Samenvatting

Het ontwikkelen van correcte en betrouwbare programmatuur is een complexe aangelegenheid. In een samenleving die meer en meer afhankelijk wordt van een toenemende hoeveelheid programmatuur is het noodzakelijk dat programma's correct zijn. Eén van de manieren om die complexiteit te verminderen is het gebruik van declaratieve programmeertalen. In deze familie van programmeertalen construeert men programma's door te beschrijven *wat* er moet gebeuren in plaats van *hoe* het moet gebeuren. Eén klasse van declaratieve programmeertalen wordt gevormd door de functionele programmeertalen. Deze worden vaak geroemd omdat men bondige programma's kan schrijven op een eenvoudige en begrijpelijke wijze, het goed mogelijk is de correctheid van programma's te bewijzen en ze geschikt zijn voor programma transformaties. Eén van de redenen waarom functionele talen declaratief zijn is dat hun berekeningsmodel een abstractie vormt van het concept van berekening in computers. Programmeurs in een functionele taal hoeven niet na te denken over geheugenbeheer en de volgorde waarin programma instructies uitgevoerd worden.

In de samenleving wordt in toenemende mate gebruik gemaakt van computers. De reden dat computers zo nuttig zijn is het feit dat ze niet alleen kunnen rekenen, maar ook *interactief* zijn, in de ware zin van het woord[1]. Computers kunnen interactie plegen met gebruikers, andere computers en meer conventionele apparaten zoals robots, vliegtuigen, treinen en auto's. Het zijn de programma's die vastleggen hoe die interactie plaatsvindt. Het ontwikkelen van correcte en betrouwbare *interactieve* programmatuur is echter complex. Eén manier om die complexiteit terug te dringen is het vinden van een declaratieve methode. Dit is het hoofdthema van dit proefschrift.

In dit proefschrift laten we zien dat functionele berekening en interactie op een natuurlijke wijze samen te brengen zijn. De programmeertaal die gebruikt is als onderzoeksgereedschap is de functionele programmeertaal Clean. De aanpak in dit proefschrift bestaat uit twee stappen. De eerste en fundamentele stap structureert interactie op een zodanige wijze dat het functionele berekening wordt. Met andere woorden: *interactie is berekening* en in rekenen blinken functionele talen uit. Deze stap is mogelijk door gebruik te maken van het speciale uniciteits typeringsysteem van Clean. Dit typeringsysteem is gebaseerd op de term graaf herschrijf semantiek van Clean. De tweede stap in de aanpak bepaalt per type interactie hoe deze in het functionele schema ingepast kan worden.

De interactietypes die we beschouwd hebben zijn bestandsbeheer en grafische gebruikers interfaces. Het blijkt dat bestandsbeheer qua programmeerstijl een speciaal geval is van berekening. Grafische gebruikers interfaces onderscheiden zich van bestandsbeheer omdat ze dynamisch zijn. Het programmeren van grafische gebruikers interfaces vereist daarom een andere aanpak. In deze aanpak is gebruik gemaakt van de flexibiliteit van algebraïsche data types in functionele talen om een specificatietaal te ontwerpen waarin grafische gebruikers interface elementen beschreven worden. Dit zijn elementen als 'windows', 'menus' en 'controls', maar ook 'timers' en 'receivers'. Aan een interface beschrijving worden functies toegevoegd ('abstract event handlers'), die de reactie van het programma beschrijven als er een bepaalde situatie optreedt ('abstract event'). Elk van deze functies kan op de toestand van het programma toegepast worden en levert een nieuwe toestand op. Omdat Clean een sterk getypeerde taal is kan het typeringsysteem bepalen of alle specificaties van een correct type zijn. Het systeem zal, gegeven de specificatie van een interactief programma, alle machine-afhankelijke activiteiten afhandelen en

[1] **interactief** … elkaar wederzijds beïnvloedend, op elkaar inwerkend. (Van Dale, 1984)

de correcte functies van het programma aanroepen. De semantiek van een interactief programma is die van een toestandsovergangssysteem. Deze is rechttoe-rechtaan te beschrijven met behulp van een eenvoudige verzameling functies.

Modulair programmeren is een krachtig taal-onafhankelijk concept om de complexiteit van programmatuur ontwikkeling terug te brengen. Het stelt programmeurs in staat een programmeertaak te verdelen in kleinere, onafhankelijke, programmeertaken. In dit proefschrift laten we zien hoe interactieve programma's op een modulaire wijze geconstrueerd worden met behulp van interactieve processen. Interactieve processen zijn een algemener concept dan interactieve programma's. Net als interactieve programma's zijn interactieve processen in wezen toestandsovergangssystemen en worden ze op dezelfde wijze beschreven. Interactieve programma's bestaan uit een willekeurig aantal interactieve processen en elk interactief proces kan bestanden en grafische gebruikers interfaces manipuleren. We laten zien dat interactieve processen zowel op een sequentiële als parallelle wijze geëvalueerd kunnen worden.

In dit proefschrift introduceren we twee vormen van communicatie tussen interactieve processen: 'data sharing' en 'message passing'. Beide zijn qua model eenvoudig te integreren in het systeem. Communicatie door middel van 'data sharing' maakt gebruikt van globaal bekende datastructuren. Er kan maar één proces tegelijk toegang hebben tot een publieke datastructuur. Een flexibele vorm van 'many-to-one' communicatie is 'message passing'. Elk interactief proces kan van een willekeurig ander interactief proces berichten ontvangen mits dat interactieve proces over zijn identificatie beschikt.

Toestandsovergangssystemen blijken een nuttig en algemeen toepasbaar concept te zijn. We hebben ze gebruikt als basis voor interactieve processen en 'data sharing'. Vanuit een iets ander standpunt kunnen toestandsovergangssystemen ook beschouwd worden als objecten, in de zin van object-georiënteerd programmeren. Objecten zijn op een uniforme wijze samen te stellen uit andere objecten en het is mogelijk een functionele betekenis te geven aan die structuren. Als toepassing breiden we grafische gebruikers interface elementen uit met lokale toestand en verkrijgen we op een nieuwe manier interactieve processen en 'data sharing'.

Alle systemen in dit proefschrift zijn polymorf en type-correct. Polymorfie staat een vrije keuze toe van de types van toestanden, berichten en objecten. Het sterke typeringsysteem van Clean is toegepast om type-correcte systemen te krijgen. Tijdens het onderzoek is gebleken dat het conventionele polymorfe Milner/Mycroft typeringsysteem, waarop de typeringsystemen van de meeste moderne functionele talen zijn gebaseerd, niet krachtig genoeg is deze complexe systemen te typeren. We laten zien dat hoewel abstracte data types met meerdere interne representaties in een polymorf Milner/Mycroft typeringsysteem gerealiseerd kunnen worden, het eleganter en algemener is het systeem uit te breiden met existentiële types. Het 'message passing' systeem is type-veilig, maar kan niet statisch getypeerd worden in het typeringsysteem. Voor de constructie van goed getypeerde interpretatie functies van object structuren dient het typeringsysteem uitgebreid te worden met type klassen en constructor klassen.

In dit proefschrift hebben we een functioneel, declaratief systeem geconstrueerd waarin men interactieve functionele programma's kan ontwikkelen op een hoog niveau van abstractie. We denken dat dit een stap is naar het terugbrengen van de complexiteit van de ontwikkeling van correcte en betrouwbare programmatuur.