CSC2125H
Types and Programming Languages
Simply Typed Lambda Calculus

Ningning Xie
Assistant Professor
Department of Computer Science
University of Toronto

Part I Simply typed lambda calculus

Types

- We will use *types* as a means to classify expressions.
- An important consequence is that we can recognize the representation of Booleans, natural numbers, and other data types and distinguish them from other forms of lambda expressions.
- We also explore how typing interacts with computation.

Simple types

- We need at least functions $\tau ::= \tau_1 \to \tau_2$; but this type might be considered ”empty” since there is no base case, so we add variables α, β, γ, etc.

$$
\begin{array}{llll}
\text{Type variables} & \alpha & & \\
\text{Types} & \tau & ::= & \tau_1 \to \tau_2 \mid \alpha
\end{array}
$$

- We follow the convention that -> is right-associative.

Typing

- For now, we write $e \ : \ \tau$ if expression e has type τ

- For example:

$$
\begin{array}{lcl}
\lambda x.\, x & : & \alpha \to \alpha \\
\lambda x.\, x & : & (\alpha \to \beta) \to (\alpha \to \beta)
\end{array}
$$

Typing

- For example:

$$\begin{array}{lcl} \mathit{true} & = & \lambda x.\, \lambda y.\, x & : & \alpha \to (\alpha \to \alpha) \\ \mathit{false} & = & \lambda x.\, \lambda y.\, y & : & \alpha \to (\alpha \to \alpha) \end{array}$$

- Informally (for now), we can reason

If $\cdot \vdash e : \alpha \to (\alpha \to \alpha)$ and e does not reduce, then $e = \mathit{true} = \lambda x.\, \lambda y.\, x$ or $e = \mathit{false} = \lambda x.\, \lambda y.\, y$.