Der Parsergenerator BTRACC2

Uwe Erdmenger
pro et con Innovative Informatikanwendungen GmbH, Annaberger Straße 240, 09125 Chemnitz
uwe.erdmenger@proetcon.de

Eine Voraussetzung für Reengineering- und Migrationsprojekte ist die Analyse der Quelltexte, Scripten, JCLs etc. des betreffenden Systems. Für die Entwicklung der dazu notwendigen Werkzeuge werden Parsergeneratoren verwendet, welche aus einer formalen Syntaxbeschreibung einen Parser generieren. Der Parsergenerator BTRACC, der nunmehr in der Version 2 vorliegt, wurde von der Firma pro et con im Hinblick auf die besonderen Anforderungen bei der Analyse von Legacy-Software entwickelt. BTRACC2 und seine Anwendung in kommerziellen Projekten sind Gegenstand der folgenden Ausführungen.

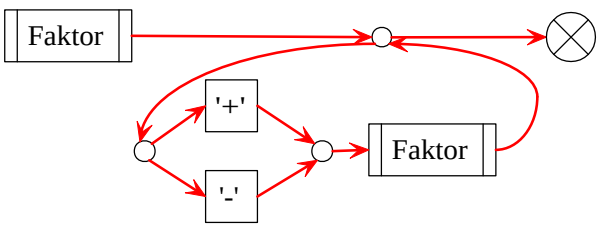
1 Motivation

Antiquierte Programmiersprachen von Legacy-Systemen wie z.B. COBOL, PL/I oder SPL zeichnen sich durch eine komplexe Grammatik aus, die auch syntaktische Mehrdeutigkeiten enthalten kann. Eine Anforderung an BTRACC2 war es, diese Grammatiken ohne aufwendige Umformungen zum Erreichen einer LL(n)- oder LR(n)-Eigenschaft verarbeiten zu können. Zusätzliche Forderung ist die Verarbeitung mehrerer Dialekte in einem Parser. Außerdem zeigen Erfahrungen die Notwendigkeit häufiger Änderungen und Erweiterungen des Parsers im Nutzungsprozeß. BTRACC2 wurde im Hinblick auf diese Anforderungen entwickelt und erweitert.

2 Arbeitsweise

BTRACC2 generiert Parser, die auf Basis eines Backtracking-Verfahrens (LL-Parser mit Backtracking) arbeiten. Die Syntax wird in einer um die Attributierung ergänzten EBNF (Erweiterten Backus-Naur-Form) notiert. Während der Verarbeitung der Grammatik durch BTRACC2 wird daraus zu jeder Regel ein Graph als interne Abbildung erstellt. Es existieren Knoten für **Terminalsymbole**, **Nichtterminale** (Aufruf anderer Regeln) und **Verzweigungs-** bzw. **Sammelknoten**. Jeder Knoten hat maximal zwei Nachfolger. Das folgende Bild zeigt den Graphen der Regel

```
Term = Faktor { ('+' | '-') Faktor }.
```



Diese Graphen werden nun nach Zyklen (auch übergreifend über mehrere Regeln) durchsucht, die z.B. bei linksrekursiven Regeln auftreten. Diese Grammatiken können mit dem zugrundeliegenden Verfahren nicht weiterverarbeitet werden. Im nächsten Schritt werden für alle Verzweigungsknoten die FIRST- und FOLLOW-Mengen der beiden abgehenden Kanten berechnet. Nach diesen Vorbereitungen beginnt die eigentliche Generierung der Zielsprache. Die spätere Syntaxanalyse des generierten Parsers wird dabei durch eine Reihe von Befehlen realisiert. Die wichtigsten sind $TEST$ (aus Terminalsymbolen generiert), $CALL$ (aus Nichtterminalen), $BRANCH$ (aus Verzweigungen) und $RETURN$ (aus dem Regelende). Sie werden zur Laufzeit von einer virtuellen Maschine (VM) wie folgt abgearbeitet: Wenn bei $BRANCH$ anhand des aktuellen Tokens sowie der FIRST- u. FOLLOW-Mengen keine Entscheidung für den weiteren Weg möglich ist, wird die erste Möglichkeit verwendet und die zweite auf einem Stack gemerkt. Beim Befehl $TEST$ wird das aktuelle Token mit dem Terminalsymbol verglichen. Sind sie nicht gleich, wird der Zustand aus dem letzten Rücksetzpunkt auf dem Stack wiederhergestellt und dort weitergearbeitet (Backtracking). $CALL$ und $RETURN$ werden wie bei imperativen Programmiersprachen abgearbeitet, allerdings wird der Stack bei RETURN nicht verkleinert, da dabei Rücksetzpunkte verloren gingen. Die Attributberechnung wird durch eine Menge generierter, rekursiver Funktionen realisiert. Sie wird nach der Syntaxanalyse (Abarbeitung der VM) ausgeführt und kann deterministisch (ohne Backtracking) realisiert werden. Der verbliebene Stackinhalt der VM dient als “roter Faden”.

3 Eigenschaften

BTRACC2-Parser arbeiten trotz des Backtracking-Verfahrens effizient. Grund ist die Effizienz der VM und die Tatsache, dass Attributberechnungen nicht zurückgenommen werden müssen. Bei der Analyse syntaktisch fehlerhafter Programme kann der Fall eintreten, dass viele Backtrackings sinnlos ausgeführt werden, bevor die Syntaxanalyse fehlschlägt. Zur Begrenzung solcher Fälle wird ein so genannter *CUT* verwendet, wie er auch aus der Programmiersprache Prolog bekannt ist. Er verwirft alle Rücksetzpunkte, die nach dem Beginn der Abarbeitung der aktuellen Regel angelegt wurden. Wenn von der Logik her klar ist, dass diese Fälle nicht eintreten können (z.B. durch Auftreten eines bestimmten Schlüsselwortes), kann dieser Operator zur Effektivierung des Parsers eingesetzt werden. BTRACC2 kennt auch so genannte *semantische Prädikate*. Das sind Ausdrücke mit booleschem Wert, die von der VM durch spezielle Befehle ausgeführt werden. Geben sie `TRUE` zurück, wird mit dem nächsten Befehl fortgesetzt. Wenn nicht, wird Backtracking ausgelöst. Das kann z.B. verwendet werden, um Regeln verschiedener Dialekte wahlweise “an- und abzuschalten”. Allerdings können die Prädikate nicht auf die Attribute zurückgreifen,

da diese zur Abarbeitung der VM noch nicht vorliegen. Der Befehl TEST vergleicht *Tokenarten*. Manchmal ist jedoch ein Vergleich mit dem *Tokenwert* (i.A. ein String) sinnvoll (z.B. zur Erkennung der Stufennummer 78 bei Konstantendefinitionen in COBOL). BTRACC2 erlaubt solche Tests, indem der zu prüfende Wert in doppelten Hochkommas in der Grammatik formuliert wird. Auch der Vergleich mit regulären Ausdrücken ist möglich.

4 Erweiterungen in der Version 2

In der aktuellen Version wurde BTRACC2 komplett überarbeitet. Ein wesentliches Ziel war, den modularen Aufbau zu verbessern und die Erweiterbarkeit zu gewährleisten. Der wichtigste Aspekt dabei ist die Trennung in eine allgemeingültige und eine von der Zielsprache abhängige Komponente. Der Aufbau der internen Darstellung, die Tests und Berechnungen sind allgemeingültig, und nur die Generierung der Zielsprache ist sprachspezifisch. Ziel war es, die Generierung von Parsern in unterschiedlichen Programmiersprachen zu ermöglichen. Aktuell kann mit BTRACC2 Parserquelltext in den Sprachen C, C++ und Java generiert werden. Die generierten Funktionen bzw. Methoden für die Berechnung der Attribute enthalten in der aktuellen Version strukturierten Code und verzichten im Gegensatz zum Vorgänger weitestgehend auf die Verwendung von Sprüngen (*goto*). Die Strukturierung orientiert sich dabei an der EBNF-Eingabe (Wiederholungen werden zu Schleifen, Alternativen zu if-Anweisungen usw.). Das vereinfacht das Debuggen der Attributberechnung. Auch die Syntax von BTRACC2 wurde erweitert. Die wichtigsten Erweiterungen sind die Unterstützung der mindestens einmaligen Wiederholung (ausgedrückt durch ein $+$) und die m- bis n-fache Wiederholung. Neben der Verwendung von Einzeltoken als Terminalsymbole können nun auch *Tokenmengen* verwendet werden. Sie werden durch Angaben der Art

```
tokenset S1 = ANY - {ADD, SUB, MUL, DIV}.
```

definiert und wie Terminalsymbole in den Regeln verwendet. Die vordefinierte Menge ANY umfasst alle im Parser verwendeten Token. Tokenmengen sind vor allem dort sinnvoll, wo eine Reihe von Schlüsselworten als Bezeichner verwendet werden dürfen, z.B. bei Gerätenamen in COBOL. Durch die vollständige Berechnung von FIRST- und FOLLOW-Mengen konnte eine Reduzierung der Backtracking-Fälle und eine weitere Verbesserung der Laufzeit gegenüber der Originalversion erreicht werden.

5 Anwendungsprojekte

BTRACC wurde und wird für die Generierung von Parsern genutzt, welche in Projekten der Firma pro et con zum Einsatz kommen. Die seit Ende 2008 fertiggestellte Version 2 wird in einem Java-Frontend eingesetzt, welches in das firmeneigene Reverse Engineering Tool Flow Graph Manipulator (FGM) [3] integriert ist. Zusätzlich kommt es im Werkzeug Rochade der Firma ASG zum Einsatz.

Seit April 2009 läuft im Rahmen von IKT das vom Bundesministerium für Bildung und Forschung geförderte Verbundprojekt SOAMIG (Migration in serviceorientierte Architekturen) [2]. Partner sind neben pro et con das Institut für Softwaretechnik (IST) der Universität Koblenz-Landau und die Amadeus Germany GmbH. Dieses FuE-Projekt hat zum Ziel, Legacy-Systeme werkzeugunterstützt durch Transformation in eine SOA zu migrieren. In dieses Projekt wird das Java-Frontend eingebunden. Da in SOAMIG ebenfalls die Aufgabe besteht, Legacy-Systeme auf COBOL-Basis zu bearbeiten, wird das von pro et con entwickelte COBOL-Frontend unter Verwendung von BTRACC2 reimplementiert und erweitert.

6 Zusammenfassung und Ausblick

Unsere Erfahrungen belegen, dass mit BTRACC2 ein ausgetesteter, praktisch anwendbarer Parsergenerator zur Verfügung steht, welcher in einer Reihe von Reengineering- und Migrationsprojekten eingesetzt wurde und wird [3].
Die Entwicklung von BTRACC ist nicht abgeschlossen, weitere Verbesserungen und Erweiterungen sind vorgesehen. Zunächst sollen zusätzliche Programmiersprachen unterstützt werden. Vor allem die Generierung von Perl-Parsern ist von Interesse. Diese Sprache wird u.a. für die Analyse von Shell-Scripten, Batch-Programmen und JCLs eingesetzt. Hier soll BTRACC perspektivisch den bisher verwendeten Parsergenerator `Parse::RecDescent` ablösen. Während die zweite Phase des Parsing-Prozesses (Attributberechnung) gut mit einem Debugger für die entsprechende Programmiersprache überwacht werden kann, ist dessen Anwendung auf die Abarbeitung der VM wenig hilfreich. Daher soll ein visueller Debugger für diese Phase realisiert werden. Ziel ist es, eine sprachunabhängige Implementation in Java zu realisieren, die über eine TCP-Schnittstelle mit der VM kommuniziert. Eine weitere, sinnvolle Erweiterung ist die Integration eines Profilers, welcher die Anzahl der Einzelbefehle und der Backtrackings für jede einzelne Regel bei einer konkreten Eingabe ermittelt. Damit können weitere Optimierungsmöglichkeiten analysiert werden.

Literaturverzeichnis

[1] Erdmenger, U.; Kaiser,U.; Loos, A.; Uhlig, D.:
Methoden u. Werkzeuge für die Software Migration.
In: Gimnich, R.; Kaiser, U.; Quante, J.; Winter, A. (Eds.): 10th Workshop Software-Reengineering (WSR 2008), 5-7 May 2008, Bad Honnef.
Lecture Notes in Informatics, (LNI)-Proceedings, Volume P-126, S. 83-97

[2] SOAMIG - Migration von Legacy-Software in service-orientierte Architekturen.
Vorhabenbeschreibung im Rahmen des IST-Verbundprojektes SOAMIG, April 2009

[3] Beier, A.:
Flow Graph Manipulator (FGM) 3.0 - Reverse-Engineering-Tool für komplexe Softwaresysteme.
11. Workshop Software Reengineering, Bad Honnef, 4.-6. Mai 2009