Samenvatting

HOOFDSTUK 2: CONCRETE SYNTAX VOOR OBJECTEN

Interfaces voor applicatieprogrammeurs (APIs) bieden toegang tot domeinkennis die geëncapsuleerd is in klassebibliotheken. Ze bieden echter vaak geen optimale notatie om programma's samen te stellen voor dit domein. Omdat object-georiënteerde talen zijn ontworpen voor uitbreidbaarheid en hergebruik, zijn de taalconstructies vaak voldoende voor het uitdrukken van domeinabstracties op een semantisch niveau. Op een syntactisch niveau bieden deze talen desalniettemin niet de juiste abstractie. In dit hoofdstuk beschrijven we MetaBorg, een methode voor het bieden van *concrete syntax* (*syntaxis*) voor domeinabstracties aan applicatieprogrammeurs. De methode bestaat uit het *embedden* van een domein-specifieke taal in een algemene programmeertaal en het *assimileren* van de geëmbedde code in de omringende hostcode. In plaats van het uitbreiden van de implementatie van de hosttaal implementeert de assimilatiefase de domeinabstracties in termen van bestaande APIs, waardoor de hosttaal verder ongemoeid wordt gelaten. MetaBorg kan zo gezien worden als een methode om APIs naar het niveau van de taal te promoveren. De methode wordt ondersteund door bestaande en bewezen technologie, namelijk het grammaticaformalisme SDF en de programmatransformatietaal en tools van Stratego/XT. We illustreren de methode met toepassingen in drie verschillende domeinen: code generatie, XML generatie, en het implementeren van gebruikersinterfaces.

HOOFDSTUK 3: TYPE-GEBASEERDE DISAMBIGUATIE

Bij metaprogrammeren met concrete objectsyntax worden programma's van de objecttaal samengesteld uit fragmenten die geschreven zijn in de concrete syntax van de objecttaal. Het gebruik van kleine programmafragmenten in zulke quotes en het gebruik van metaexpressies in deze fragmenten (antiquotes) leidt vaak tot syntactische ambiguïteiten. Dit probleem wordt meestal opgelost door de programmeur de fragmenten expliciet te laten disambigueren, wat zorgt voor aanzienlijke syntactische overhead. Enkele systemen vermijden dit door typeinformatie te gebruiken tijdens het parseren (*ontleden*). Doordat dit lastig te realiseren is met traditionele parseertechnieken zijn deze systemen specifiek voor een combinatie van één meta- en één objecttaal. Ook zijn de implementaties van deze systemen niet herbruikbaar.

In dit hoofdstuk generaliseren we deze aanpak en presenteren een *taalonafhankelijke* methode voor het introduceren van concrete objectsyntax zonder expliciete disambiguatie te vereisen. De methode maakt gebruik van het *generalized LR* algoritme voor het parseren van metaprogramma's met concrete objectsyntax. Dit resulteert in een woud van alle mogelijke manieren waarop

het programma te parseren is. Dit woud wordt gereduceerd tot een enkele boom door een disambiguerende typechecker voor de metataal. Ter validatie hebben we diverse objecttalen opgenomen in Java, waaronder AspectJ en Java zelf.

HOOFDSTUK 4: PREVENTIE VAN INJECTIEAANVALLEN

Software geschreven in een bepaalde programmeertaal moet vaak zinnen construeren in een andere programmeertaal, bijvoorbeeld SQL-zoekacties, XML-documenten, of Shellcommando's. Dit wordt vrijwel altijd gedaan met onhygiënische stringmanipulatie, waarbij constante strings en gebruikersinvoer samengevoegd worden. Een gebruiker kan in deze situatie een kwaadaardige string invoeren die ervoor zorgt dat de uiteindelijk gegenereerde zin geïnterpreteerd wordt op een door de programmeur onbedoelde wijze. Dit is een injectieaanval.

Wij presenteren in dit hoofdstuk een meer natuurlijke stijl van programmeren die code oplevert die ongevoelig is voor injectieaanvallen. Onze aanpak garandeert door de manier waarop zinnen geconstrueerd worden dat de uiteindelijk gegenereerde zin altijd overeenkomt met de zinnen die de programmeur bedoelde te genereren. Voor onze aanpak embedden we een gasttaal (bijvoorbeeld SQL) in een hosttaal (bijvoorbeeld Java) en genereren uit de geëmbedde fragmenten van de gasttaal automatisch code die deze omzet naar code in de hosttaal die deze zinnen veilig construeert. Waar nodig worden automatisch functies aangeroepen die karakters met een voor de hosttaal speciale betekenis omzetten. Onze methode is generiek doordat het relatief eenvoudig toegepast kan worden op willekeurige combinaties van host- en gasttalen.

HOOFDSTUK 5: GRAMMATICA VAN ASPECTJ

Aspect-georiënteerd programmeren (AOP) geniet belangstelling vanuit zowel de academische wereld als de industrie. Dit wordt geïllustreerd door de alsmaar groeiende populariteit van AspectJ, de standaard AOP uitbreiding van Java. Vanuit het perspectief van compilerbouw is AspectJ interessant omdat het een typisch voorbeeld is van een taalagglomeraat, een taal die bestaat uit een aantal subtalen met ieder een verschillende syntax. Naast Java bevat AspectJ namelijk ook talen voor het definiëren van patronen, *pointcuts*, en *advice*. Dergelijke samenstelling van talen is een uitdaging voor conventionele parseertechnieken. Het samenvoegen van twee of meer talen met een verschillende lexicale syntax zorgt voor nogal wat complexiteit in de lexicale toestanden waarmee scanners vaak werken. Ook is er nog steeds actief onderzoek naar nieuwe taaleigenschappen voor AOP. Dergelijke voorstellen zijn vaak uitbreidingen van AspectJ, waardoor er behoefte is aan een uitbreidbare grammatica van AspectJ.

In dit hoofdstuk laten we zien hoe parseren zonder een scanner (*scannerless parsing*) elegant deze problemen met het gebruik van conventionele par-

seertechnieken voor AspectJ oplost. We presenteren het ontwerp van een modulaire, uitbreidbare, en formele definitie van zowel de lexicale als de context-vrije syntax van AspectJ in het grammaticaformalisme SDF. Parsers die gegenereerd worden uit SDF grammatica's maken gebruik van het scannerless generalized LR algoritme. Verder introduceren we *grammar mixins*, een nieuwe toepassing van het SDF modulesysteem. Grammar mixins maken het mogelijk om de verschillende keyword policies (*beleid* voor *gereserveerde woorden*) van AspectJ-compilers declaratief te beschrijven. We illustreren de modulaire uitbreidbaarheid van onze definitie met syntactische uitbreidingen die voorgesteld zijn in recent onderzoek naar aspecttalen. Tot slot laten benchmarks zien dat de snelheid van de scannerless generalized LR parser acceptabel is voor deze grammatica.

HOOFDSTUK 6: SAMENSTELLEN VAN PARSEERTABELLEN

Modulesystemen, gescheiden compilatie, het afleveren van binaire componenten, en dynamisch linken zijn algemeen geaccepteerd in programmeertalen en systemen. De syntax van een taal is daarentegen meestal niet modulair gedefiniëerd, kan niet gescheiden gecompileerd worden, kan niet eenvoudig gecombineerd worden met de syntax van andere talen, en kan niet afgeleverd worden als een component die later gecombineerd wordt met andere componenten. Grammaticaformalismen die wel modules ondersteunen compileren uiteindelijk toch alle modules gezamenlijk.

De huidige uitbreidbare compilers zijn ontworpen om uitgebreid te worden op het niveau van hun broncode. De gebruiker moet hierdoor de *compiler* compileren voor elke specifieke configuratie van uitbreidingen. Voor elke combinatie moet hiervoor ook een samengestelde parser gegenereerd worden. Het genereren van een parser is echter duur, wat vooral een probleem is wanneer de combinaties niet vastliggen en de gebruiker zelf taaluitbreidingen kan kiezen.

In dit hoofdstuk introduceren we een algoritme voor het samenstellen van parseertabellen. Dit algoritme ondersteunt het gescheiden compileren van grammatica's naar *parseertabelcomponenten*. Parseertabelcomponenten kunnen efficiënt samengesteld (gelinkt) worden juist voordat de tabellen gebruikt worden voor het parseren. De complexiteit van het samenstellen van parseertabellen is in het slechtste geval exponentieel (zoals ook de complexiteit van parseertabelgeneratie), maar voor realistische scenario's van het combineren van talen is ons algoritme substantieel sneller dan het berekenen van een parseertabel uit de gecombineerde grammatica's.

HOOFDSTUK 7: AFLEIDEN VAN GROEPERINGSREGELS

Er zijn veel verschillende parsergeneratoren in gebruik. De grammaticaformalismen van deze parsergeneratoren bieden verschillende methoden voor het definiëren van groeperingsregels (*precedence rules*). Sommige generatoren (zoals bijvoorbeeld YACC) ondersteunen declaratie van groeperingsregels, ter-

wijl andere generatoren vereisen dat groeperingsregels afgedwongen worden door de productieregels van de grammatica. Zelfs wanneer een grammaticaformalisme declaraties voor groeperingsregels ondersteunt, zou een specifieke grammatica deze taalconstructie kunnen negeren en de groeperingsregels toch in de productieregels verwerken.

Het resultaat is een verzameling varianten van grammatica's die allemaal dezelfde taal definiëren. Voor de taal C gebruikt de GNU Compiler de parsergenerator YACC met groeperingsregeldeclaraties, het C-Transformers project gebruikt SDF zonder prioriteiten, terwijl de C-grammatica van de SDF bibliotheek deze prioriteiten wel gebruikt. Voor PHP gebruikt Zend YACC met groeperingsregeldeclaraties, terwijl het PHP-front pakket SDF met prioriteiten associativiteitdeclaraties gebruikt. Deze variatie in grammatica's roept de vraag op of de groeperingsregels van de ene grammatica wel overeenkomen met die van de andere. In het algemeen is dit vaak niet direct duidelijk, omdat sommige talen zeer complexe groeperingsregels hebben. Voor sommige parsergeneratoren is de semantiek van groeperingsregeldeclaraties ook louter operationeel gedefiniëerd, waardoor het lastig is om te redeneren over het effect van deze declaraties op de gedefiniëerde taal.

In dit hoofdstuk presenteren we een methode en een tool voor het vergelijken van groeperingsregels van verschillende grammatica's en parsergeneratoren. Alhoewel de vraag of twee grammatica's dezelfde taal definiëren onbeslisbaar is, ondersteunt onze methode het vergelijken en afleiden van groeperingsregels. Dit is in het bijzonder nuttig voor het betrouwbaar migreren van een grammatica naar een ander grammaticaformalisme. We evalueren onze methode door deze toe te passen op enkele niet-triviale programmeertalen, namelijk PHP en C.