Einblicke in das MOSTflexiPL-Typsystem

Christian Heinlein
Studiengang Informatik
Hochschule Aalen – Technik und Wirtschaft

Neben der syntaktischen Erweiterbarkeit, die das herausragende „Markenzeichen“ der Programmiersprache MOSTflexiPL (Modular, Statically Typed, Flexibly Extensible Programming Language, http://flexipl.info) darstellt, bietet die Sprache auch ein ausdrucksstarkes und interessantes Typsystem, das im Rahmen des Vortrags etwas näher beleuchtet werden soll.

Neben einigen elementaren vordefinierten Typen, wie z. B. `int` (beliebig große ganze Zahlen) und `char` (Unicode-Zeichen), bietet die Sprache vordefinierte Typkonstruktoren `•*` und `•?`, d. h. Postfix-Operatoren `*` und `?`, die zu einem beliebigen Typ `T` jeweils einen neuen Typ `T*` (Sequenzen mit Elementtyp `T`) bzw. `T?` (Variablen mit Inhaltstyp `T`) liefern.

Elementare benutzerdefinierte Typen wie zum Beispiel `Person` können durch Deklarationen der Art `"Person" : type` als Konstanten des ebenfalls vordefinierten Metatyps `type` definiert werden. (Damit die zugehörigen Werte bzw. Objekte „tatsächlich“ Personen repräsentieren, müssen dann noch geeignete Attribute wie z. B. `name` und `date of birth` definiert werden.)

Benutzerdefinierte Typkonstruktoren wie z. B. `•x•`, d. h. ein Infixoperator `x` zur Repräsentation von Paaren wie z. B. `int x bool`, können einfach als typwertige Operatoren definiert werden (wobei auch hier wieder geeignete Attribute ergänzt werden müssen):

```
["A" : type; "B" : type] A "x" B : type
```

Da die Parameter solcher Typkonstruktoren nicht auf Typen beschränkt sind, erhält man ganz nebenbei *abhängige Typen* (dependent types), zum Beispiel:

```
["T" : type; "M" : int; "N" : int] T "{" M "x" N "}" : type
```

Eine Anwendung dieses Typkonstruktors wie z. B. `int{3x4}` soll den Typ aller 3×4-Matrizen mit Elementen des Typs `int` repräsentieren (was wiederum durch geeignete Attribute „materialisiert“ werden muss).

Mit Hilfe *deduzierter Parameter* (vergleichbar mit Typparametern in Java und Template-Parametern in C++) lassen sich dann leicht generische Operatoren zur Verarbeitung entsprechender Objekte definieren. Beispielsweise definiert der folgende Infix-Operator `•*•` die Multiplikation einer `LxM`-Matrix `A` mit einer `MxN`-Matrix `B`, die als Ergebnis eine `LxN`-Matrix liefert, wobei der Elementtyp jeweils `T` ist:

```
[
    "T" : type;
    "L" : int; "M" : int; "N" : int;
    "A" : T{LxM};
    "B" : T{MxN}
]
A "*" B : T{LxN}
{ ... (Implementierung der Multiplikation) ... }
```

Bei einer konkreten Anwendung des Operators werden die Belegungen der deduzierten Parameter `T`, `L`, `M` und `N` automatisch aus den Typen der expliziten Parameter `A` und `B` ermittelt („deduziert“).

Mit Hilfe *impliziter Parameter* lassen sich schließlich auch Einschränkungen für deduzierte Parameter definieren, beispielsweise dass für den Parameter `T` des vorigen Beispiels nur Typen eingesetzt werden dürfen, für die es einen Additionsoperator `•+•` und einen Multiplikationsoperator `•*•` gibt.