeden van Kepler zegt dat de voor de hand liggende stapeling van bollen in de ruimte ook de optimale is. Anders gezegd: de manier waarop de groenteman zijn sinaasappels stapelt is inderdaad de manier waarop de meeste sinaasappels in een krat gaan.

Dit bewijs, door Hales gegeven in 1998 beslaat driehonderd pagina's en werd aan de Annals of Mathematics ter publicatie aangeboden. Na vijf jaar van 'peer reviewing' was de conclusie dat het bewijs voor 99 procent correct was. Wat was het probleem?

In zijn bewijs reduceert Hales het probleem door middel van afschattingen tot een collectie van 1039 gecompliceerde ongelijkheden. Om deze

ongelijkheden te verifiëren schreef hij computer programma's die met intervalrekenkunde de geldigheid nagingen. De referenten hadden hier problemen mee: zelf alle ongelijkheden checken ging uiteraard te ver: een week per ongelijkheid is zo'n vijfentwintig manjaren werk. Het enige wat ze hadden kunnen doen is nagaan of de programma's correct waren, maar dat hebben ze niet gedaan.

Naar aanleiding hiervan is Hales tot de conclusie gekomen dat het bewijs op een computer geformaliseerd zou moeten worden. Daartoe heeft hij het Flyspeck project opgezet. In Flyspeck worden de programma's om ongelijkheden na te gaan formeel opgeschreven. Deze programma's worden correct bewezen ten opzichte van een geformaliseerde wiskundige bibliotheek.

Software- en hardwarecorrectheid

Alle systemen die ontworpen en gebouwd worden, moeten aan specifieke eisen voldoen. Bruggen, vliegtuigen, lampen, en zo ook computers en computer programma's. Bij het ontwerp en de bouw van al deze systemen hoort dus een gedegen verificatiefase waarin het ontwerp en het eindproduct tegen het licht gehouden worden.

Bij bruggen, vliegtuigen en lampen is er een hele geschiedenis van ervaring en ambachtelijkheid die samen met nieuw ontwikkelde wetenschappelijke en technische knowhow geleid hebben tot een gedegen ontwikkeltraject van deze producten.

Voor computers en computer software zijn er onderhand ook gedegen ontwikkeltrajecten die op allerlei manieren fouten trachten te voorkomen. Toch ontstaan er soms grote problemen door computerfouten. Is de situatie met computers en software dus veel slechter gesteld dan met andere producten? Het mag opmerkelijk heten dat men op software geen garantie krijgt.

Computer systemen zijn niet alleen zeer complex, belangrijk is ook dat een kleine fout zeer grote gevolgen kan hebben. Dat kan natuurlijk bij een ander product ook, maar er zijn essentiële verschillen:

- Omdat het discrete systemen zijn is er bij computers geen 'veiligheidsmarge': bijna goed bestaat niet. Een brug die berekend is op een maximale last van 100 ton en een levensduur van 40 jaar, maar feitelijk maar 95 ton aan kan en 37 jaar meekan, zal in de praktijk doorgaans geen problemen opleveren.

- Bij computers zijn er mensen die actief op zoek gaan naar fouten in de systemen, de *hackers*. Zij proberen een klein foutje uit te buiten in hun voordeel.

- Computer systemen worden zeer snel verspreid. Dat geldt zowel voor hardware als voor software. In de jaren tachtig werd de nieuwste Intel chip eerst in een relatief kleine oplage gemaakt en verspreid. In 1994 bevatte de nieuwste Pentium chip een fout, zoals ik al meldde in begin van mijn verhaal. Deze werd in miljoenvoud gemaakt en dan is het niet zo eenvoudig deze allemaal te vervangen. Software wordt tegenwoordig via het internet verspreid en daarmee ook de fouten in die software. Ook de laatste informatie over fouten, zowel in software als in hardware, wordt via het internet verspreid: een foutje blijft niet lang onopgemerkt en heeft dus grote consequenties voor het verantwoordelijke bedrijf.

Hoe kan computer-ondersteund redeneren een bijdrage leveren aan het voorkomen van fouten? Uiteraard door software en hardware correct te bewijzen. In de informatica gebruiken we de term 'formele methoden' voor het samenstel van logische en wiskundige methoden en technieken om informatica fenomenen te modelleren, ontwerpen en verfiëren. De kracht van de formele methode zit erin dat eigenschappen op een abstracte manier uitgedrukt en bestudeerd kunnen worden. Zo'n methode wordt pas echt bruikbaar als ze ondersteund wordt door een 'tool', een computerprogramma dat ons in staat stelt om eenvoudig fenomenen te modelleren en hierover eigenschappen af te leiden.

Tools voor formele methoden zijn vaak zeer specifiek voor een bepaalde methode, en daarom ook vaak snel en bruikbaar. Bewijsassistenten zijn feitelijk juist zeer generieke tools voor formele methoden. In een bewijsassistent kan men allerlei formele methoden implementeren en deze zo ondersteuning bieden. Dit zal doorgaans minder snel en gebruikersvriendelijk zijn dan een specifieke tool. Voordeel is wel dat men in een bewijsassistent veel meer kan modelleren, bijvoorbeeld ook de omgeving waarin de methode werkt.

Mijn onderzoek

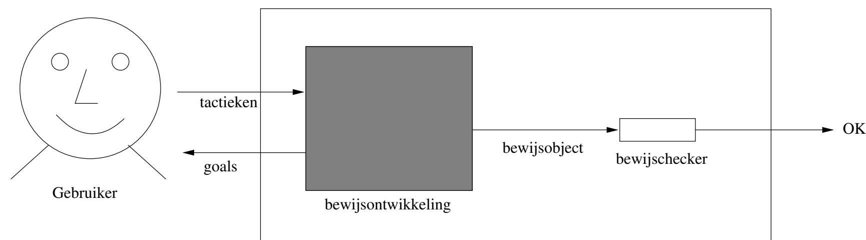
Mijn onderzoek bevindt zich op een breed gebied van computer-ondersteund redeneren, van het bestuderen van fundamentele theorieën, met name type-

theorie en logica, tot het toepassen van redeneersystemen op verificatieproblemen.

Grondslagen

De onderliggende theorieën voor de bewijsverificatiesystemen die we gebruiken, zijn typetheorie, λ-calculus, termherschrijven en logica. De λ-calculus geeft een eenvoudig notatiesysteem voor functies. Termherschrijven en 'pattern matching' geven een eenvoudig en intuïtief mechanisme om functies te definiëren en ermee te rekenen. Logica geeft de theorie van bewijzen. Type theorie geeft een syntactische notie van 'verzameling'.

Een belangrijk aspect van typetheorie als systeem voor bewijsverificatie is het *formules-als-types*-isomorfisme van Curry, Howard en De Bruijn. Dit identificeert een formule uit de logica met het type van zijn bewijzen. De voordelen hiervan zijn dat bewijzen termen worden, dus objecten om eenvoudig te manipuleren en dat bewijschecken hetzelfde wordt als typechecken. Het belangrijkste is echter dat bewijzen objecten zijn die onafhankelijk van de bewijsassistent gecheckt kunnen worden. Dit geeft aanleiding tot de architectuur in het plaatje 1.



Figuur 1: Bewijsassistent en zijn componenten

Het systeem genereert in interactie met de gebruiker een *bewijsobject*, doorgaans een term. Hiervoor kunnen heel ingewikkelde beslissingsprocedures en *tactieken* gebruikt worden, het maakt niet uit, als er maar een bewijsterm geproduceerd wordt. Deze bewijsterm kan – onafhankelijk van de bewijsassistent – door een typeringsalgoritme geverifieerd worden. De clou is dat het checken van een bewijs veel eenvoudiger is dan het vinden ervan en dat een criticus eenvoudig zelf een bewijschecker kan programmeren. Het

idee van bewijstermen die onafhankelijk gecheckt kunnen worden is ook het basisprincipe van 'Proof Carrying code'.

Bewijsassistenten

Het systeem dat we in het plaatje boven zagen heet een *bewijsassistent*. Zo'n systeem bewijst interactief met de gebruiker een stelling. We zouden het systeem alles automatisch kunnen laten doen, maar Church en Turing hebben al in 1936 laten zien dat dat niet kan: er is geen algoritme dat, gegeven een formule, bepaalt of daar een bewijs voor is. Anders gezegd: de predicatenlogica is onbeslisbaar. Ondanks het feit dat er geen algemene beslissingsprocedure is, kunnen we natuurlijk wel zoveel mogelijk automatisch doen. Dat werkt heel aardig voor kleine tussenresultaten, maar voor iets grotere formules werkt het niet. De zoekruimte aan mogelijkheden wordt te groot.

Dus we zijn veroordeeld tot een situatie waar de gebruiker de belangrijke stappen zet en de computer als een soort boekhouder, bijhoudt waar we zijn, of de stappen kunnen, en tussenresultaten automatisch afleidt.

Een verschil tussen de systemen is de invoertaal.

Invoertaal De invoertaal van een bewijssysteem kan *declaratief* of *procedureel* zijn. Bij een procedurele taal zeg je *wat* de bewijsassistent moet doen. Bij een declaratieve taal zeg je *waar* de bewijsassistent heen moet gaan. Dat lijkt misschien bijna hetzelfde, maar dat is het niet. Dit kunnen we illustreren aan de hand van een routebeschrijving. Om van Comeniuslaan 2, waar we nu zijn, naar mijn huis te gaan zou je als instructies kunnen geven

vertrek in oostelijke richting	
na 65 m.	links afslaan
na 120 m.	links afslaan
na 1430 m.	rechts afslaan
na 1640 m.	links afslaan
etcetera	

Dit is een *procedurele* routebeschrijving: er wordt verteld *wat* we moeten doen. Een kenmerk van zo'n beschrijving is dat we er alleen een interpretatie aan kunnen geven door haar *uit te voeren.*'Na 120 m. links afslaan' betekent alleen iets in de toestand waarin we op dat moment zijn. Een gevolg hiervan

is dat we vast zitten, zodra er een fout in de beschrijving zit. Als 'na 120 m. links afslaan' niet kan, bijvoorbeeld omdat de weg opgebroken is, dan hebben we niets meer aan de rest van de beschrijving.

op Comeniuslaan	ga naar Erasmuslaan
op Erasmuslaan	ga naar St. Annastraat
op St. Annastraat	ga naar Grootstalselaan
op Grootstalselaan	ga naar Hatertseweg
etcetera	

Dit is een *declaratieve* routebeschrijving: er wordt verteld *waar* we vervolgens naar toe moeten. Hoe we daar moeten komen is aan ons. Voordeel is dat deze beschrijving *robuuster* is, want als bijvoorbeeld de Grootstalselaan afgesloten is kunnen we op een andere manier naar de Hatertseweg gaan en van daar verder. Nadeel is dat we zelf maar moeten uitzoeken hoe we naar het volgende punt komen. Uiteraard is dat in dit voorbeeld geen probleem.

Een echte routebeschrijving, bijvoorbeeld zoals we die op het web vinden, combineert procedurele en declaratieve elementen. Daardoor bevat ze veel redundantie, maar dat is voor de gebruiker alleen maar plezierig.

In bewijsstijl zien we het onderscheid nog duidelijker. Hier een geformaliseerd bewijs in *procedurele stijl* van de simpele stelling dat als we een getal verdubbelen en dan door 2 delen, we weer hetzelfde getal terug krijgen. Dit bewijs is van Théry.

```
Theorem double_div2: forall (n : nat),  div2 (double n) = n.
simple induction n; auto with arith.
intros n0 H.
rewrite double_S; pattern n0 at 2; rewrite <- H; simpl; auto.
Qed.
```

U kunt niet zien wat dit bewijs doet. Ik ook niet, want het heeft alleen betekenis als we het uitvoeren in Coq. Dan zien we wat de *bewijstoestand* is na regel 3 en dan snappen we waarom regel 4 een zinvolle vervolgstap is en wat hij doet.

Hier een bewijs in *declaratieve stijl* van dezelfde stelling. Dit bewijs is van Corbineau.

```
Theorem double_div2: forall (n : nat), div2 (double n) = n.
```

```
proof.
  assume n:nat.
  per induction on n.
    suppose it is 0.
      thus thesis.
    suppose it is (S m) and Hrec:thesis for m.
      have (div2 (double (S m))= div2 (S (S (double m)))).
           ~= (S (div2 (double m))).
      thus ~= (S m) by Hrec.
  end induction.
end proof.
Qed.
```

We kunnen zien wat dit bewijs doet, ook zonder het in Coq uit te voeren. U misschien niet, maar iemand die wat wiskunde gestudeerd heeft en zich een beetje in de syntax verdiept ziet meteen wat hier staat.

Aan dit voorbeeld zien we nog een aspect van declaratieve en procedurele bewijzen: het declaratieve bewijs is langer. Dat blijkt algemeen zo te zijn. Plezierig voor de lezer, maar vervelend voor degene die het in moet typen. In een declaratief bewijs staat informatie die het systeem ook zelf kan genereren. Bij een procedureel bewijs geeft men de minimale hoeveelheid informatie die de volgende beoogde bewijstoestand oplevert. Het is dus zaak om de twee stijlen zo veel mogelijk te combineren: de gebruiker wil zo min mogelijk hoeven in te typen, maar ziet wel graag een leesbaar bewijsscript.

Formele bibliotheken

Het is belangrijk om een echte grote geformaliseerde wiskundige *bibliotheek* op te zetten. Zo'n bibliotheek is absoluut noodzakelijk om wiskundigen geïnteresseerd te krijgen in het formaliseren van hun bewijzen. De basiskennis waar ze zelf niet meer over hoeven na te denken moet gewoon beschikbaar zijn. Zoals reeds gezegd is het maken van zo'n bibliotheek bepaald geen sinecure.

Mexicaanse hoed Een nadeel van het werk aan een formele bibliotheek is verder dat het niet zo tot de verbeelding spreekt. Je publiceert een mooi artikel over de formalisatie van een ingewikkelde stelling, niet over de formalisatie van de cursus Calculus 1. Daarom is een goede benadering de

zogenaamde 'Mexicaanse Hoed'-benadering. Hierbij richten we ons op een grote 'hoofdstelling', maar resultaten die we bewijzen formuleren we zo algemeen mogelijk en stoppen we in een algemene bibliotheek. Zo ontstaat een brede basis (de rand van de hoed) en een piek met technische ad hoc lemma's voor onze hoofdstelling (de piek van de hoed). Idealiter ontstaan er na verloop van tijd meer pieken op de rand en wordt de rand steeds breder. Doel is dus om de rand zo breed mogelijk en de pieken zo smal mogelijk te houden.

Een ander probleem bij deze bibliotheken is de documentatie. Hoe vertellen we een gebruiker wat hij waar kan vinden? Dat blijkt lastig te zijn. Als we de hele bibliotheek zonder bewijzen op een nette manier, met wiskundige notatie in plaats van asci, printen, dan hebben we een overzicht van alle lemma's maar dat zijn er heel veel, zonder structuur. Bij onze eigen bibliotheek C-CoRN gaat het om 962 definities en 3554 lemma's op 394 pagina's. We willen graag op een hoog niveau overzicht van de inhoud van het materiaal, inclusief motivaties, intuïties en dergelijke. Dit vraagt om een soort 'literate proving' benadering, vergelijkbaar met Knuths literate programming , waarbij de documentatie en de formele bewijzen in één systeem en één bestand geschreven worden.

Interactieve wiskundige documenten en interfaces

Idealiter zou men een wiskundig document uit een formalisatie kunnen extraheren, maar zo simpel ligt het niet. Het kan wel, maar de uitkomst is een vrij directe 'pretty printed' vertaling van de computer code. We kunnen wel wat details onderdrukken zodat we alleen de belangrijkste ingrediënten zien, maar het is moeilijk om in het algemeen te bepalen wat belangrijk is.

Om dichter te komen bij het idee van literate proving hebben we het TexMacs systeem zo uitgebreid dat we zowel een wiskundig document kunnen schrijven (in een LaTeX-achtige stijl) en tegelijk een Coq-formalisatie van de wiskunde kunnen doen . Het systeem is nog in ontwikkeling, maar is veelbelovend.

Hybride systemen

Een interessante toepassing in de informatica waaraan we sinds kort werken is hybride systemen. Een hybride systeem bevat zowel continue componenten,

zoals een klok, een thermometer of een snelheidsmeter, als discrete componenten, zoals een gaskraan die in drie standen gezet kan worden. De software die zo'n systeem bestuurt moet op basis van invoergegevens van sensoren de gaskraan aansturen, zodat de temperatuur of de snelheid binnen een bepaalde bandbreedte blijft. Interessant hieraan is dat we voor het correct bewijzen van de besturingssoftware ook de omgeving moeten modelleren, met differentiaalvergelijkingen. Een ander interessant aspect is dat de toestandsruimte overaftelbaar is en we dus alleen na een discrete abstractie onze geautomatiseerde tools, zoals 'model checkers', toe kunnen passen. We willen nagaan in hoeverre we dit modelleren en abstraheren, en mogelijk ook het doorrekenen van de abstracte toestandsruimte, in een bewijsassistent kunnen doen, gebruik makend van onze formele bibliotheek van reëele getallen.

Onderwijs

Het zal duidelijk zijn dat het computer-ondersteund redeneren een zeer interactieve bezigheid is. De computer is actief, maar de gebruiker heel nadrukkelijk ook. We verwachten soms wonderen van computers, maar wonderen komen er alleen uit als we die er zelf eerst ingestopt hebben. Dat geldt niet alleen voor bewijsassistenten, maar voor alle computer tools, dus ook voor tools die formele methoden ondersteunen.

Dus, beste studenten, wees actief met de theorie en met de tools die ze ondersteunen. Laat de computer voor je werken, maar bedenk wel dat je zelf altijd de eerste actie in gang moet zetten.

Het actief met de computer bezig te zijn wordt gelukkig ook nog steeds ondersteund door ons taalgebruik. We zitten *achter* of aan de computer, terwijl we *voor* de televisie zitten. Het ergens 'achter ' zitten wil zeggen dat je ermee werkt. Je *werkt met* de computer en je *kijkt naar* de tv. Zo zit een producer ook *achter* (of aan) de knoppen en iemand zit *achter* zijn bureau, maar *voor* het raam.

Ik ben een pleitbezorger van het gebruik van wiskundige en logische methoden in de informatica en informatiekunde. Abstractie is een groot goed. Het is belangrijk dat we studenten de methoden aanreiken om die abstracties te maken en dat we ze leren dat je met die abstracte theorie ook echt iets beter kunt begrijpen, kunt berekenen of beredeneren. Wiskunde en logica leert men op de universiteit. Een document schrijven of presenteren kun je altijd nog leren. Moeten de studenten dan niet leren presenteren? Jawel,

maar ze moeten vooral leren *inhoud* te presenteren. Dat zie ik ook als het belangrijkste doel van onze research lab vakken: leren inhoud te verwerken en uit te leggen aan anderen.

Ik hoor wel eens de klacht dat studenten dommer zouden zijn dan vroeger. Dat is niet mijn ervaring. Wel hebben de studenten van nu een andere achtergrond en vooropleiding dan twintig jaar geleden. Een collega vatte het recentelijk zo samen:

> 'Students get more excited by stuff that works than by stuff one has to think about.'

Dat geldt niet voor alle studenten, maar er zit wel een kern van waarheid in. Aan ons docenten de taak te laten zien dat de theorie werkt. Niets is zo praktisch als een goede theorie.

Dankwoord

Ik wil iedereen met wie ik in al die jaren heb samengewerkt van harte bedanken voor de inzichten die ze met mij gedeeld hebben. De academische wereld is bijzonder in de zin dat de mensen die er werken zeer gefocust zijn op inhoud. Ik ben erg gesteld op die bijzonderheid.

Mijn academische loopbaan heeft zich in Eindhoven en Nijmegen afgespeeld. In Nijmegen wil ik de wiskunde staf bedanken die me de wiskundige basis heeft geleerd, in het bijzonder Wim Veldman die me attent maakte op de filosofische aspecten van de wiskunde en me logica onderwees. In Eindhoven dank ik in het bijzonder de Formele Methoden-groep waar het plezierig en inspirerend toeven was. Van Jos Baeten heb ik geleerd hoe je invloed uitoefent en een afdeling bestuurt. Rob Nederpelt dank ik voor het vertrouwen dat hij in mij stelde door me naar Eindhoven te halen, maar vooral voor de wijze lessen in het omgaan met mensen en het begeleiden van promovendi.

Het grootste deel van mijn academische leven heeft zich hier in Nijmegen afgespeeld bij de subfaculteit informatica, nu ICIS. Ook iedereen hier bedankt voor de inspirerende omgeving. Alle leden en oudleden van mijn eigen afdeling Grondslagen wil ik dank zeggen voor een plezierige sociale omgeving en een dynamisch wetenschappelijk klimaat. Eén persoon wil ik in het bijzonder noemen en dat is mijn promotor Henk Barendregt. Hij liet me al tijdens mijn studie zien hoe je moeilijke dingen makkelijk kunt zeggen en hoe je door abstractie kunt doordringen tot de kern van de zaak. Zijn niet

aflatende wetenschappelijke honger en zijn focus op inhoud zullen altijd een voorbeeld voor me zijn.

Tot slot een woord van dank voor mijn vrienden en familieleden. Sociale context is zeer belangrijk en het is ook belangrijk om even iets anders te doen dan werken, op de fiets, aan een diner, in een café of waar dan ook. Dank voor de fijne momenten, de nodige reflectie en de steun. Rob, bedankt voor je filosofische noten en relativeringen. In het bijzonder dank aan mijn gezin, Judith, Wouter, Koen en vooral Monique, bij wie het fijn thuiskomen is.

Ik heb gezegd