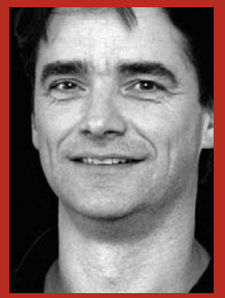
Computer-ondersteund redeneren: de boekhouder steunt de denker

INAUGURELE REDE DOOR PROF. DR. HERMAN GEUVERS

Radboud Universiteit Nijmegen

INAUGURELE REDE
PROF. DR. HERMAN GEUVERS



Dat computers fouten maken, weet iedereen die wel eens achter een pc werkt. Soms zijn deze fouten niet alleen vervelend, maar hebben ze ook verstrekkende gevolgen. Met als meest beruchte voorbeeld de Ariane 5 raket die kort na lancering volledig uit koers raakte en zichzelf opblies als gevolg van een softwarefout. In zijn oratie als hoogleraar Computerondersteund redeneren maakt Herman Geuvers duidelijk dat het correct bewijzen van software en hardware een complexe zaak is. Dit kan eigenlijk niet meer zonder een geautomatiseerde bewijsassistent. Deze checkt als een boekhouder een voor een alle stapjes. Maar de boekhouder heeft wel degelijk ook een denker nodig: iemand die de computer de weg wijst en bewijsstappen in geformaliseerde vorm voert. Eigenlijk zou alle wiskundige basiskennis geformaliseerd in een grote bibliotheek beschikbaar en bruikbaar moeten zijn. Om deze megaklus te klaren pleit Geuvers voor een Wikipedia van geformaliseerde wiskunde.

Herman Geuvers (1964) studeerde wiskunde aan de Radboud Universiteit Nijmegen en behaalde in 1988 cum laude het doctoraalexamen met specialisatie in de logica en grondslagen van de wiskunde. In 1993 volgde de promotie te Nijmegen. Van 1993 tot eind 1999 was hij universitair docent in de groep Formele Methoden van de TU Eindhoven. Sinds 2000 werkt hij weer in Nijmegen, eerst als universitair hoofddocent en sinds 2006 als hoogleraar Computerondersteund redeneren bij het Institute for Computing and Information Sciences.

Radboud Universiteit Nijmegen

COMPUTER-ONDERSTEUND REDENEREN: DE BOEKHOUDER STEUNT DE DENKER

Computer-ondersteund redeneren: de boekhouder steunt de denker

Rede in verkorte vorm 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

door prof. dr. Herman Geuvers

Vormgeving en opmaak: Nies en Partners bno, Nijmegen
Drukwerk: Thieme MediaCenter Nijmegen

ISBN 978-90-9021687-4

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 back-ups. 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[1]. 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', een fout in de software. 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[2]. 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 (wat moeten we nog doen) 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.'[3]

Gates heeft het hier over de bij Microsoft ontwikkelde Static Driver Verifier [Ball et al. 2006]. 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 [Muñoz Dowek 2005] gebruikt bewijsassistenten om software voor de luchtverkeersleiding te verifiëren. Intel gebruikt bewijsassistenten bij het verifiëren van de nieuwe chips [Harrison 2000].

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 – de bewijsassistent – 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.

De Nederlandse wiskundige De Bruijn (zie [Nederpelt et al. 1994]) brak al eind jaren zestig een lans voor volledig geformaliseerde bewijzen. Zijn idee was dat een wiskundige een bewijs zou geven dat anderen, bijvoorbeeld studenten in hun masterfase, in een computer zouden kunnen invoeren, die het vervolgens zou verifiëren. Bij deze laatste stap zouden dan de details verder uitgewerkt moeten worden.

De Bruijn heeft hiervoor ook systemen ontwikkeld, maar er was vanuit de wiskundige wereld niet veel interesse in het formaliseren van bewijzen. Het formaliseren had geen toegevoegde waarde. Verder waren de computers uit die tijd – de pc bestond nog niet – gebruikersonvriendelijk en qua capaciteit ongeschikt voor dit soort grote

taken. Deze situatie is sindsdien wel enigszins veranderd. We hebben nu pc's en wiskundigen zijn net als iedereen meer gewend geraakt aan het gebruik van computers in onderzoek, onder meer voor type setting, voor (symbolisch) rekenen en visualisatie. Eerst wil ik nu wat dieper ingaan op de status van een wiskundig bewijs.

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. De vraag dringt zich op waarom dit allemaal werkt.

- Welke kleine redeneerstapjes zijn correct? En is iedereen het daarover eens?
- Waarom leveren al die kleine redeneerstapjes bij elkaar een stelling op die *absoluut* waar is?

De eerste vraag is eenvoudig te beantwoorden. Er zijn precieze formele regels die voorschrijven wanneer een redeneerstap correct is. Bij een spel als schaken zijn er regels voor de correctheid van een zet. Net zo zijn er bij bewijzen regels voor de correctheid van een redeneerstap. Dit verplaatst de kwestie van correctheid naar het vertalen van *informele* redeneerstappen naar *formele* redeneerstappen, dat wil zeggen, van de natuurlijke taal naar de formele formuletaal. Dit wordt echter in de wiskunde niet als het grootste probleem gezien, want wiskundigen schrijven bijna alles al in formules en gebruiken in bewijzen een vrij beperkt technisch jargon. Er is wel enige discussie over *welke* kleine redeneerstapjes dan de juiste zijn. Dat is niet alleen een praktische kwestie (welke collectie kleine redeneerstappen is het makkelijkst om een bewijs naartoe te vertalen?) maar ook een inhoudelijke: welke stappen zijn *correct*? Hoewel er vrij grote consensus is onder wiskundigen is toch niet iedereen het hier over eens. Dit mag opmerkelijk lijken, maar is het toch niet. De wiskundige *realiteit* ligt niet zo absoluut vast als men misschien zou denken. Maar hier komen we op het gebied van de filosofie en grondslagen van de wiskunde, en daar wilde ik het bij deze oratie niet over hebben.

Het cruciale punt is natuurlijk welke redeneerstapjes *waar* zijn, dat wil zeggen welke stapjes leveren resultaten op die waar zijn in de realiteit. Hiermee zijn we meteen bij de tweede vraag. Waarom is alles wat we met een wiskundig bewijs bewijzen ook echt waar? De fysicus en Nobelprijs winnaar Eugene Wigner noemde dit 'de onredelijke effectiviteit van de wiskunde in de natuurwetenschappen' en ook Albert Einstein was zich bewust van deze opmerkelijke situatie: de wiskunde lijkt terug te voeren op een formeel spel met precieze regels. Maar deze regels zijn niet zo willekeurig, want alles wat er met deze spelregels afgeleid kan worden, blijkt waar te zijn. Dit is een interessant wetenschapsfilosofisch vraagstuk dat ik hier echter terzijde zal leggen.

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. Hierover later meer. 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?

In een bewijs dat we in een artikel of boek vinden worden beide 'rollen' door elkaar geweven. Er wordt intuïtie gegeven en er wordt uitgelegd hoe we kunnen inzien en verifiëren dat de stelling geldt.

De wiskundige Paul Halmos benadrukt dat een wiskundig bewijs niet op dezelfde manier wordt opgeschreven als het gevonden is:

> Mathematics is not a deductive science – that's a cliche. When you try to prove a theorem, you don't just list the hypotheses, and then start to reason. What you do is trial and error, experimentation, guesswork. [Halmos 1985]

Een bewijs heeft eigenlijk twee stadia:
1. De bewijsconstructie ofwel het zoeken naar een bewijs. Hierbij is van alles mogelijk, gokken, experimenteren, uitproberen.
2. Het opgeschreven bewijs. Dit bevat uitleg waarom het gestelde geldt, waarom het bewijs er zo uit ziet (punt B hierboven) en het bevat bewijsstappen die één voor één geverifieerd kunnen worden (punt A hierboven).

Dat een bewijs ook uitleg geeft en vooral ook inzicht verschaft blijkt uit het feit dat er voor één stelling vaak meer bewijzen zijn. Gedeeltelijk is dit een kwestie van smaak: niet iedereen vindt hetzelfde bewijs het 'mooist', maar verschillende bewijzen geven ook ieder een eigen invalshoek op de stelling en belichten verschillende aspecten. Een fraai voorbeeld is het boek *Proofs from THE BOOK* [Aigner and Ziegler 2004] dat 'mooie' bewijzen van bekende stellingen presenteert. Het boek bevat bijvoorbeeld zes bewijzen van de stelling van Euclides dat er oneindig veel priemgetallen zijn. Ieder bewijs is interessant op zich, omdat het een bepaald aspect van de wiskunde van priemgetallen belicht.

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 Bruijn wist dit al en in het door hem geleide Automath-project [Nederpelt et al. 1994] 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. De *bewijsassistent* checkt de syntaxis, doet een paar stappen, probeert een paar stappen en houdt de bewijstoestand bij.

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.

Het manifest beschrijft de ambitieuze doelen van het project en bespreekt mogelijke tegenwerpingen en antwoorden hierop. In 1994 en 1995 zijn er nog twee workshops over QED geweest en daarna niet meer. Dat wil niet zeggen dat er daarna niets meer gedaan is. In verschillende projecten wordt gewerkt aan het formaliseren van wiskunde met behulp van bewijsassistenten, waarbij de motivaties van het QED-manifest (meestal impliciet) een uitgangspunt vormen.

In het QED-manifest worden negen redenen opgesomd als motivatie voor het QED project. De eerste drie 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 hoogtechnologische 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.
3. Voor het onderwijs kan het QED systeem cursusmateriaal aanbieden waarmee studenten individueel kunnen oefenen, bijvoorbeeld met interactieve bewijs- of programmeeropgaven.

Deze doelen zijn ambitieus en er is sinds 1994 wel enige vooruitgang 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 een coherente formele bibliotheek. De Mizar Mathematical Library[4] is de enige die enigszins in de buurt komt. Het derde punt tenslotte: er zijn wat kleinere projecten om bewijsassistenten in het onderwijs in te zetten, maar een grootschalig gebruik van een wiskundige bibliotheek in het onderwijs is er niet.

Is het dan te ambitieus? Ja, op dit moment wel. Alle wiskunde formaliseren is sowieso niet realistisch, 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 topdown, 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.

Wat moet er dan wel gebeuren, gegeven het gebrek aan een Wikipedia-achtige simpele basistechnologie? Volgens mij gaat het hierbij om de volgende zaken:

- Bewijsassistenten verder ontwikkelen, zodat we toewerken naar een eenvoudige basistechnologie die voor iedereen bruikbaar is. Op dit moment zijn er veel verschillende bewijsassistenten. Dat is goed. Competitie leidt tot nieuwe ideeën en verbetering, door het overnemen en verbeteren van ideeën van anderen. Belangrijke punten die verdere ontwikkeling behoeven zijn: bewijsautomatisering en goede interfaces.
- Grote formalisaties doen en op basis daarvan feedback geven. De systemen worden alleen maar beter door ze echt te gebruiken en zo de tekortkomingen te ervaren.
- Een basisbibliotheek opbouwen, waarbij met name ook gelet moet worden op samenhang, bruikbaarheid en documentatie. In hoeverre is de bibliotheek bruikbaar voor een nieuwkomer die een stelling wil formaliseren met behulp van basiswiskunde uit de bibliotheek?
- Toepassingen op basis van de bibliotheek. Kunnen we de bewijsassistent en de bibliotheek gebruiken bij het modelleren, ontwerpen en verifiëren van een nieuw product, bijvoorbeeld een netwerkprotocol of een aansturing van een robotarm?

Op basis van dit gezamenlijke werk kan op termijn een QED-achtig systeem ontstaan. Het grootste risico is dat men te snel wonderen verwacht. In dit kader is het interessant om eens na te denken over de hoeveelheid werk die gepaard gaat met het creëren van een geformaliseerde bibliotheek. Een gemotiveerde berekening van Wiedijk[5] 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. Dat wil niet zeggen dat het onmogelijk is. Ontwikkelingen als Linux en Wikipedia laten zien dat een gedistribueerde, goed georganiseerde opzet veel kan bewerkstelligen.

BEWIJSVERIFICATIE

Wiskundige bewijzen worden steeds complexer. Dat is onvermijdelijk, want er zijn altijd korte stellingen met lange bewijzen. Dat kun je zelfs bewijzen, iets preciezer: 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 [Hales 2005]. 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. In zijn originele bewijs gebruikt Hales computer programma's om ongelijkheden na te gaan. Wat is daar eigenlijk mis mee?

1. Zijn programma's doen misschien iets heel anders dan ongelijkheden checken. Dus misschien geven ze 'true' als uitkomst van een check, terwijl de ongelijkheid niet geldt. Om dit te voorkomen moet de programmacode geverifieerd worden: doet deze echt wat hij verondersteld wordt te doen.
2. Als de programmacode correct is, kan het nog zo zijn dat het compileren van de programmacode fout gaat, of dat het besturingssysteem fouten maakt, of dat de hardware fouten maakt, waardoor er een antwoord 'true' komt terwijl een ongelijkheid niet geldt.

Het eerste is het probleem van softwarecorrectheid. Hier komen we straks op terug. Het ware in principe mogelijk voor de referenten van Hales' bewijs om de programmacode te controleren, maar die optie is niet in ogenschouw genomen.

Het tweede is het probleem dat de programmacode foutief verwerkt zou kunnen worden. Dit behelst ook 'toevallige fouten'. Dit probleem kan men ondervangen door de software met verschillende compilers naar executeerbare code te vertalen en op verschillende platformen uit te voeren. Als al deze compilers en platformen hetzelfde antwoord 'true' geven, dan is het zeer onwaarschijnlijk dat ze allemaal precies dezelfde fout veroorzaken waardoor het antwoord eigenlijk 'false' zou moeten zijn. Zo introduceert men een vorm van 'peer review' voor programma-executie: het is bij 'peer review' mogelijk dat iemand iets over het hoofd ziet (dat gebeurt zelfs best vaak), maar de kans dat velen hetzelfde over het hoofd zien is erg onwaarschijnlijk.

Terug naar Hales en het door hem bewezen vermoeden van Kepler. Omdat de wiskundige wereld zijn bewijs niet accepteerde heeft Hales het *Flyspeck*-project opgezet[6], met als doel om zijn bewijs volledig te formaliseren. Met de bewijsassistenten HOL Light[7], Coq[8] en Isabelle[9] zijn onderzoekers bezig om delen van het bewijs volledig te formaliseren. Een ander belangrijk formalisatiewerk is het bewijs van de vier-kleuren-stelling door Gonthier in Coq[10]. Het bewijs van deze stelling bestaat uit het reduceren van het probleem tot 633 gevallen, die dan met algoritmen doorgerekend worden. Gonthier heeft dit alles geformaliseerd in Coq: de reductie tot de 633 gevallen, het definiëren en correct bewijzen van de algoritmen en het uitvoeren van deze algoritmen op de 633 gevallen.

SOFTWARE- EN HARDWARECORRECTHEID

Alle systemen die ontworpen en gebouwd worden, moeten aan specifieke eisen voldoen. Bruggen, vliegtuigen, lampen, veiligheidsspelden, 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 nog steeds soms grote problemen door computerfouten. Iedere computergebruiker is vertrouwd met een computer die 'hangt' of software die 'het niet doet'. Dat we regelmatig onze software moeten updaten, omdat er een veiligheidslek in het besturingssysteem is gevonden, vinden we normaal. 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. Nu zijn software en hardware zeer complexe systemen die voortdurend op allerlei, soms van te voren niet verwachte manieren gebruikt worden. De al genoemde besturingssoftware van de Ariane 5 raket is daar een voorbeeld van. Verder is er nog de wens van 'upward compatibiliteit' en de gebruikers verwachten ook een zekere 'downward compatibiliteit': men wil zijn oude software kunnen draaien op nieuwe computers en het liefst de nieuwste software ook nog op oude machines. Dat je niet met een paard-en-wagen op de snelweg rijdt, of andersom, met een sportauto over een modderige zandweg, vinden we heel normaal, maar van computers verwachten we meer.

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 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. Zo'n formele methode bevat over het algemeen een formele taal, waarin bepaalde objecten en fenomenen kunnen worden uitgedrukt, en een operationele semantiek, die beschrijft hoe deze geformaliseerde objecten zich gedragen. Vaak zal er nog meer zijn, maar dit is het voornaamste. 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.

We hebben al gezien dat bewijsassistenten ook daadwerkelijk in de informatica gebruikt worden bij het correct bewijzen, bijvoorbeeld bij Intel en de NASA. Een ander interessant project is de 'PoplMark Challenge'[11]. Popl is de conferentie Principles of Programming Languages. De auteurs van het artikel [Aydemir et al.] dagen iedereen uit om een artikel over programmeertalen altijd te voorzien van een appendix met computergeverifieerde bewijzen. In het bijzonder zou de metatheorie van een programmeertaal in een bewijsassistent geverifieerd moeten worden. De uitdaging is niet alleen gericht aan de ontwerpers van nieuwe programmeertalen om dit te doen, maar ook aan de ontwikkelaars van bewijsassistenten om dit mogelijk te maken. Vandaar de titel van hun artikel: 'Mechanized metatheory for the masses'. De oproep heeft al het een en ander aan activiteit gegenereerd. Dat men met een bewijsassistent serieuze computer software correct kan bewijzen werd aangetoond door [Blazy et al. 2006], waar een C compiler correct wordt bewezen in Coq.

MIJN ONDERZOEK

Mijn onderzoek bevindt zich op een breed gebied van computer-ondersteund redeneren, van het bestuderen van fundamentele theorieën, met name typetheorie en logica, tot het toepassen van redeneersystemen op verificatieproblemen. Ik zal nu puntsgewijs een overzicht geven van dit onderzoek.

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'. Het is belangrijk een syntactische notie van verzameling te hebben, typen dus, zodat de 'element-van'-relatie beslisbaar blijft. Verzamelingen zijn flexibeler dan types, maar werken met geformaliseerde Zermelo-Fränkel-verzamelingenleer is lastig. Ik denk dat in de wiskunde types fundamenteler zijn dan verzamelingen. Het is ook opmerkelijk dat een bewijs assistent als Mizar, die gebaseerd is op verzamelingenleer, ook een heel verfijnde typetheorie heeft.

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 Figuur 1.

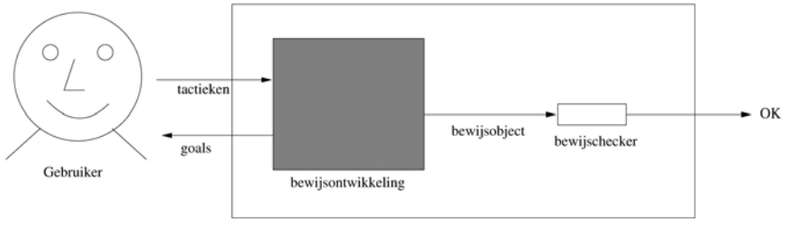


Figure 1: Bewijsassistent en zijn componenten

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. Bewijsassistenten die zich aan bovenstaand principe houden, voldoen aan het zogenaamde *De Bruijn*-criterium (zie [Barendregt Geuvers 2001]). De bewijsassistenten Coq, HOL Light en Isabelle voldoen aan het De Bruijn-criterium. Het idee van bewijstermen die onafhankelijk gecheckt kunnen worden is ook het basisprincipe van 'Proof Carrying code'.

Er is niet één manier om bewijzen te representeren als termen. Het ligt er ook maar aan of men bewijzen uit een bepaalde logica wil representeren of misschien natuurlijketaalbewijzen. In ieder geval is er een vorm van variabele binding nodig, omdat we willen bewijzen onder locale hypothesen. De gewone simpel getypeerde λ-calculus representeert bewijzen in Gentzen stijl natuurlijke deductie. Een natuurlijke deductie is een afleidingsboom. Voordeel van bomen is dat ze een heldere structuur hebben, maar het nadeel is dat er geen 'sharing' is. Daarom heb ik samen met de promovendus Loeb 'deductiegraphen' ontwikkeld [Geuvers Loeb 2007]. Dit zijn gerichte acyclische graphen die natuurlijke deductie met sharing weergeven.

Zo zijn er op het meest fundamentele niveau nog meer interessante kwesties over de basiscalculi waar we mee werken. Sacerdoti Coen heeft een systeem bestudeerd waarmee we zowel 'forward' als 'backward' bewijzen kunnen representeren, iets wat in pure λ-calculus niet mogelijk is. Hier zitten nog vele vragen.

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 zo veel mogelijk 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.

Nog even voor de duidelijkheid: of er een bewijs van *A* is, is niet beslisbaar, maar de vraag of *p* een bewijs van *A* is, is wel (eenvoudig) beslisbaar.

Er zijn allerlei verschillende bewijsassistenten, zoals Coq[8], PVS[12], Mizar[13], Isabelle[9], HOL4[14], HOL Light[7], ProofPower[15]. Het is niet eenvoudig en soms zelfs onmogelijk om resultaten van één systeem te vertalen naar een ander. Sommige systemen hebben een andere logica waardoor het ten principale onmogelijk is, maar het echte probleem is dat de syntaxis van de systemen en de representatie van wiskundige objecten in deze systemen zo verschillend zijn dat een generieke vertaling ondoenlijk is. Als een generieke vertaling al mogelijk is, dan levert het meestal iets op in het nieuwe systeem wat volstrekt onnatuurlijk is voor de gebruikers van dat systeem.

Het is wel duidelijk dat het hergebruik van resultaten uit verschillende bewijsassistenten erg nodig is om de grote inspanningen die gepaard gaan met het formaliseren enigszins in de hand te houden. Een praktisch voorstel in die richting is de zogenaamde Little Theories benadering [Farmer et al. 1992], waarbij wiskunde ontwikkeld wordt als een netwerk van theorieën die met elkaar verbonden zijn via theorie interpretaties. Het 'Mathematic Components Project' van het nieuwe INRIA-Microsoft onderzoekscentrum in Parijs beoogt iets vergelijkbaars: wiskundige theorieën opbouwen uit componenten, net zoals moderne software is opgebouwd uit modules.

Een verschil tussen de systemen is de mate waarin ze voldoen aan het De Bruijn criterium, maar er zijn ook andere belangrijke verschillen. Enkele daarvan zijn de mate van automatisering en de invoertaal van het systeem.

Automatisering • De automatisering kan ingebouwd zijn in het systeem, zoals met name bij PVS het geval is. Het systeem biedt dan krachtige mogelijkheden om allerlei doelen snel te bewijzen of te reduceren tot andere – hopelijk eenvoudiger – doelen. Andere systemen, zoals Coq, geven de gebruiker de mogelijkheid om zelf automatisering toe te voegen. De wijsheid ligt zoals altijd in het midden. Krachtige ingebouwde beslissingsprocedures zijn handig, maar als ze het probleem niet oplossen of vertalen naar een eenvoudiger probleem, dan staat de gebruiker met lege handen. Systemen als Coq geven de gebruiker de controle over de beslissingsprocedures en wanneer die aangeroepen worden. Nadeel is wel dat die dan eerst geïmplementeerd moeten worden. Kunnen we niet allebei hebben? Dat is moeilijk en tot dusverre niet naar tevredenheid

opgelost. Ingebouwde beslissingsprocedures zijn zo geïmplementeerd en geoptimaliseerd dat ze *in de praktijk* werken. Zodra we de gebruiker toestaan daar zelf beslissingsprocedures aan toe te voegen werkt het niet meer. Het systeem wordt veel te traag of het kan zelfs gebeuren dat problemen die eerst eenvoudig oplosbaar waren dat nu niet meer zijn.

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 230 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. Interessante ideëen hiervoor staan in [Wiedijk 2004].

FORMELE BIBLIOTHEKEN

Als we wiskunde formaliseren kunnen we ons richten op één grote indrukwekkende stelling die we proberen te bewijzen, om te zien of en hoe het systeem dat aankan en wat er eventueel extra nodig is om dit te bewerkstelligen. Voor de PR is dit goed: het maakt indruk om de moeilijke stelling X bewezen te hebben. Een probleem is dat de honderden kleine lemma's die we onderweg bewezen hebben weinig herbruikbaar zijn. Hergebruik is sowieso een probleem van formalisaties. De geformuleerde lemma's zijn soms net niet algemeen genoeg. Of de formalisatie gebruikt heel specifieke definities, handig voor de hoofdstelling, maar niet altijd handig voor algemeen gebruik.

Het is daarom belangrijk om een echte grote geformaliseerde wiskundige *bibliotheek* op te zetten, een samenhangende collectie van basisresultaten die goed toegankelijk en gedocumenteerd is en als zodanig goed herbruikbaar. 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.

Een nadeel van het werk aan een formele bibliotheek is 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 dikker. Doel is dus om de rand zo dik mogelijk en de pieken zo smal mogelijk te houden.

Het lijkt erg voordehandliggend om het zo te doen, maar dat is het niet. Het vereist discipline om steeds bij ieder tussenresultaat een stapje terug te doen en te kijken hoe algemeen het geformuleerd kan worden. Het is twee stappen vooruit en één stap terug en men moet oppassen dat het niet verzandt in één stap vooruit en twee stappen terug.

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 [Cruz-Filipe et al. 2004] gaat het om 962 definities en 3554 lemma's op 394 pagina's. Er zijn zoeksystemen zoals Whelp [Asperti et al. 2006a] die het mogelijk maken op laag niveau lemma's van een bepaalde vorm te zoeken. Dat helpt, maar we willen ook 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 [Knuth 1992], waarbij de documentatie en de formele bewijzen in één systeem en één bestand geschreven worden. Om uit te leggen wat het probleem is laat ik u zien hoe de bewijsontwikkeling in ons eigen FTA-project verliep. Men begint met een wiskundig bewijs van een stelling, met al zoveel mogelijk details ingevuld. Het plan is om dat in twee stappen volledig te formaliseren: eerst de definities en de statements van de lemma's en vervolgens alle bewijzen invullen. Het wiskundig bewijs – het eerste document – fungeert zo tevens als documentatie van de formalisatie.

1	Wiskundig bewijs	Document met veel details (meestal LaTeX bestand)
2	Theorie ontwikkeling	Computer bestand in Bewijsassistent (definities en statements van lemma's)
3	Bewijsontwikkeling	Computer bestand in Bewijsassistent (bewijzen ingevuld)

De werkelijke situatie is dat we voortdurend tussen deze drie fases heen en weer springen: zodra we het bewijs gaan invullen blijkt dat het lemma net iets anders geformuleerd moet worden of dat de definities net niet handig zijn. Dus terwijl we in de derde fase bezig zijn moeten we ook nog regelmatig de bestanden van fases 1 en 2 aanpassen. Maar dat is erg vervelend, want we willen vooruit met het formele bewijs! Dus in de praktijk wordt document 1 na enige tijd niet meer geupdated. De enige manier om dit te laten werken is als we alle drie documenten tegelijk, in één bestand, kunnen maken.

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, zoals het Mowgli project (zie [Asperti et al. 2006a]) heeft aangetoond, 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 [Geuvers Mamane 2006]. Het systeem is nog in ontwikkeling, maar is veelbelovend.

Zoals ik al vermeld heb, is het voor het creëren van momentum essentieel dat we een grote bibliotheek van geformaliseerde wiskunde hebben. Dat kost dus veel tijd en mankracht. Een manier om daar aan te komen is door er een gezamenlijke gedistribueerde inspanning van te maken waaraan iedereen die wil op een eenvoudige manier via het web kan bijdragen, een Wikipedia voor formele wiskunde. Om dit 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'.

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 (zie [Alur et al. 2006]). 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 reële 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 daarvoor je moet wel eerst zelf een actie in gang 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. In verschillende internationale projecten heb ik ook zeer plezierig samengewerkt met Europese partners. Ik wil hier met name de Types-projecten noemen. Dat waren en zijn altijd zeer inspirerende bijeenkomsten waarbij je het gevoel hebt oude vrienden te ontmoeten.

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 voor-

al 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 oud-leden 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

REFERENCES

1 ARIANE 5 Flight 501 Failure, Report by the Inquiry Board, Chairman Prof. J.L. Lions, Paris, 19 July 1996 url: http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

2 Intel, FDIV Replacement Program Statistical Analysis of Floating Point Flaw (Intel White Paper) url: http://support.intel.com/support/processors/pentium/fdiv/wp/

3 Bill Gates, Keynote address bij WinHec 2002, 18 april 2002

4 Mizar Mathematical Library url: http://www.mizar.org/library/

5 Wiedijk, F., Estimating the Cost of a Standard Library for a Mathematical Proof Checker, op: url http://www.cs.ru.nl/ freek/notes/index.html

6 The Flyspeck project, url: http://www.math.pitt.edu/ thales/flyspeck/

7 The HOL Light theorem prover url: http://www.cl.cam.ac.uk/ jrh13/hol-light/

8 The Coq proof assistant, url: http://coq.inria.fr/

9 Isabelle, url: http://isabelle.in.tum.de/

10 Gonthier, G., A computer-checked proof of the Four Colour Theorem, url http://research.microsoft.com/ gonthier/4colproof.pdf

11 The POPLmark Challenge url: http://fling-l.seas.upenn.edu/ plclub/cgi-bin/poplmark

12 PVS Specification and Verification System url: http://pvs.csl.sri.com/

13 Mizar Home Page, url: http://www.mizar.org/

14 HOL4, url:http://hol.sourceforge.net/

15 ProofPower, url: http://www.lemma-one.com/ProofPower/index/

BIBLIOGRAFIE

- [Aigner and Ziegler 2004] Aigner, M. & Ziegler G., *Proofs from THE BOOK*, 3rd edition, Springer 2004.
- [Alur et al. 2006] Alur, R., & Dang, T., & Ivancic, F., Predicate abstraction for reachability analysis of hybrid systems, in: *ACM Transactions on Embedded Computing Systems* (TECS), Volume 5, Issue 1, February 2006, pp. 152 – 199.
- [Asperti et al. 2006a] Asperti, A., & Geuvers, H., & Loeb, I., & Mamane,L., & Sacerdoti Coen,C., An Interactive Algebra Course with Formalised Proofs and Definitions, in: *Proceedings of the Fourth Conference Mathematical Knowledge Management*, *MKM* 2005 LNAI 3863, Springer, 2006, pp. 315–329.
- [Asperti et al. 2006a] Asperti, A., & Guidi, F., & Sacerdoti Coen, C., & Tassi, E., & Zacchiroli, S., A content based mathematical search engine: Whelp, in: *Types for Proofs and Programs International Workshop*, TYPES 2004, Filliatre, J.-C., & Paulin-Mohring, C., & Werner, B., editors, LNCS 3839, Springer, 2006, pp. 17–32.
- [Aydemir et al.] Aydemir, B., & Bohannon, A., & Fairbairn, M., & Foster, J.N., & Pierce, B.C., & Sewell, P., & Vytiniotis, D., & Washburn, G., & Weirich, S., & Zdancewic, S., 'Mechanized Metatheory for the Masses: The PoplMark Challenge', op: [11].
- [Ball et al. 2006] Ball, T., & Bounimova, E., & Cook, B., & Levin, V., & Lichtenberg, J., & McGarvey, C., & Ondrusek, B., & Rajamani, S.K., & Ustuner, A., Thorough Static Analysis of Device Drivers, in: *Proceedings of EuroSys 2006*, ACM SIGOPS.

- [Barendregt Geuvers 2001] Barendregt, H.& Geuvers, H., Proof Assistants using Dependent Type Systems, in: Robinson, A., & Voronkov, A., editors, *Handbook of Automated Reasoning* (Vol 2) , Elsevier 2001, pp. 1149 – 1238 (chapter 18).
- [Blazy et al. 2006] Blazy, S., & Dargaye, Z., & Leroy, X., Formal verification of a C compiler front-end, in: *FM 2006: Int. Symp. on Formal Methods*, LNCS 4085, Springer, 2006, pp. 460–475.
- [Cruz-Filipe et al. 2004] Cruz-Filipe, L., & Geuvers, H., & Wiedijk, F., C-CoRN, the Constructive Coq Repository at Nijmegen, in: Asperti, A., & Bancerek, G., & Trybulec, A., editors, *Mathematical Knowledge Management, Proceedings of MKM 2004*, LNCS 3119, Springer, 2004, pp. 88-103.
- [Farmer et al. 1992] Farmer, W.M., & Guttman, J.D., & Thayer, F.J., Little theories, in: Automated Deduction - CADE-11, ed. D. Kapur, LNCS 607, Springer, 1992, pp. 567–581.
- [Geuvers Loeb 2007] Geuvers, H., & Loeb, I., Natural Deduction via Graphs: Formal Definition and Computation Rules, to appear in: *Mathematical Structures in Computer Science*, Special Issue on Theory and Applications of Term Graph Rewriting, 2007.
- [Geuvers Mamane 2006] Geuvers, H., & Mamane, L., A Document-Oriented Coq Plugin for TeXmacs, in: Libbrecht, P., editor, *Proceedings of the MathUI workshop at the MKM 2006 conference*, Wokingham, UK, 2006, url http://www.activemath.org/ paul/MathUI06/
- [Hales 2005] Hales, Th., A proof of the Kepler conjecture, in: *Annals of Mathematics*, 162, 2005, pp. 1065–1185.
- [Halmos 1985] Halmos, P., *I want to be a Mathematician: An Automathography*, Springer, 1985, 410 pp.
- [Harrison 2000] Harrison, J., Formal verification of IA-64 division algorithms, in: Aagaard, M., & Harrison, J. , editors, *Theorem Proving in Higher Order Logics*, *TPHOLs 2000* LNCS 1869, Springer, 2000, pp. 234–251.
- [Knuth 1992] Knuth, D., *Literate Programming*, Center for the Study of Language and Information, 1992, xvi+368pp. CSLI Lecture Notes, no. 27, Stanford, California.
- [Muñoz Dowek 2005] Muñoz, C., & Dowek, G., Hybrid Verification of an Air Traffic Operational Concept, in: *Proceedings of IEEE ISOLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation*, Columbia, Maryland, 2005.
- [Nederpelt et al. 1994] Nederpelt, R.P., & Geuvers, H., & de Vrijer, R.C., (editors), Selected Papers on Automath, Volume 133 in Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, 1994, pp 1024.
- [Qed 1994] The QED Manifesto, in: *Automated Deduction - CADE 12*, LNAI 814, Springer, 1994, pp. 238-251.
- [Wiedijk 2004] Wiedijk, F., Formal Proof Sketches, In *Types for Proofs and Programs: Third International Workshop, TYPES 2003*, in: Berardi, S., & Coppo, M., & Damiani, F., editors, Springer, 2004, LNCS 3085, pp. 378–393.