Parallel Graphherschrijven

—

Enkele bijdragen aan de theorie, de implementatie en de toepassing ervan.

SAMENVATTING

Een *berekeningsmodel* van een programmeertaal is een formeel model zo dicht mogelijk bij zowel de semantiek als de implementatie waarbij alleen de essentiële aspecten gemodelleerd worden door enkele abstracties te maken. Door zo'n berekeningsmodel is het veel gemakkelijker om te redeneren over de taal en de bijbehorende programma's, vertalers en machines.

Het is natuurlijk heel moeilijk om een ideaal model te vinden dat elk essentieel aspect van een programmeertaal modelleert. Alle klassieke (sequentiële en imperatieve) programmeertalen hebben in feite het zelfde berekeningsmodel: de Turing machine. Het is duidelijk dat dit model alleen de meest fundamentele aspecten van die talen kan modelleren. Dit proefschrift beschrijft onderzoek naar een berekeningsmodel voor een algemeen paradigma van het programmeren (het declaratieve paradigma) en voor één stijl van programmeren in het bijzonder (functioneel programmeren).

In het *declaratieve* paradigma wordt een gewenste berekening uitgedrukt op een statische manier als een lijst van decalaraties met een expressie die geëvalueerd dient te worden. Een programma wordt beschouwd als een executeerbare specificatie. De belangrijkste eigenschap in het declaratieve paradigma is het feit dat een expressie altijd dezelfde betekenis heeft onafhankelijk van de geschiedenis van de berekening (de taal is *referentieel transparant*). Functioneel programmeren is heel aantrekkelijk omdat voornamelijk door het gebruik van hogere orde functies de uitdrukkingskracht van functionele programmeertalen groter is dan die van conventionele talen. Functioneel programmeren lijkt meer op het mathematisch specificeren van het algorithme. Functionele programma's zijn daarom over het algemeen korter dan hun conventionele tegenhangers en dus zijn ze gemakkelijker te verbeteren en te onderhouden.

Oorspronkelijk werd λ-calculus algemeen gebruikt als het berekeningsmodel voor functionele talen. Combinatorische logica heeft echter een eenvoudiger substitutiemechanisme dan de λ-calculus maar het heeft geen pattern matching. Termherschrijfsystemen bevatten wel pattern matching maar termherschrijfsystemen hebben niet de mogelijkheid om sharing direct uit te drukken. Graphherschrijfsystemen combineren al deze aspecten. Daarom zijn graphherschrijf-systemen het onderwerp van onderzoek in dit proefschrift. Graphherschrijfsystemen worden onderzocht in algemene zin en als berekeningsmodel voor functionele talen en hun implementaties.

In hoofdstuk 3 hebben we meer-niveau herschrijfsystemen geïntroduceerd. Van deze systemen is het in de praktijk bewezen dat ze bijzonder nuttig zijn voor hoog niveau specificaties varierend van het specificeren van reductiestrategieën tot het specificeren van de operationele semantiek van een programmeertaal.

Met behulp van termgraphherschrijven (ingevoerd in hoofdstuk 4) zijn enkele fundamentele resultaten behaald op het gebied van het modelleren van termherschrijven met graphherschrijven. *Termgraphherschrijven* betekent dat een TermHerschrijfSysteem (THS) wordt geïnterpreteerd als (getild naar) een GraphHerschrijfSysteem (GHS). De normaalvormen van het GHS, dat zijn graphen, worden ontrafeld tot termen in de THS-wereld. Via termgraphherschrijven wordt bewezen dat het 'sharen' van termen gezond is. Bovendien worden restricties gegeven die de compleetheid garanderen van implementaties die sharing gebruiken. Termgraphherschrijven is een veelbelovend onderwerp voor verder onderzoek.

In hoofdstuk 5 wordt gegeneraliseerd graphherschrijven gedefinieerd. Dit berekeningsmodel is bijzonder krachtig en het is van onafhankelijk belang als algemeen berekeningsmodel voor parallelle architecturen. Lean is een experimentele taal voor het specificeren van berekeningen in termen van graphherschrijvingen. Deze taal is bijzonder krachtig aangezien er slechts weinig restricties zijn op de graph die getransformeerd wordt, en eveneens zijn er weinig restricties op de transformaties die uitgevoerd kunnen worden. Het is de moeite waard gegeneraliseerd graphherschrijven verder te onderzoeken, wat als 'spin-off' programmeertalen oplevert die gebaseerd zijn op deelklassen met specifieke eigenschappen en voordelen.

Clean (beschreven in hoofdstuk 6) is een voorbeeld van zo'n spin-off taal. De taal is gebaseerd op een beperkt soort graphherschrijven (functioneel graphherschrijven). De praktijk heeft uitgewezen dat Clean zeer geschikt is als tussentaal voor functionele talen en sequentiële machine-architecturen. Een efficiënte sequentiële implementatie van een hogere functionele programmeertaal is met behulp van Clean als tussentaal tot stand gekomen.

In hoofdstuk 7 wordt graphherschrijven uitgebreid met lui copiëren en expliciet parallellisme, hetgeen een veelbelovend model oplevert voor los-gekoppelde parallelle evaluatie van functionele programma's. De uitdrukkingskracht en de eigenschappen van die uitbreidingen zullen het in de nabije toekomst mogelijk maken om het potentiële parallellisme in functionele programma's op een algemene manier succesvol te benutten. Met daadwerkelijk parallelle implementaties is een begin gemaakt.

Er dient nog veel onderzoek op het gebied van graphherschrijfsystemen gedaan te worden (taxonomie, typering, strategieën, strictheidsanalyse, implementatietechnieken, parallelle evaluatie, 'garbage' collectie, etcetera). Maar de tot nu toe geboekte resultaten zijn veelbelovend hetgeen verder onderzoek rechtvaardigt op het gebied van de graphherschrijftheorie en naar het indentificeren van speciale klassen van graphherschrijfsystemen. Bovendien zijn daadwerkelijke experimenten met sequentiële en parallelle implementaties noodzakelijk om meer ervaring te verkrijgen en om de kernproblemen te identificeren.