Praktischer Übersetzerbau

Von Prof. Dr. rer. nat. Ernst-Erich Doberkat,
Universität Essen
und Dr. rer. nat Dietmar Fox,
Universität Hildesheim

Springer Fachmedien Wiesbaden GmbH

Prof. Dr. rer. nat. Ernst-Erich Doberkat

Geboren 1948 in Breckerfeld/Westfalen. Von 1968 bis 1973 Studium der Mathematik und Philosophie an der Ruhr-Universität Bochum, von 1973 bis 1976 wiss. Mitarbeiter am Forschungs- und Entwicklungszentrum für objektivierte Lehr- und Lernverfahren GmbH in Paderborn, 1976 Promotion in Mathematik an der Universität Paderborn. Von 1976 bis 1981 Assistent in Bonn und Hagen, 1980 Habilitation für Informatik an der FernUniversität. 1981 Associate Professor of Mathematics and Computer Science, Clarkson College of Technology, Potsdam, New York, 1985 ordentlicher Professor für Praktische Informatik an der Universität Hildesheim, seit 1988 ordentlicher Professor für Informatik/Software Engineering an der Universität Essen.

Dr. rer. nat. Dietmar Fox

Geboren 1953 in Essen. Von 1973 bis 1979 Studium der Informatik und Mathematik an der RWTH in Aachen, 1979 wiss. Mitarbeiter im Lehrgebiet Programmiersprachen/Formale Sprachen der FernUniversität in Hagen, 1983 Promotion in Informatik an der FernUniversität, 1985 Akademischer Rat am Lehrstuhl für Praktische Informatik A der Universität Hildesheim.

CIP-Titelaufnahme der Deutschen Bibliothek

Doberkat, Ernst-Erich:
Praktischer Übersetzerbau / von Ernst-Erich Doberkat u. Dietmar Fox. – Stuttgart : Teubner, 1990
ISBN 978-3-519-02288-6 ISBN 978-3-322-94714-7 (eBook)
DOI 10.1007/978-3-322-94714-7
NE: Fox, Dietmar:

Ursprünglich erschienen bei B. G. Teubner Stuttgart 1990

Vorwort

Wir berichten in diesem Buch über ein Praktikum, in dem der Übersetzer für eine Programmiersprache implementiert worden ist. Das Praktikum sollte den Teilnehmern zeigen, wie man von einer abstrakt vorgegebenen Sprachdefinition zu einem funktionsfähigen Compiler gelangen kann. Wir haben dieses Praktikum im Sommersemester 1988 am Institut für Informatik der Universität Hildesheim durchgeführt; die Teilnehmer an diesem Praktikum waren Studenten im Diplom-Studiengang Informatik, die sich im sechsten oder achten Fachsemester befanden. An Vorkenntnissen wurde in diesem Praktikum vorausgesetzt:

- Kenntnis der Programmiersprache C und einiger Werkzeuge unter UNIX,
- Kenntnisse aus dem Bereich der Programmiersprachen und des Übersetzerbaus, wie sie etwa in einer vierstündigen Hauptvorlesung "Programmiersprachen und Übersetzerbau I" in Diplom-Studiengängen der Informatik vermittelt werden,
- elementare Kenntnisse der Linearen Algebra.

Warum nun ein Übersetzer für eine Sprache, die sich mit Fragestellungen der Linearen Algebra befaßt? Zunächst ist hier anzumerken, daß die Lineare Algebra nicht als Selbstzweck betrachtet wurde; in der mathematischen Grundausbildung für Informatiker wird Lineare Algebra gelehrt, daher haben Informatik-Studenten Grundkenntnisse und brauchen sich dort nicht weiter einzuarbeiten. Könnte man sich darauf verlassen, daß Informatik-Studenten einen gleichmäßigen Kenntnisschatz etwa der Manipulation dreidimensionaler geometrischer Objekte hätten, hätte man ohne Zweifel auch eine Programmiersprache hierauf aufbauen können.

Bei diesem Praktikum kam es uns darauf an, Zugänge aufzuzeigen, mit deren Hilfe man die praktischen Probleme der Übersetzung kleinerer Programmiersprachen beherrschen kann. Diese kleinen Programmiersprachen werden bekanntlich in der Praxis häufig gebraucht, wenn es darum geht, einen Prozessor für eine spezialisierte Problemstellung zu konstruieren, etwa wenn sich die Problemstellung gut mit Hilfe kontextfreier Grammatiken beschreiben läßt, wie es beispielsweise im Bereich der Benutzerschnittstellen oder bei dedizierten Datenbank-Abfragen der Fall sein kann.

Wir wollen mit diesem Projekt-Bericht aber auch zeigen, wie sich die gängigen Techniken des Software Engineering an einem überschaubaren Beispiel realisieren lassen: auf der einen, der abstrakten Seite finden wir die Aufgaben des Compilers, auf der anderen, der konkreten Seite stehen wir vor der Notwendigkeit, diese Aufgaben realisieren zu müssen. Die Dekomposition des Problems ergibt in natürlicher Weise ein Skelett für den Entwurf der Lösung. Das wird im vorliegenden Projekt handgreiflich in einem noch überschaubaren Rahmen demonstriert.

Welche Zielgruppe haben wir nun mit diesem Praktikum im Auge? Zunächst wollen wir interessierten Kollegen an Universitäten oder Fachhochschulen Einblick in ein erfolgreich abgewickeltes Vorhaben geben, das sich von der Größe gerade noch in einem Semester mit Studenten kontrolliert realisieren läßt. Wir wollen aber auch Studenten der Informatik einen Einblick in die praktischen Techniken geben, wenn sie vor einem ähnlichen Problem stehen. Gleiches gilt für den in der Praxis stehenden Informatiker, der gelegentlich in die Verlegenheit kommt, eine eigene kleine Sprache realisieren zu müssen. Dadurch daß wir die Realisierung des Übersetzers in großem Detail schildern und damit einen Protoypen für die Realisierung ähnlicher Vorhaben liefern, ist es nicht allzu schwierig, in ähnlich gelagerten Problemstellungen auch ähnlich vorzugehen. Schließlich wenden wir uns an Informatiklehrer in der Oberstufe von Gymnasien oder Gesamtschulen, die sich mit praktischen Aspekten des Übersetzerbaus befassen und gleichzeitig die konkrete Implementation eines solchen Übersetzers studieren möchten. Wir beschäftigen uns dazu im ersten Kapitel mit den Zielsetzungen dieses Projekts, beschreiben, welche Zielvorstellungen wir mit diesem Programmierpraktikum verbinden, gehen kurz auf Vorkenntnisse, Werkzeuge und Realisierungsmöglichkeiten ein und schildern dann die Organisationsform dieses Praktikums.

Ein kurzer Überblick über den weiteren Inhalt des Buchs soll nun folgen. Wir geben in sehr gestraffter und komprimierter Form einen Überblick über die Arbeitsweise eines Compilers, wie er sich für uns darstellt, und diskutieren die Aufgaben des Compilers sowie ihre programmtechnische Realisierung hauptsächlich unter dem Gesichtspunkt der Programmierwerkzeuge. Nach einer kurzen Erinnerung an die Phasen eines Compilers diskutieren wir die lexikalische, die syntaktische und die semantische Analyse, wobei wir in jeder dieser Phasen zunächst kurz die mathematischen Grundbegriffe zitieren und für wichtige Spezialfälle bei der syntaktischen und bei der semantischen Analyse auf die zugrundeliegenden Konstruktionen eingehen. Wir diskutieren also bei der syntaktischen Analyse die Prinzipien für einen einfachen Parser-Generator, der die Prinzipien der bottom-up-Analyse verdeutlichen soll, und beschreiben bei der semantischen Analyse die Grundlage für ein populäres Verfahren der Attribut-Auswertung mit Hilfe von geordneten attributierten Grammatiken. Schließlich kommen wir auf Laufzeit-Umgebungen zu sprechen und erinnern an die fundamentalen Tafeln, Bindungen von Werten an Variablen, Übergabe von Parametern, activation records und schließlich an die Allokation von Speicherplatz.

An diesen Abschnitt schließt sich die Beschreibung der Sprache *LA* an. Wir haben uns bemüht, diese Sprachbeschreibung als Manual für die Sprache *LA* zu gestalten. Da wesentlich die Konstruktion von Algorithmen für die Standard-Operationen aus der Linearen Algebra benötigt werden (als *domain knowledge*), geben wir in dem folgenden Kapitel die Spezifikation einiger wichtiger Algorithmen für diese Standard-Operationen an. Genannt seien hier exemplarisch Operationen zur Bestimmung des Kerns einer Matrix oder zur Berechnung der Exponentialfunktion für ein quadratische Matrix.

Diese Vorarbeiten sind nötig, um in die Konstruktion des Compilers einzusteigen. Zunächst diskutieren wir die Analyse der lexikalischen und der syntaktischen Struktur, wobei wir die Grammatik für *LA* gleich so spezifizieren, daß sie als Eingabe in den Parser-Generator *yacc* dienen kann. Auf die lexikalische und syntaktische Analyse folgt die semantische Analyse, die im nächsten Abschnitt diskutiert wird. Wir zeigen, wie der abstrakte Syntaxbaum aufgebaut ist, diskutieren die Hauptfunktionen für die semantische Analyse und gehen auf die Arbeitsweise einiger wichtiger Hilfsfunktionen ein. Als nächstes beschäftigen wir uns mit der Erzeugung von Code. Wir erzeugen Code für eine abstrakte Maschine, deren Assembler spezifiziert wird. Zunächst wird für die meisten Konstrukte der Sprache die Code-Erzeugung

spezifisch diskutiert. Das vorletzte Kapitel befaßt sich mit der abstrakten Maschine, diskutiert hier die Speicherverwaltung und die spezifische Laufzeit-Umgebung und stellt dann die einzelnen Operationen der abstrakten Maschine zur Verfügung. Bemerkungen über die Implementierung dieser Maschine und über die Speicherbereinigung schließen das Kapitel ab. Das letzte Kapitel zeigt Erweiterungsmöglichkeiten für Compiler und Sprache auf.

In zwei Anhängen werden weitere nützliche Informationen geliefert, zum einen findet sich ein Beispiel-Programm in Anhang A, zum anderen sind die Grundbegriffe der Linearen Algebra, wie wir sie hier benötigen, in Anhang B zusammengefaßt. Das Beispiel-Programm soll dazu dienen, dem Leser ein Gefühl für die Arbeitsweise und die Ausdrucksfähigkeit von *LA* zu vermitteln.

Das vorliegende Buch ist nicht allein das Werk seiner beiden Autoren, sondern hat von der Arbeit der am Praktikum beteiligten Studenten profitiert. Wir möchten gerne den folgenden Studenten für ihre Mitarbeit danken:

Asmus Bumann	Markus Ebigt	Thomas Ernst
Wolfgang Fischer	Manfred Friese	Matthias Gevers
Ralf Gieseke	Sebastian Heckler	Andreas Jährig
Stefan Kropp	Matthias Lübberstedt	Ulrich Lammers
Bernd-Uwe Pagel	Holger Schirnick	Reinhard Schmoldt
Hans-Gerald Sobottka	Frank Weidele	Janet Wundenberg
Sing Young		

Mit dem Wechsel des älteren der beiden Verfasser an die Universität Essen wechselte auch der Schwerpunkt der Beschäftigung mit *LA* und diesem Buch (bis auf Kap. 4, 6, 7.3, Anhang A) ins Ruhrgebiet. Wir möchten uns bei Hartmut Henning für seine Portierung des Compilers auf den Apple/Macintosh bedanken, bei Stefani Kamphausen dafür, daß sie einen Teil des Manuskripts geschrieben hat. Unser besonderer Dank gilt Ingrid Kleinstoll-Snoussi für die große Sorgfalt, mit der sie den Text editiert und aus teilweise ziemlich unleserlichen Vorlagen ein lesbares Manuskript gezaubert hat, aber vor allem dafür, mit wachem Auge verhindert zu haben, daß sich stilistische oder typographische Ungereimtheiten in den endgültigen Text eingeschlichen haben.

In Hildesheim hat sich Markus Ebigt der gelegentlich undankbaren Aufgabe unterzogen, den Code von Fehlern und Inkonsistenzen zu bereinigen. Er hat auch das Beispiel im Anhang A implementiert und getestet. Ulrich Gutenbeil war einer der ersten Konsumenten des Texts; er hat das *front end* des Compilers für *LA* mit dem Werkzeug Eli zu Studienzwecken neu implementiert (vgl. [Gut90]) und im Laufe dieser Arbeiten einige Verbesserungsvorschläge zur Darstellung gemacht. Beiden sei an dieser Stelle herzlich gedankt.

Schließlich möchten wir uns bei Herrn Dr. Spuhler vom Teubner-Verlag für die wiederum sehr angenehme Zusammenarbeit bedanken.

Essen und Hildesheim, im Juni 1990 — Die Verfasser

Inhaltsverzeichnis