JLex & CUP

Martin Knobloch

Allgemeines

JLex und CUP sind als Java-Pendants zu Lex/Flex und bison/yacc konzipiert. JLex erzeugt als in Java geschriebener Scanner-Generator auf seinen Scanner-Klassen basierende Scanner, CUP erzeugt auf seinen Klassen basierende Parser.
JLex und CUP werden an der Universität Princeton betreut. Unter www.cs.princeton.edu/~appel/modern/java/JLex oder .../java/cup sind die Homepages der Projekte zu finden.

Installation

Beide Programme liegen im Quellcode oder kompiliert vor. Vor dem Benutzen müssen die Quellen ggf. kompiliert werden.

JLex

JLex wurde von Elliot Berk an der Princeton-Universität entwickelt und wird als freie Software angeboten. Die aktuelle Version ist 1.2.6 vom 3.Feb. 2003.
Da das Tool als Java-Pendant zu Lex/Flex geschrieben wurde, bestehen auch starke Ähnlichkeiten im Funktionsumfang. JLex kann alle Zeichenfolgen scannen, die durch einen regulären Ausdruck dargestellt werden können. Der Aufruf erfolgt nach dem Schema `jlex <Datei>`. Die Eingabedatei umfasst drei Sektionen: User-Code mit Java-Code, der direkt an den Anfang der von JLex erstellten Datei kopiert wird. Anschließend erwartet JLex Direktiven, die die Scanner-Erstellung beeinflussen, sowie die Definition von Makros für die regulären Ausdrücke der nachfolgenden Sektion. Hier werden die zu erkennenden Zeichenfolgen definiert und ein Aktionscode eingefügt, der bei erfolgreicher Suche ausgeführt wird.
Der Scanner kann durch die Verwendung des User-Codes (Klasse mit main-Methode) oder durch die Verwendung der Direktiven (main-Methode in die Scanner-Klasse einfügen) als Stand-alone-Scanner benutzt werden. Im Allgemeinen wird JLex jedoch in Verbindung mit CUP benutzt.
Der Scanner arbeitet mit der longest match - Regel, d.h. er betrachtet immer die längstmögliche Zeichenfolge. Als weiteres Feature wird die Unicode - Tauglichkeit angegeben, die sich über einen geeigneten Reader im Constructor-Aufruf realisieren lässt.

CUP

Der „Constructor for Useful Parsers" wurde von Scott E. Hudson entwickelt und wird an der Princeton-Universität weiterentwickelt. Die aktuellste, zum Vortragszeitpunkt bekannte Version 0.10k stammt vom 24.Juli 1999.
CUP generiert LALR(1) Parser und ist, wie auch JLex, seinem Vorgänger sehr ähnlich. Als Eingabe wird eine Datei mit User-Code (wie JLex), Tokenlisten (Terminal/Nonterminal), Direktiven, Assoziativitäten und Präferenzen erwartet. Darunter hat sich dann die Grammatik zu befinden, die allerdings nur in BNF vorliegen darf.
In den Tokenlisten werden, getrennt nach Terminalen und Nichterminalen, alle Symbole aufgelistet und ggf. mit einem Typ versehen. Die Symbole werden so entweder als native Datentypen oder als Objektinstanzen betrachtet.
Das Startsymbol ist, wenn durch die Direktiven nicht anders festgelegt, das Symbol der am weitesten oben stehenden Regel. In den Regeln befinden sich semantische

Aktionen, die entweder beim Reduzieren oder auch während des Aufbaus der Regel ausgeführt werden können. Letzteres zieht allerdings undefinierte Auswirkungen nach sich. Die Erstellung von semantischen Aktionen wird durch die Verwendung von Labeln vereinfacht, durch die die Symbole referenziert werden.
Die Nummern der Terminale werden standardmäßig in einer Klasse als Konstanten ausgegeben, das definieren eigener Werte ist nicht möglich.
Konflikte in der Grammatik werden bei der Parser-Generierung gemeldet, Shift-Reduce-Konflikte können durch die Angabe einer erwarteten Anzahl von S-R-Konflikten in der Komandozeile (`-expect n`) gelöst werden. In diesem Fall shiftet der Parser standardmäßig.
Die Wiederaufnahme des Parsevorgangs nach einem Fehler kann durch das Einfügen des vordefinierten Symbols `error` ermöglicht werden.
Die für den Parsevorgang benötigten Symbole können durch einen beliebigen Scanner zur Verfügung gestellt werden.

JLex & CUP im Team
Wie auch ihre Vorgänger sind die Tools aufeinander abgestimmt. CUP liefert die Nummern der Terminalsymbole, die JLex beim Scanvorgang an CUP weitergibt. JLex benutzt dazu Objekte der Klasse `java_cup.runtime.Symbol`. Zudem besitzen beide Tools Direktiven, die den erzeugten Scanner/Parser auf das Zusammenspiel optimal abstimmen sollen.

Vergleich: JLex & CUP vs. Flex & bison
Die Tools unterscheiden sich in erster Linie in der verwendeten Sprache, da es sich schließlich um Pendants unterschiedlicher Sprachen handeln soll. Die Java-Varianten sind hinsichtlich der Eingabedateien strukturierter, d.h. Namen von Makros werden durch ein = von ihren Regeln getrennt, semantische Aktionen stehen prinzipiell in geschweiften Klammern. Zudem sind die Eingabedateien in einer anderen Reihenfolge strukturiert, was hinsichtlich der verwendeten Sprachen nicht verwunderlich ist.
JLex macht Einschränkungen hinsichtlich einiger Regel-Definitionen: Eine genaue Wiederholungsanzahl kann nicht festgelegt werden, R/S (R, wenn S folgt) fehlt auch.
In CUP fallen vor allem die Label auf, die die Adressierung des Stacks mit $$ oder $1...$n ersetzen.
Es bestehen alles in allem jedoch nur wenige Unterschiede.

Fazit
JLex ist ein relativ leicht zu bedienender Scanner-Generator, mit dem sich schnell Scanner erzeugen lassen.
CUP hat wie auch bison den Nachteil, keine EBNF verarbeiten zu können. Somit ist die Generierung eines Parsers teilweise recht aufwendig, da Listen erstellt und Sonderregeln für optionale Symbole eingefügt werden müssen.
Durch die gegenseitige Optimierung der Tools aufeinander wird dieser Nachteil jedoch wenigstens etwas vermindert.