Zutaten für erweiterbare Programmiersprachen

Christian Heinlein
Studiengang Informatik
Fakultät Elektronik und Informatik
Hochschule Aalen – Technik und Wirtschaft

Erweiterbare Programmiersprachen erlauben ihrem Benutzer, die Syntax der Sprache mehr oder weniger umfassend zu erweitern, beispielsweise durch die Definition zusätzlicher Operatoren und Kontrollstrukturen oder durch die Einführung neuer Deklarationsmöglichkeiten. Üblicherweise werden neue syntaktische Konstruktionen durch einen mehr oder weniger komfortablen und flexiblen Ersetzungsmechanismus auf vordefinierte Sprachmittel zurückgeführt.

Beispielsweise bietet der C/C++-Präprozessor die Möglichkeit, Makroaufrufe im Programmtext durch ihre (ggf. parametrisierte) Definition zu ersetzen und damit in begrenztem Maße syntaktische Erweiterungen der Sprache vorzunehmen. Common Lisp bietet einen wesentlich mächtigeren Makromechanismus an, der nahtlos in die Basissprache integriert ist, so dass der Ersatztext eines Makros bei Bedarf erst zur Laufzeit des Programms berechnet werden kann. Da Common Lisp außerdem eine sehr einfache und einheitliche syntaktische Grundstruktur besitzt (geklammerte Ausdrücke bestehend aus einem Funktions- oder Operatorsymbol und beliebig vielen Operandenausdrücken), lassen sich Erweiterungen ebenfalls nahtlos integrieren.

Dylan bietet ebenfalls einen mächtigen Makromechanismus an, bei dem zwischen Definitions-, Anweisungs- und Funktionsmakros unterschieden wird, mit denen unterschiedliche Teile der zugrundeliegenden Grammatik erweitert werden können. Moderne funktionale Sprachen wie z. B. ML oder Haskell bieten die Möglichkeit, benutzerdefinierte Operatorsymbole einzuführen und anschließend zu verwenden. Andere moderne Sprachen wie z. B. Scala oder Seed7 bieten ebenfalls bestimmte, zum Teil relativ weitreichende Erweiterungsmöglichkeiten.

Alle oben genannten Ansätze erlauben Erweiterungen allerdings nur innerhalb bestimmter Grenzen, die durch die syntaktische Grundstruktur der Sprache vorgegeben sind. Ausgehend von Ausdrücken als syntaktisches Grundmuster, die ganz allgemein aus Operanden und Operatoren aufgebaut sind, lassen sich jedoch Sprachen konzipieren, die nahezu beliebig syntaktisch erweiterbar sind:

1. Damit die Syntax von Ausdrücken beliebig erweiterbar ist, muss es nicht nur möglich sein, vordefinierte Operatorsymbole (wie z. B. in C++) zu überladen, sondern nach Belieben *neue Operatorsymbole* einführen zu können. Neben den bekannten Präfix-, Infix- und Postfixoperatoren sollten beliebige *Operatorkombinationen* (wie z. B. `...?...:...` in C/C++ oder `select...from...where...` in SQL) definierbar sein.
2. Da man Anweisungen wie z. B. `if...then...else...end` oder `repeat...until...` ebenfalls als Operatorkombinationen interpretieren kann, ist damit auch die Syntax von Anweisungen beliebig erweiterbar.
3. Da man auch Typen wie z. B. `int`, `int*` oder `int[10]` als Ausdrücke auffassen kann – `int` ist ein atomarer Typausdruck, `int*` die Anwendung des Postfixoperators `*` auf den Teilausdruck `int` und `int[10]` eine Anwendung der Operatorkombination `...[...]` auf die Operanden `int` und `10` –, lässt sich auch die Syntax von Typen beliebig erweitern.
4. Schließlich können auch Deklarationen wie z. B. `x : int` als Ausdrücke interpretiert werden (der linke Operand des Doppelpunkt-Operators ist ein Name, der rechte ein Typ), sodass auch deren Syntax prinzipiell beliebig erweiterbar ist.

Im Vortrag werden die oben skizzierten Ideen genauer erläutert und auf die Implementierung eines entsprechenden Parsers eingegangen.