GENERIC INTERPRETER

von Christian Gierds

ALLGEMEINES

Der *„Generic Interpreter“* (http://www.csupomona.edu/~carich/gi/) ist ein andauerndes Forschungsprojekt von Prof. Craig A. Rich, der an der California State Polytechnic University, Pomona arbeitet. Er will mit diesem Werkzeug die Möglichkeit bieten, kontextfreie Grammatiken „on the fly“ zu parsen.
Zum Zeitpunkt des Vortrages lag es in der Version 1.0 vor.
Das Paket kommt in Form eines Java-Archives, wahlweise mit Quelltext, dessen Klassen einfach in die Projekte eingebunden werden und somit sofort nutzbar sind.

FÄHIGKEITEN

Mit Hilfe des *„Generic Interpreter“* ist es möglich, Grammatiken der Typen LL(1), LR(0), SLR(1) sowie LR(1) zu implementieren und zu parsen, wobei man betonen muss, dass durch die Nutzung des Klassenkonzepts von Java, die entsprechende Grammatik erst zu Laufzeit erzeugt wird.
Diese Umsetzung eines Parsers bietet die beabsichtigte Eigenschaft, Grammatiken „on demand“, während der Abarbeitung des Programms zu erzeugen und zu parsen. Außerdem kann damit die Grammatik nachträglich geändert werden.
Es gibt jedoch auch einige Nachteile: Relativ unbedeutend ist die Einschränkung auf Grammatiken, die einen maximalen Look-a-head von eins haben, da viele relevante Grammatiken, z. B. von gängigen Programmiersprachen, in diese Klasse fallen.
Was jedoch momentan als gravierend angesehen werde sollte, ist die Tatsache, dass der *„Generic Interpreter“* bei Konflikten weder Warnungen noch Fehler ausgibt, sodass ein falsche Implementierung einer Sprache nur schwer oder gar nicht erkannt werden kann.
Stattdessen wird bei Konflikten in LL-Grammatiken sowie bei Reduzier-/ Reduzierkonflikten die letzte passende Regel gewählt, und bei Schiebe-/ Reduzierkonflikten in LR-Grammatiken wird standardmäßig geschoben.

GRAMMATIK EINBINDEN

Die Grammatiktypen liegen wie bereits erwähnt in Form von Klassen vor, und eine neue Grammatik wird in Form einer Klassenerweiterung eines dieser Typen realisiert. Nun stehen verschiedene Methoden bereit, um Lexik, Syntax und Semantik einer Sprache zu realisieren.

LEXIK

Zum Einbinden von lexikalischen Ausdrücken steht die Methode `Lexicon.put(...)` zu Verfügung (Details siehe Homepage).
Mittels eines DFAs, nach dem Prinzip des „longest match“, werden dann durch den Lexer Token erzeugt.

SYNTAX

Zum Einbinden syntaktischer Regeln muss die Methode `Grammar.put(...)` benutzt werden (Details siehe Homepage).
Das Startsymbol der Grammatik wird in der ersten Regel erwartet.
Eine Grammatikregel kann meist leider nicht direkt umgesetzt werden, da der *„Generic Interpreter“* keine Unterstützung für Grammatiken in EBNF bietet. Das heißt, dass z. B. Regeln mit optionalen Symbolen, oder Teilregeln, auf die der Plus- bzw. Sternoperator angewendet wird, durch umstellen der Regeln in mehrere Alternativen bzw. rekursive Listen realisiert werden müssen.
Somit gibt es jedoch große Parallelen zu der Art, wie die Syntax einer Sprache in *„yacc“/ „bison“* ein gebunden wird.

SEMANTIK

Auch zum Einbinden von semantischen Aktionen stellt der *„Generic Interpreter“* eine Klasse bereit.
Die Semantik wird mit Hilfe von Variablen des Typs `Semantics` umgesetzt. In dieser Klasse wird beim Nutzen der Variable für entsprechende Aktionen standardmäßig die Methode `Semantics.evaluate(...)` aufgerufen (Details siehe Homepage).
Da diese Methode per definitionem einen leeren Methodenkörper hat, sollte sie beim Definieren der Variable überschrieben werden. Als Parameter erhält sie den aktuellen Parsebaum, der mit Hilfe der Klasse `ParseTree` aufgebaut wird.
Da die Semantik durch Variablen realisiert wird, muss sie vor der syntaktischen Regel definiert werden, in der sie später benutzt wird, indem sie einfach in die Aufzählung der Terminale und Nicht-Terminale eingefügt wird.

PARSEN

Der eigentliche Parseprozess wird initiiert, in dem der Methode `Grammar.interpret(...)` ein String oder ein Datenstrom übergeben wird.
Daraufhin werden aus der Eingabe Token entsprechend der Lexik generiert, diese werden dann versucht in entsprechende syntaktische Regeln zu überführen, wobei dort integrierte semantische Aktionen ausgeführt.
Bei einem Fehler bricht das Programm sofort ab, ansonsten wird die Methode normal beendet.