Projectvoorstel Machines en Berekenbaarheid

Parser generator: ***Parsodus***

Thomas Avé, Robin Jadoul, Kobe Wullaert

In fase `I` werd door deze groep reeds de omzetting van een CFG naar CNF[1], van een CFG naar een PDA en het CYK algoritme[2], PDA naar CFG, een SLR(1) en een LL(1) parser geïmplementeerd[3].

Voor fase `II` stellen we onszelf tot doel een programmeertaal agnostische generator voor parsers te maken. In tegenstelling tot de bekende en frequent gebruikte *yacc* of *bison* is het ons doel hierbij in de feitelijke specificatie voor de analysator geen specifieke programmeertaal te laten voorkomen (bij yacc/bison moet men specifieke code schrijven in de gewenste programmeertaal). Dit heeft tot voordeel dat eenzelfde input bestand eenvoudig hergebruikt kan worden in meerdere projecten, zonder verdere compatibiliteitsproblemen. Zo kan bijvoorbeeld een standaardcomité van een programmeertaal een voorbeeldspecificatie in dit inputformaat vrijgeven, en kunnen verschillende toepassingen daarvan gebruik maken zonder op eender welke manier gerestricteerd te zijn, of van elkaar afhankelijk te zijn.

Intern zal de generator werken door een context-vrije grammatica (die aanvaard wordt door het gewenste type parser) die de specificatie van een taal beschrijft om te zetten naar een `LR of LL parse table`, waarop een generisch LR/LL parse algoritme kan werken (het exacte type parse table is mogelijk variabel en specifieerbaar door de gebruiker), dat extra annotaties of code kan hebben om het aanmaken van een *abstract syntax tree* of het uitvoeren van acties tijdens het parsen kan bevorderen. Een mogelijke uitbreiding voor het oplossen van ambiguïteiten in de grammatica zou het ondersteunen van voorrangsregels (zoals wiskundig gezien: eerst vermenigvuldiging, dan optelling), en associativiteitsspecificaties zijn.

Voor een gegeven invoerbestand zal dan de mogelijkheid bestaan om dit om te zetten naar een sourcecode bestand naar keuze[4]. Dit bestand kan dan mee gecompileerd worden (of geïnterpreteerd, afhankelijk van de taal) en op die manier een interface beschikbaar stellen waarmee de uiteindelijke toepassing een inputstroom (van tokens, dankzij de geplande integratie met Lexesis) zal kunnen parsen en verder verwerken. Bovendien zal de generator in staat zijn (dankzij de programmeertaal-onafhankelijkheid) een idiomatische interface vrij te geven, per programmeertaal.

Om verder ook het nut en de gebruiksvriendelijkheid van onze generator aan te tonen, zullen we mogelijk ook trachten een eenvoudige programmeertaal (bv. *brainfuck*) te parsen en te interpreteren of compileren en de regex-parser van Lexesis te vervangen door een nieuwe gegenereerde parser.

---

[1] Kobe Wullaert
[2] Thomas Avé
[3] Robin Jadoul
[4] Indien de gewenste taal geïmplementeerd is, initieel wordt enkel `C++` ondersteuning gepland