Beschreibung der Diplomarbeit

Andreas Zahnert

17. Januar 1996

1 Der Rahmen

Das Projekt B4 “Efficient compilation of HPSG grammars” des SFB 340 befaßt sich mit dem Parsingproblem fuer Grammatiken der HPSG (Head-Driven Phrase Structure Grammar) Theorie. Das Ziel ist eine Modellierung von Grammatiken, die sich möglichst genau an der linguistischen Theoriebildung orientiert. Dies macht das Parsingproblem besonders schwierig, da linguistische Grammatiken im allgemeinen ohne Rücksicht auf Implementierbarkeit entwickelt werden.

Das Parsingproblem für uneingeschränkte HPSG Grammatiken ist unentscheidbar. Dies ist analog zu anderen logischen Grammatikformalismen (definite clause grammars, PATR, etc.) und Logikprogrammen im allgemeinen. Es können jedoch hinreichende (schwache) Beschränkungen für Grammatiken formuliert werden, die das Parsingproblem entscheidbar machen.

Um die rechnerische Komplexitaet des Parsingproblems praktisch in den Griff zu bekommen, gehen wir in drei Schritten vor:

1) Kompilierung einer Grammatik in ein Logikprogramm
2) Off-line-Optimierung des Logikprogramms (Programmtransformation)
3) Ausführung des optimierten Logikprogramms.

Schritt 3) wird bisher mit einem in Prolog geschriebenen Interpreter abgedeckt. Dies kann durch einen speziellen Compiler natürlich stark verbessert werden.

2 Der Compiler

Die Grundkonzeption basiert, wie bei heutigen Prolog–Compilern üblich, auf einer abstrakten Maschine. Sie dient als Basis für ein spezielleres, auf das gegebene Problem zugeschnittenes, objektorientiertes Konzept. Da die Programme jedoch automatisch erzeugt und dabei sehr groß werden, stehen dem hohe Anforderungen an die Speicheroptimierung gegenüber — bei einer Million Elementen auf dem Heap fallen z.B. kaschierte Objektdaten für jedes Element stark ins Gewicht. An solchen Stellen wurde auf C zurückgegriffen; da dies jedoch möglichst lokal gehalten wurde, sind die Konzeption und ein Großteil der Implementierung trotzdem objektorientiert.

Der Compiler läßt sich grob in zwei Komponenten aufgliedern:

- den Übersetzer
- die Laufzeitumgebung

Der Übersetzer erzeugt hierbei lediglich den für die vom Benutzer programmierten Regeln spezifischen Code (z.B. Aufrufe anderer Regeln, Einsprungpunkte beim FAIL etc.). Dieser hat eine sehr abstrakte Sicht auf die Laufzeitumgebung, so daß hier eine möglichst weitreichende Unabhängigkeit erreicht werden kann.

Zu den Aufgaben der Laufzeitumgebung gehört unter anderem, Anfragen des Benutzers an das Programm zu interpretieren. An dieser und anderen Stellen können Objekte so gestaltet werden, daß sie für beide Komponenten verwendbar sind. Diese Codebasis wird möglichst groß gehalten.

Die Laufzeitumgebung muß all die Aufgaben ausführen, die der erzeugte Code nur auf sehr abstraktem Niveau anstößt. Hierzu gehören die Aufrufe von Regeln, das dazu nötige Verwalten eines Laufzeitstacks, Backtracking, Speicherverwaltung, die "physikalische" Repräsentation der abstrakten Variablen usw. Daraus ergibt sich folgende grobe Einteilung in folgende Komponenten:

- Laufzeitstack
- Heap
- Unifier
- Rulecaller
- Trailer (für Backtracking)
- QueryInterpreter

3 Die Diplomarbeit

Die Diplomarbeit besteht zum einen aus der Fertigstellung des Compilers, zum anderen soll eine Spracherweiterung vorgenommen werden.

Es dürfte bei beiden Aufgabenteilen praktisch nicht möglich sein, die Arbeit in der Diplomarbeit selbst völlig unabhängig von dem schon vorher Entstandenen zu betrachten. So steckt z.B. in der Aufgabe, den Laufzeitstack auf relative Adressierung umzustellen, um eine dynamische Größenveränderung zu erlauben, für sich gesehen keine allzu große Leistung. Die eigentliche Leistung besteht vielmehr in der Gesamtkonzeption, die ein schrittweises Entwickeln und Testen auf Basis wohldefinierter Schnittstellen erlaubt. Da der gesamte Compiler von mir ohne Hilfe anderer konzeptioniert und entwickelt wurde, wäre es schön, wenn auch diese Leistung in die Bewertung einfließen könnte.

3.1 Fertigstellung

Noch zu erstellende Komponenten sind (abgesehen von der gerade genannten kleinen Änderung):

- die Fertigstellung des Unifiers,
- die Entwicklung des Trailers,
- des QueryInterpreters
- und der Verwaltung des Heaps (insbesondere Garbage Collection).

Dabei ist zu beachten, daß alles bisher entwickelte bis jetzt nur sehr oberflächlich und als Komponenten getestet werden konnte. Es müssen also auch zahlreiche Fehler im Gesamtsystem ausgemerzt werden.

3.2 Spracherweiterung

Die Spracherweiterung ist in der Logikprogrammierung unter dem Begriff Concurrency bekannt. Eigentlich hat der Benutzer keinen Einfluß auf die Reihenfolge der Regelaufrufe, die auf der rechten Seite einer Regel spezifiziert sind. So können z.B. bei der Regel

$$p(X,Y) \text{ :- } q_1(a,X),\, q_2(X,Y).$$

q_1 und q_2 parallel abgearbeitet werden. In der Praxis wäre jedoch ein Kopieren von dem eventuell gegebenen X nötig; dieser Aufwand steht meist in keinem Verhältnis zur Parallelisierung. Die neue Kontrollstruktur soll erlauben, die Ausführung zu synchronisieren.

$$p(X,Y) \text{ :- } q_1(a,X),\, \mathit{freeze}(X, q_2(X,Y)).$$

soll bewirken, daß q_2 erst dann aufgerufen wird, wenn X gebunden ist. Ist p so definiert, daß X ein Rückgabewert ist, so wird q_1 vor q_2 ausgeführt. Auch wenn man auf Parallelisierung verzichtet, erlaubt dies eine von der Reihenfolge der Literale unabhängige Festlegung der Ausführungsreihenfolge.

Diese Modifikation zieht Änderungen in großen Teilen des Compilers nach sich, deren Umfang im Moment schwer abschätzbar sind.