Compilerbau: Zusammenfassung der Ergebnisse

Plümicke

Thomas Stüber

Donnerstag, 14. Februar 2015

Allgemein

Mein Teil des Projekts war die Implementierung von Scanner und Parser. Dabei wurde mit der Programmiersprache Haskell und den Scanner- bzw. Parsergeneratoren “alex” und “happy” gearbeitet. Den groben zeitlichen Verlauf können sie dem Verlauf des Github-Repository oder der Präsentation entnehmen.

Scanner

Der Scanner wurde in kurzer Zeit implementiert. Alle Schlüsselwörter und Operatoren stellen selbst schon die zugehörigen regulären Ausdrücke dar. Es mussten etwas aufwendigere reguläre Ausdrücke für Literale und Bezeichner geschrieben werden, so dass Binär-, Oktal- und Hexliterale, sowie Unterstriche in Literalen, unterstützt werden. Strings stellten ein größeren Problem dar, da keine einfache Regel aufgestellt werden konnte die sich von escapeten Anführungszeichen nicht “verwirren” ließ. Außerdem mussten Unicode-Characters sowie Escape-Sequenzen aufgelöst werden, was durch eine Hilfsfunktion durchgeführt wurde. Der Scanner erkennt zwei Sorten von lexikalischen Fehlern: Identifier die mit Zahlen beginnen und Zeichen die in Identifiern ganz verboten sind.

Parser

Der Parser war die eigentliche Aufgabe an meinem Teil des Projekts. Als Basis wurde hier ihre Grammatik verwendet, da diese schon einige Probleme behob die ich mit eigenen Grammatiken hatte, etwa dem Dangling-Else-Problem. Nun musste für unzählige Konstrukte der genaue Syntax ermittelt werden welche dann in Form einer möglichst konfliktfreien Grammatik dargestellt werden mussten. Dies für sich war schon schwer, was nicht leichter wurde durch die Typen der eizelnen Nichtterminale die man kaum im Überblick behalten konnte, weshalb man oft mit schwer zu findenden Typfehlern rechnen musste. Auch Cast-Expressions führten zu Konflikten, was durch einen Trick mit einer Hilfsfunktion gelöst wurde, so dass ein Nichtterminal wiederverwendet werden konnte das eigentlich für andere Dinge gedacht war. Deklerationen von mehreren Variablen in einem Statement wurden entzuckert zu mehreren einzelnen Deklerationen. Operatoren die Literale verarbeiten werden nicht in abstrakten Syntax übersetzt, stattdessen werden die resultierenden Ergebnisse direkt berechnet, so dass Constand-Folding implementiert wurde. Ob die linke Seite einer Zuweisung überhaupt als solche geeignet ist wird ebenso erkannt, wie die falsche Verwendung der vorhandenen Modifier. Wenn unerreichbarer Code vorhanden ist führt dies ebenfalls zu einer Fehlerausgabe.

Abstrakter Syntax

Der abstrakte Syntax ist eine einfache Erweiterung des von ihrer Vorlage, allerdings wurde diese deutlich Erweitert. Auch wurden Fehler ihres abstrakten Syntax behoben, etwa die falsche Darstellung der linken Seite von Zuweisungen. Alle Parameter von Datenkonstruktoren haben Namen erhalten, so dass der Haskell-Compiler automatische Zugriffsfunktionen generieren kann. Nachdem Datenkonstruktoren für alle untersützten Konstrukte implementiert waren wurde noch eine Pretty-Print-Funktion implementiert, die hauptsächlich zum einfachen Testen des Parsers gedacht war während des Entwicklungsprozesses, und den abstrakten Syntax als Baum darstellt. Diese Funktion war vermutlich für die anderen Teilnehmer auch relevant beim Untersuchen wie der abstrakte Syntax genau funktioniert.

Ausblick

Der Scanner enthält schon die Schlüsselwörter und Operatoren die Java momentan anbietet, hier muss nur eventuell für Fließkommaliterale nachgebessert werden. Weitere Features im Parser lassen sich großteils leicht implementieren wenn man erst mal eingearbeitet ist. Einzige Ausnahme wären hier wohl Generics und Wildcards, welche größere Umbauten verlangen, da sich dadurch die interne Repräsentation von Datentypen ändern müsste. Durch die Implementierung von “happy” dürfte Aufsetzen bei Fehlern kaum implementierbar

sein, allerdings können die anderen Compilerteile vereinfacht werden wenn der Parser mehr Konstrukte entzuckert, etwa die vielen zusammengesetzten Zuweisungsoperatoren. Im großen ganzen hat mir das Projekt Spaß gemacht und ich würde an einer Folgeveranstaltung teilnehmen.