OS FKPS: Dependent Types

Johannes Waldmann, HTWK Leipzig

17. Mai 2018

Im Vortrag geht es um grundsätzliche Konzepte für Dependent Types [ML72]. Sie werden durch Beispiele in der Sprache Agda [Nor07, unk18] illustriert, die aus [BD08, Mal13] übernommen sind.

1 Definition, Beispiele, Motivation

Ein *Typ* ist eine Menge von Werten. Beispiele sind der Typ `Nat` der natürlichen Zahlen, der Typ `List Nat` der Listen mit Elementen vom typ `Nat`.

Ein *(daten-)abhängiger Typ* (dependent type, DT) ist ein Typ, der von Elementen anderer Typen abhängt. Ein Beispiel ist der Typ `Vector 5 Nat` der Listen der Länge 5 mit Schlüsseln vom Typ `Nat`.

Man verwendet DT, um Eigenschaften von Programmen zu spezifizieren und die Einhaltung dieser Eigenschaften exakt und soweit möglich automatisiert zu überprüfen. Beispiel: die Konstruktoren und Zugriffsfunktion haben präzise Typen:

```
data Vector (A : Set) : Nat -> Set where
  nil : Vector A zero
  cons : (n : Nat) -> A -> Vector A n -> Vector A (succ n)
head : (A : Set) -> (n : Nat) -> Vector (succ n) -> A
head (cons x _) = x
```

Damit ist der in Haskell mögliche Laufzeitfehler `head Nil` statisch ausgeschlossen.

2 Intuitionistische Typtheorie

Man benutzt die Curry-Howard-Isomorphie:

- Typ $\approx$ Aussage (der Typ ist nicht leer)
- Programm $\approx$ Beweis (ein Objekt des Typs wird konstruiert)

Beispiel: das Programm (der Lambda-Ausdruck) $\lambda x.\lambda y.x$ hat den Typ $\forall AB : A \to (B \to A)$. Dabei sind A, B Typvariablen und die Pfeile bezeichnen Funktionenräume. Die Pfeile können jedoch auch als Implikationen gelesen werden und A, B als Aussagenvariablen. Die Aussage $A \to (B \to A)$ ist allgemeingültig. Der Term $K = \lambda x.\lambda y.x$

ist ein *Beweis* dafür, denn er konstriert aus einem Beweis von A (einem Objekt vom Typ A) und einem Beweis von B (einem Objekt vom Typ B) ein Objekt vom Typ A.

3 Aussagen als Typen in Agda

Beispiel: Man definiert die Gleichheitsrelation als (polymorphen) Typ mit nur einem Konstruktor

```
data _==_ { A : Set } : A -> A -> Set where
  refl : (a : A) -> a == a
```

Beispiel: die Aussage $5 = 5$ ist dann der Typ `5 == 5` und ihr Beweis ist der Ausdruck `refl 5`, weil er genau diesen Typ hat.

Beispiel: „die Addition von (Peano-)Zahlen ist kommutativ“ zeigt man dadurch, daß man ein Element des Typs `x -> y -> plus x y == plus y x` angibt, d.h. eine Funktion mit zwei Argumenten, die einen Beweis für die Gleichheit konstruiert.

Dieses Vorgehen ist korrekt, weil alle Funktionen in Agda terminieren. Das wird statisch überprüft.

4 Agda ausprobieren

`cabal install Agda`, dann `agda-mode setup`, dann `emacs M.agda`, dann wie [Mal13].

Literatur

[BD08] Ana Bove and Peter Dybjer. Dependent types at work. In Ana Bove, Luís Soares Barbosa, Alberto Pardo, and Jorge Sousa Pinto, editors, *Language Engineering and Rigorous Software Development, International LerNet ALFA Summer School 2008, Piriapolis, Uruguay, February 24 - March 1, 2008, Revised Tutorial Lectures*, volume 5520 of *Lecture Notes in Computer Science*, pages 57–99. Springer, 2008.

[Mal13] Jan Malakhovski. Brutal [meta]introduction to dependent types in agda. `http://oxij.org/note/BrutalDepTypes/`, 2013.

[ML72] Peer Martin-Löf. An intuitionistic theory of types. Technical report, 1972.

[Nor07] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

[unk18] unknown. Agda. `http://wiki.portal.chalmers.se/agda/`, 2018.