Project T&A – Scheme interpreter

Koen Pauwels, Ajaya Adhikari, Rafael De Smet

Universiteit Antwerpen

België

koen.pauwels@student.uantwerpen.be

ajaya.adhikari@student.uantwerpen.be

rafael.desmet@student.uantwerpen.be

Als project hebben we gekozen voor de implementatie van een Scheme interpreter. Scheme is een minimalistisch dialect van Lisp, met een erg gestructureerde en uniforme structuur. Deze eigenschappen maken het relatief makkelijk om de taal te parsen. Een eerste stap van het parsen is de lexicale analyse: het omzetten van een string van karakters naar een serie van tokens die door de parser eenvoudig herkend kunnen worden. De toepassing van Talen en Automaten in dit project ligt in deze lexicale analyse.

De automaat die de analyse uitvoert noemen we een 'lexer'. Deze is geïmplementeerd aan de hand van een DFA die alle sleutelwoorden herkent die ingebakken zijn in de interpreter, maar ook de namen van alle globaal gedefinieerde waarden en functies en de namen van de waarden in de relevante scope. Er zal een nauwe samenwerking bestaan tussen de lexer en de rest van de parser: de parser houdt op elk moment bij welke strings aanvaard worden (maw welke tokens relevant zijn binnen de scope) en past de lexer aan om enkel die strings te aanvaarden. Als een functie een parameter aanvaardt, is die parameter een geldige string binnen de scope van de functie, maar niet meer erbuiten. De lexer moet dus buiten de scope van de functie die string terug weigeren. Verder zijn er situaties waar nieuwe tokens kunnen worden gecreëerd, zoals (define …), dus moet de lexer alle geldige namen voor variabelen aanvaarden, iets wat best uitgedrukt wordt aan de hand van een regex.

De interpreter zal dus nood hebben aan een algoritme om het product van automaten te nemen, en een algoritme om regexes om te zetten naar DFA's. Verder is een minimalisatiealgoritme ook wenselijk, om de lexicale analyse efficiënt te laten verlopen.

Voor fase 1 verdelen we het werk als volgt: Ajaya implementeert product van automaten, Rafael zorgt voor de omzetting van regexes naar DFA's en Koen schrijft het table-filling algoritme.

Tijdsindeling Fase 1

Productautomaat	10 uur	Ajaya Adhikari
Implementatie DFA .	15 uur	Allen
Implementatie ε-NFA.	15 uur	Allen
Omzetting ε-NFA naar DFA	20 uur	Rafael De Smet
Omzetting regex naar DFA	20 uur	Ajaya Adhikari
DFA minimalisatie door table-filling algoritme	20 uur	Koen Pauwels

Tijdsindeling Fase 2

Parser – lexer	10 uur per persoon	Allen
Parser – de rest	30 uur per persoon	Allen
Eigenlijke interpretatie, echte uitvoering	10 uur per persoon	Allen