Präsentation der Projektergebnisse

Thomas Stüber

13. Februar 2015

Der Scanner

Kurze Wiederholung: der Scanner zerlegt die Eingabe in atomare Bausteine, sogenannte Lexeme oder Tokens. Die Regeln durch die diese erzeugt werden werden durch reguläre Ausdrücke beschrieben.

Der Parser

Kurze Wiederholung: der Parser verwendet die vom Scanner erzeugten Lexeme und überführt diese gemäß syntaktischen Regeln in einen abstrakten Syntax. Die entsprechenden Regeln werden durch eine kontextfreie Grammatik (mit Einschränkungen), notiert in Backus-Naur-Form, beschrieben.

Aufgabe

Mit dem Scanner-Generator “alex” und dem Parsergenerator “happy” sollten beide eben genannten Teile eines Compilers implementiert werden für die Programmiersprache Java (mit einigen Einschränkungen). Dabei wurde hauptsächlich das beschrieben was Scanner und Parser tun sollen und am Kern der Funktionalität musste wenig selbst programmiert werden. Außerdem musste eine Zusammenfassung und eine Präsentation der Ergebnisse erstellt werden.

Zeitverlauf

7.12.	Erster Commit
8.12.	Pretty Printer für abstrakten Syntax
8.-13.	Weiteres arbeiten am Parser
13.12.	Parser neu begonnen um einige Shift/Reduce Konflikte von Anfang an zu vermeiden, abstrakter Syntax erweitert
13.-26.	Kleine Tests
26.12.	Erste Version auf Basis von Prof. Plümickes Grammatik
26.-30.	Hauptteil der Arbeit an abstraktem Syntax und Parser
12.2.	Letzte Arbeiten, verschönern und aufräumen des Codes, kleinere Features, Präsentation, Dokumentation

Aufgaben

Der Scanner

Der Parser

Allgemein

1. Der Scanner wurde ohne die Vorlage entwickelt, da aus bloßen Tests der Funktionen von “alex” schnell ein vollständiger Scanner wurde.
2. Die regulären Ausdrücke für Schlüsselwörter und Operatoren waren jeweils einfach das Schlüsselwort oder der Operator selbst.
3. Es wurde der Wrapper mit Positionsangaben verwendet.
4. Der Token-Datentyp besitzt für jedes Lexem einen Datenkonstruktor. Dieser erhält bei jedem Lexem die Positionsangabe und bei einigen Lexemen noch spezielle Werte, z.B. der String der als Stringliteral erkannt wurde.

Weitere Features

1. Umrechnen von Hex-, Oktal- und Binärliteralen.
2. Identifier die mit Zahlen beginnen führen zu Fehlern.
3. In String- und Charliteralen werden Escape-Sequenzen aufgelöst, insbesondere Unicode Zeichen.
4. Zeichen die für Bezeichner überhaupt nicht erlaubt sind führen zu einer entsprechenden Fehlermeldung.

Probleme

1. In Integerliteralen sind Unterstriche erlaubt, allerdings nicht an Anfang und Ende. Außerdem führt eine führende 0 zu einer Oktalzahl, mehr als eine führende 0 hingegen zu einem lexikalischen Fehler.
2. Das Auflösen von Escape-Sequenzen gestaltete sich schwierig.
3. In String darf \" vorkommen, was aber nicht das Ende des Strings markiert. Einfach einen regulären Ausdruck wie ".*" zu verwenden wird also fehlschlagen.

Aufgaben

Der Scanner

Der Parser

Allgemein

1. Der Parser wurde mehrfach zu großen Teilen entwickelt, der endgültige Parser basiert auf der Vorlage von Prof. Plümicke.
2. Es existiert eine Grammatik von Oracle an der sich die Entwickler der offiziellen Implementierung orientieren, diese ist aber weder LALR(1), noch in Backus-Naur-Form verfügbar. An ihr konnte ich aber gut prüfen ob ich Sprachkonstrukte vergessen habe die ich selbst einfach nicht kenne.
3. Es wurde ein Parser ohne Monaden gewählt, da die Funktionalität bei der die Monaden mehr als syntaktischer Overhead wären über das Projekt hinaus gehen würden.

Weitere Features

1. Constand-Folding, Operatoren auf Literalen werden bereits vom Parser berechnet.
2. Einfache Fehlerverarbeitung, immerhin wird zwischen 7 verschiedenen Fehlern unterschieden.
   - 2.1 Unerreichbarer Code wird erkannt.
   - 2.2 Ungültige Modifier für das jeweilige Konstrukt werden erkannt
   - 2.3 Unerwartetes Symbol oder unerwartetes Ende der Eingabe
3. Operatorpräzedenz und Assoziativität durch hierarchischen Aufbau der Grammatik umgesetzt anstatt durch Features von “happy”
4. Keine Shift/Reduce Konflikte oder Reduce/Reduce Konflikte, insbesondere bei Casts und Dangling-Else.

Probleme

1. Diverse Typfehler, die nur sehr schwer zu finden waren.
2. Viele Sprachkonstrukte lassen sich einfach in BNF darstellen aber führen dann zu Konflikten.
3. Viele unbekanntere Sprachkonstrukte, die ich selbst erst durch den Parser kennen gelernt habe, deren genauer Syntax nicht trivial ist.
4. For-Schleifen und Switch-Verzweigungen erfordern viele Regeln für wenig genutze Sonderfälle.
5. Unterscheidung von Statements mit folgendem Teilstatement und “dem Rest”, Statements in Schleifen und Statements außerhalb von Schleifen.