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/plf-current/Stlc.html
  - https://softwarefoundations.cis.upenn.edu/plf-current/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

t	$::=$			term
	$\|$	x		variable
	$\|$	$\lambda x.t$	bind x in t	lambda
	$\|$	$t\,t'$		app
	$\|$	(t)	S	
	$\|$	$[t/x]t'$	M	
v	$::=$			value
	$\|$	$\lambda x.t$		lambda
$typ,\ T$	$::=$			types
	$\|$	o		base type
	$\|$	$T_1 \to T_2$		function types

Lambda Calculus reduction reminder

$$\frac{}{(\lambda x.t_{12})\, v_2 \longrightarrow [v_2/x]t_{12}} \quad \text{RED\_\_AX\_\_APP}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t \longrightarrow t_1'\ t} \quad \text{RED\_\_CTX\_\_APP\_\_FUN}$$

$$\frac{t_1 \longrightarrow t_1'}{v\, t_1 \longrightarrow v\, t_1'} \quad \text{RED\_\_CTX\_\_APP\_\_ARG}$$

STLC typing relation

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \textsc{typing\_var}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash \lambda x.t : T_1 \to T_2} \quad \textsc{typing\_abs}$$

$$\frac{\begin{array}{l}\Gamma \vdash t_1 : T_1 \to T_2 \\ \Gamma \vdash t_2 : T_1\end{array}}{\Gamma \vdash t_1\, t_2 : T_2} \quad \textsc{typing\_app}$$

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
	$\|$	x		variable
	$\|$	$\lambda x.t$	bind x in t	lambda
	$\|$	$t\,t'$		app
	$\|$	$\textbf{if}\,t\,\textbf{then}\,t'\,\textbf{else}\,t''$		conditional
	$\|$	true		true
	$\|$	false		false
	$\|$	(t)	S	
	$\|$	$[t/x]t'$	M	
v	$::=$			value
	$\|$	$\lambda x.t$		lambda
$typ,\ T$	$::=$			types
	$\|$	Bool		bool type
	$\|$	$T_1 \to T_2$		function types

Instantiating STLC with booleans

$$\frac{}{(\lambda x.t_{12})\, v_2 \longrightarrow [v_2/x]t_{12}} \quad \text{RED\_\_AX\_\_APP}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t \longrightarrow t_1'\ t} \quad \text{RED\_\_CTX\_\_APP\_\_FUN}$$

$$\frac{t_1 \longrightarrow t_1'}{v\, t_1 \longrightarrow v\, t_1'} \quad \text{RED\_\_CTX\_\_APP\_\_ARG}$$

$$\frac{}{\textbf{if}\, \mathsf{true}\, \textbf{then}\ t_1\ \textbf{else}\ t_2 \longrightarrow t_1} \quad \text{RED\_\_IF\_\_TRUE}$$

$$\frac{}{\textbf{if}\, \mathsf{false}\, \textbf{then}\ t_1\ \textbf{else}\ t_2 \longrightarrow t_2} \quad \text{RED\_\_IF\_\_FALSE}$$

$$\frac{t_1 \longrightarrow t_1'}{\textbf{if}\ t_1\ \textbf{then}\ t_2\ \textbf{else}\ t_3 \longrightarrow \textbf{if}\ t_1'\ \textbf{then}\ t_2\ \textbf{else}\ t_3} \quad \text{RED\_\_IF}$$

Instantiating STLC with booleans

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

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

$$\frac{\begin{array}{l}\Gamma \vdash t_1 : T_1 \rightarrow T_2 \\ \Gamma \vdash t_2 : T_1\end{array}}{\Gamma \vdash t_1\ t_2 : T_2} \quad \text{TYPING\_APP}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}} \quad \text{TYPING\_TRUE}$$

$$\frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}} \quad \text{TYPING\_FALSE}$$

$$\frac{\begin{array}{l}\Gamma \vdash t_1 : \mathsf{Bool} \\ \Gamma \vdash t_2 : T_1 \\ \Gamma \vdash t_3 : T_1\end{array}}{\Gamma \vdash \mathbf{if}\ t_1\ \mathbf{then}\ t_2\ \mathbf{else}\ t_3 : T_1} \quad \text{TYPING\_IF}$$

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