DD2552 Seminar 3: Simply Typed Lambda Calculus and Beyond

Karl Palmskog

KTH

Wednesday September 6, 2023

Course material

- PFPL chapter 4
- Commentary in Software Foundations
 - https://softwarefoundations.cis.upenn.edu/plfcurrent/Stlc.html
 - https://softwarefoundations.cis.upenn.edu/plfcurrent/StlcProp.html

Why types?

- lack of restrictions in formal systems can lead to contradictions (paradoxes)
- types can enforce enough discipline to rule out contradictions
- "set of all sets" vs. "class of all sets" vs. hierarchy of classes

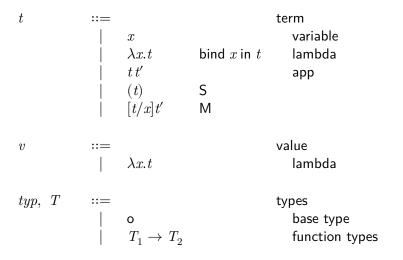
Why Simply Typed Lambda Calculus?

- possibly simplest meaningful typesystem for lambda calculus (add function types)
- showcases why we have (several) types in functional languages
- stepping stone to practical languages like CakeML
- usually abbreviated STLC

What is STLC?

- lambda calculus with a typing relation
- a basis for metatheory (safety and progress)
- benchmark for formal metatheory

Lambda Calculus syntax



Lambda Calculus reduction reminder

$$\overline{(\lambda x.t_{12}) \: v_2 \longrightarrow [v_2/x]t_{12}} \quad \text{RED_AX_APP}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1 \ t \longrightarrow t_1' \ t} \quad \text{RED_CTX_APP_FUN}$$

$$\frac{t_1 \longrightarrow t_1'}{v \, t_1 \longrightarrow v \, t_1'} \quad \text{RED_CTX_APP_ARG}$$

STLC typing relation

$$\frac{x: T \in \Gamma}{\Gamma \vdash x: T} \quad \text{TYPING}_\text{VAR}$$

$$\frac{\Gamma, x: T_1 \vdash t: T_2}{\Gamma \vdash \lambda x.t: T_1 \rightarrow T_2} \quad \text{typing} \text{Abs}$$

$$\begin{array}{c} \Gamma \vdash t_1 : T_1 \rightarrow T_2 \\ \hline \Gamma \vdash t_2 : T_1 \\ \hline \Gamma \vdash t_1 t_2 : T_2 \end{array} \quad \text{TYPING_APP} \end{array}$$

STLC property: progress

Pierce et al.:

"closed, well-typed terms are not stuck: either a well-typed term is a value, or it can take a reduction step."

Theorem

For all terms t and types T, if $\bullet \vdash t : T$, then t is a value or there exists t' such that $t \longrightarrow t'$.

STLC property: preservation

Pierce et al.:

if a closed, well-typed term t has type T and takes a step to t', then t' is also a closed term with type T. In other words, the small-step reduction relation preserves types.

Theorem

For all terms t, t' and types T, if $\bullet \vdash t : T$ and $t \longrightarrow t'$, then $\bullet \vdash t' : T$.

Instantiating STLC with booleans t term ::=variable x $\lambda x.t$ bind x in tlambda t t'app if t then t' else t''conditional true true false false (t)S [t/x]t'Μ value v::= $\lambda x.t$ lambda typ, T::=types Bool $T_1 \rightarrow T_2$ bool type function types

Instantiating STLC with booleans

$$\begin{split} \overline{(\lambda x.t_{12}) v_2 \longrightarrow [v_2/x]t_{12}} & \text{RED_AX_APP} \\ \\ \overline{(\lambda x.t_{12}) v_2 \longrightarrow [v_2/x]t_{12}} & \text{RED_AX_APP_FUN} \\ \\ \overline{t_1 \longrightarrow t_1'} & \text{RED_CTX_APP_FUN} \\ \\ \\ \overline{t_1 \longrightarrow t_1'} & \text{RED_CTX_APP_ARG} \\ \\ \hline \overline{iftrue then t_1 else t_2 \longrightarrow t_1}} & \text{RED_IF_TRUE} \\ \\ \\ \hline \overline{iffalse then t_1 else t_2 \longrightarrow t_2}} & \text{RED_IF_TRUE} \\ \\ \\ \hline \overline{iffalse then t_1 else t_2 \longrightarrow t_2}} & \text{RED_IF_FALSE} \\ \\ \\ \\ \\ \\ \hline \frac{t_1 \longrightarrow t_1'}{if t_1 then t_2 else t_3 \longrightarrow if t_1' then t_2 else t_3}} & \text{RED_IF} \\ \end{split}$$

Instantiating STLC with booleans

 $x: T \in \Gamma$ $\Gamma \vdash r: T$ TYPING_VAR $\frac{\Gamma, x: T_1 \vdash t: T_2}{\Gamma \vdash \lambda x.t: T_1 \rightarrow T_2} \quad \text{TYPING_ABS}$ $\Gamma \vdash t_1 : T_1 \to T_2$ $\frac{\Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1 t_2 : T_2} \quad \text{TYPING_APP}$ TYPING TRUE $\Gamma \vdash \mathsf{true} : \mathsf{Bool}$ TYPING FALSE $\Gamma \vdash \mathsf{false} : \mathsf{Bool}$ $\Gamma \vdash t_1$: Bool $\Gamma \vdash t_2 : T_1$ $\Gamma \vdash t_3 : T_1$ TYPING IF $\Gamma \vdash \mathbf{if} t_1 \mathbf{then} t_2 \mathbf{else} t_3 : T_1$

Towards a more realistic language

- convenient to annotate types in programs
- we also need a datatype definition mechanism so preservation/progress proofs do not need to change with every new kind of data
- examples: OCaml Light, CakeML