A Type Theory with Partially Defined Functions and Pattern Matching

Yong Luo

University of Kent, UK

18th April 2006

Partially defined functions

A subtitle: Partially Defined Functions Are Harmless.

Partially defined function:

- the domain of the function is not totally covered
- NOT a non-terminated function

Example:

$$pred(succ(x)) = x$$

Motivation

In theory, it is novel.

- In conventional type theories, e.g. Martin-Lof's TT and Luo's UTT, only are total functions are allowed.
- Now, it is not important whether a function is totally defined.

In practice, it is good for implementation.

- Avoid an undecidable problem (total covering is undecidable).
- Allow any pattern, and hence easy to define functions. (explain it by examples)

Meaning

$$\begin{array}{lcl} pred\ [x : Nat] & : & Nat \\ pred(succ(x)) & = & x \end{array}$$

What about $pred(zero)$?

1. $pred(zero)$ is of type Nat.
2. For all $x : Nat$, $x =_{Nat} zero$ or $x = succ(y)$ for some y.
3. At this moment, $pred(sero)$ is a normal form and its value is unknown.
4. Its value can be defined later, or will never be defined.

Implementation (1)

The Ackermann function

$$ack \quad : \quad Nat \rightarrow Nat \rightarrow Nat$$

$$\begin{array}{lcl} ack(zero, y) & = & succ(y) \\ ack(succ(x), zero) & = & ack(x, succ(zero)) \\ ack(succ(x), succ(y)) & = & ack(x, ack(succ(x), y)) \end{array}$$

Nested Patterns

$$np \quad : \quad List(List(Nat)) \rightarrow Nat \rightarrow Nat$$

$$\begin{array}{lcl} np([], n) & = & n \\ np(([] : xss), n) & = & np(xss, n) \\ np(((x : xs) : xss), n) & = & np((xs : xss), succ(n)) \end{array}$$

Implementation (2)

The Quick Sort

$$QS \quad : \quad List(Nat) \longrightarrow List(Nat)$$

$$\begin{array}{lcl} QS([]) & = & [] \\ QS(x : xs) & = & QS(leq(x, xs)) + +[x] + +QS(bt(x, xs)) \end{array}$$

Programs are coded as it is. The internal behaviour such as complexity will not be changed.

- A type theory is often regarded as a programming language.
- Program: functionality and code