DD2552 Seminar 3: Simply Typed Lambda Calculus and Beyond

Karl Palmskog

KTH

Wednesday September 6, 2023

KTH Research Preparation course

Module D: PL Semantics

Pablo Buiras

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

λ-calculus with evaluation order

CBV

Call-by-value (CBV)
λ-calculus

(applicative order)

Syntax Inductive set

$x, y, z, \ldots$ variables

$M, N ::= x$ terms
$\quad\quad | \; M\ N$
$\quad\quad | \; \lambda x.M$

$v ::= \lambda x.M$ values

Operational semantics Inductive relation

$$\frac{}{(\lambda x.M)v \rightarrow_\beta M[v/x]}\ \beta$$

$$\frac{M \rightarrow_\beta M'}{M\ N \rightarrow_\beta M'\ N}$$

$$\frac{N \rightarrow_\beta N'}{v\ N \rightarrow_\beta v\ N'}$$

Properties Proofs

Church-Rosser (confluence) Equational theory
Turing completeness Fixed-point existence

Extensions

Natural numbers & addition

x, y, z, ...	variables
M, N ::= x	terms
| M N	
| λx.M	
v ::= λx.M	values

x, y, z, ...	variables
n, m, ...	literals
M, N ::= x	terms
| n	
| add(M,N)	
| M N	
| λx.M	
v ::= λx.M	values
| n	

Extensions – δ rules

Natural numbers & addition

Literal addition

$$\frac{r = m+n}{add(m,n) \rightarrow r}$$

Context closure

$$\frac{M \rightarrow M'}{add(M,N) \rightarrow add(M',N)} \qquad \frac{N \rightarrow N'}{add(v,N) \rightarrow add(v,N')}$$

Extensions – δ rules

Booleans

Syntax

x, y, z, ... variables

M, N, R ::= x terms
| true | false
| if M then N else R
| M N
| λx:τ.M

v ::= λx:τ.M values
| true | false

Operational semantics

$$\frac{}{(\lambda x.M)v \rightarrow M[v/x]}\ \beta$$

$$\frac{}{\text{if true then N else R} \rightarrow \text{N}}\ \text{If-true}$$

$$\frac{}{\text{if false then N else R} \rightarrow \text{R}}\ \text{If-false}$$

+ context closure rules

Programs can go wrong

- Reduction not always leads to a value
  - Nontermination
  - Non-value normal forms often make no sense
- Problem gets worse as we keep adding extensions
  - add(1,true) → ?
  - if (λx.x) then λy.true else false → ?

Type systems

- **Idea**: classify terms by a set of syntax-based rules
- Terms that ‘make sense’ are assigned a **type** (well-typed)
- Terms with the same type have related semantics/behaviour
- Terms that do not have a type are not supposed to be evaluated
- Formally, we have a set of types T, a set of environments Env = Var → T, and a typing judgment relation

$$\_ \vdash \_ : \_ \subseteq Env \times \Lambda \times T$$

$$\Gamma \vdash 1 : nat$$
$$\Gamma \vdash true : bool$$
$$\Gamma \vdash add(1,2) : num$$

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

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

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

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\ t_2 : T_2} \quad \textsc{typing\_app}$$

Instantiating STLC with booleans

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

Instantiating STLC with booleans

$$\frac{}{(\lambda x.t_{12})\, v_2 \longrightarrow [v_2/x]t_{12}} \quad \textsc{red\_\_ax\_\_app}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\, t \longrightarrow t_1'\, t} \quad \textsc{red\_\_ctx\_\_app\_\_fun}$$

$$\frac{t_1 \longrightarrow t_1'}{v\, t_1 \longrightarrow v\, t_1'} \quad \textsc{red\_\_ctx\_\_app\_\_arg}$$

$$\frac{}{\textbf{if}\,\mathsf{true}\,\textbf{then}\,t_1\,\textbf{else}\,t_2 \longrightarrow t_1} \quad \textsc{red\_\_if\_\_true}$$

$$\frac{}{\textbf{if}\,\mathsf{false}\,\textbf{then}\,t_1\,\textbf{else}\,t_2 \longrightarrow t_2} \quad \textsc{red\_\_if\_\_false}$$

$$\frac{t_1 \longrightarrow t_1'}{\textbf{if}\,t_1\,\textbf{then}\,t_2\,\textbf{else}\,t_3 \longrightarrow \textbf{if}\,t_1'\,\textbf{then}\,t_2\,\textbf{else}\,t_3} \quad \textsc{red\_\_if}$$

Instantiating STLC with booleans

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

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

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\ t_2 : T_2} \quad \textsc{typing\_app}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}} \quad \textsc{typing\_true}$$

$$\frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}} \quad \textsc{typing\_false}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Bool} \qquad \Gamma \vdash t_2 : T_1 \qquad \Gamma \vdash t_3 : T_1}{\Gamma \vdash \mathbf{if}\ t_1\ \mathbf{then}\ t_2\ \mathbf{else}\ t_3 : T_1} \quad \textsc{typing\_if}$$

Example typing derivations

$$\dfrac{\dfrac{x : \text{bool} \in \Gamma'}{\Gamma' \vdash x : \text{bool}} \quad \dfrac{\dfrac{y : \text{bool} \in \Gamma'}{\Gamma' \vdash y : \text{bool}} \quad \Gamma' \vdash \text{true} : \text{bool} \quad \Gamma' \vdash \text{false} : \text{bool}}{\Gamma' \vdash \text{if } y \text{ then true else false} : \text{bool}} \quad \dfrac{}{\Gamma' \vdash \text{false} : \text{bool}}}{\dfrac{\Gamma, x{:}\text{bool}, y{:}\text{bool} \vdash \text{if } x \text{ then (if } y \text{ then true else false) else false} : \text{bool}}{\dfrac{\Gamma, x{:}\text{bool} \vdash (\lambda y{:}\text{bool}.\ \text{if } x \text{ then (if } y \text{ then true else false) else false}) : \text{bool} \to \text{bool}}{\Gamma \vdash (\lambda x{:}\text{bool}.\ \lambda y{:}\text{bool}.\ \text{if } x \text{ then (if } y \text{ then true else false) else false}) : \text{bool} \to \text{bool} \to \text{bool}}}}$$

Properties

Canonical forms

- Well-typed values are of a specific form that depends on their type.
- **Canonical Forms Lemma:** Assume M is a value.
  - $\Gamma \vdash M : \sigma \to \tau \Rightarrow M = \lambda x{:}\sigma.N$ for some x, N.
  - $\Gamma \vdash M : \text{bool} \Rightarrow M = \text{true}$ or $M = \text{false}$

- **Proof.** Trivial by inspection of the typing rules and definition of values.

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 $t \longrightarrow t'$.

Progress

- If $\vdash M : \sigma$, either M is a value or $\exists N.M \rightarrow N$
- **Proof.** Induction on typing derivation
  - (Var): $\vdash x : \sigma$, not possible
  - (Bool-1, Bool-2, Abs): Trivial
    - $\vdash$ true, false : bool or $\vdash \lambda x{:}\tau.M : \tau \rightarrow \sigma$
  - (App): $\vdash M\ N : \sigma$, and $\vdash M : \tau \rightarrow \sigma$, $\vdash N : \tau$
    - By IH, either M is a value or it can take a step. (cont)

Progress (2)

- Case M is a value: Now by IH then N is either a value or can take a step.
  - Case N is a value: By canonical forms lemma, M is an abstraction, so we have a redex.
  - Case N can take a step: M N can take a step by context closure.
- Case M can take a step: M N can take a step by context closure.

– (If): by a similar case analysis after applying IH.

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 $\bullet \vdash t' : T$.

Preservation

- If $\Gamma \vdash M : \sigma$ and $M \to N$ then $\Gamma \vdash N : \sigma$
- **Proof.** By induction on the derivation of $M \to N$.
  - Case (β): $M = (\lambda x{:}\tau.M_1)\ M_2$, hence

    $N = M_1[M_2/x]$. Result follows by Subst. lemma (omitted).
  - Case context rules: Similar result, typing is closed under contexts (omitted).
  - Case (If-true): M = if true then M_1 else M_2, $N = M_1$.

    From $\Gamma \vdash M : \sigma$ we know $\Gamma \vdash M_1 : \sigma$.
  - Case (If-false): analogous

Properties

Type system Soundness

- Soundness = progress + preservation
- Progress:
  - If $\vdash M : \sigma$, either M is a value or $\exists N.M \rightarrow N$
- Preservation:
  - If $\Gamma \vdash M : \sigma$ and $M \rightarrow N$ then $\Gamma \vdash N : \sigma$

Properties

- Normal forms and nontermination
  - Ω is not well-typed
  - (Weak) Normalisation: All well-typed terms have a normal form
- Fixed points?
  - Y is not well-typed

Fixed points

M, N, R ::= x terms
| M N
| true | false
| if M then N else R
| λx:τ.M
| fix M

$$\frac{\Gamma \vdash M : \tau \to \tau}{\Gamma \vdash \text{fix } M : \tau}\ \text{ty-fix}$$

$$\frac{}{\text{fix } M \to M\ (\text{fix } M)}\ \text{fix}$$

Russell's Paradox

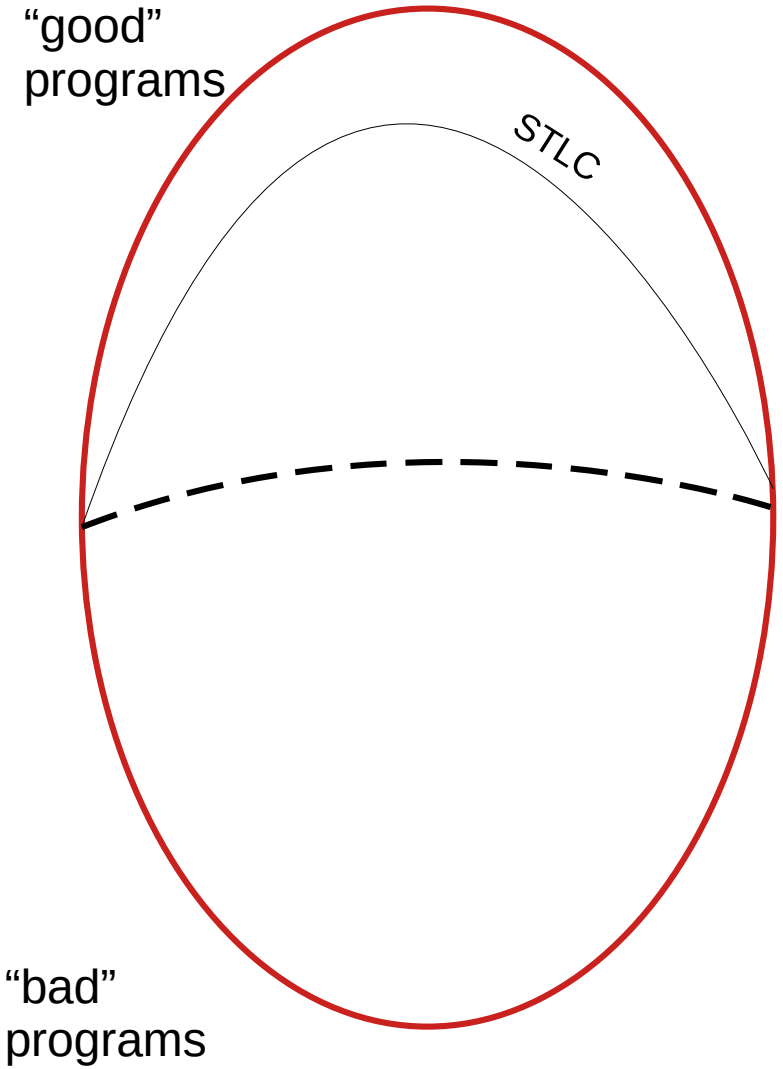
- Y can be seen as an instance of Russell's paradox
- Consider sets represented by characteristic functions
  - $g = \lambda x{:}bool.\ \neg(x\ x)$

    "the set of all sets that don't contain themselves"
  - $g\ g$?
    - $g\ g =_\beta \neg(g\ g)$

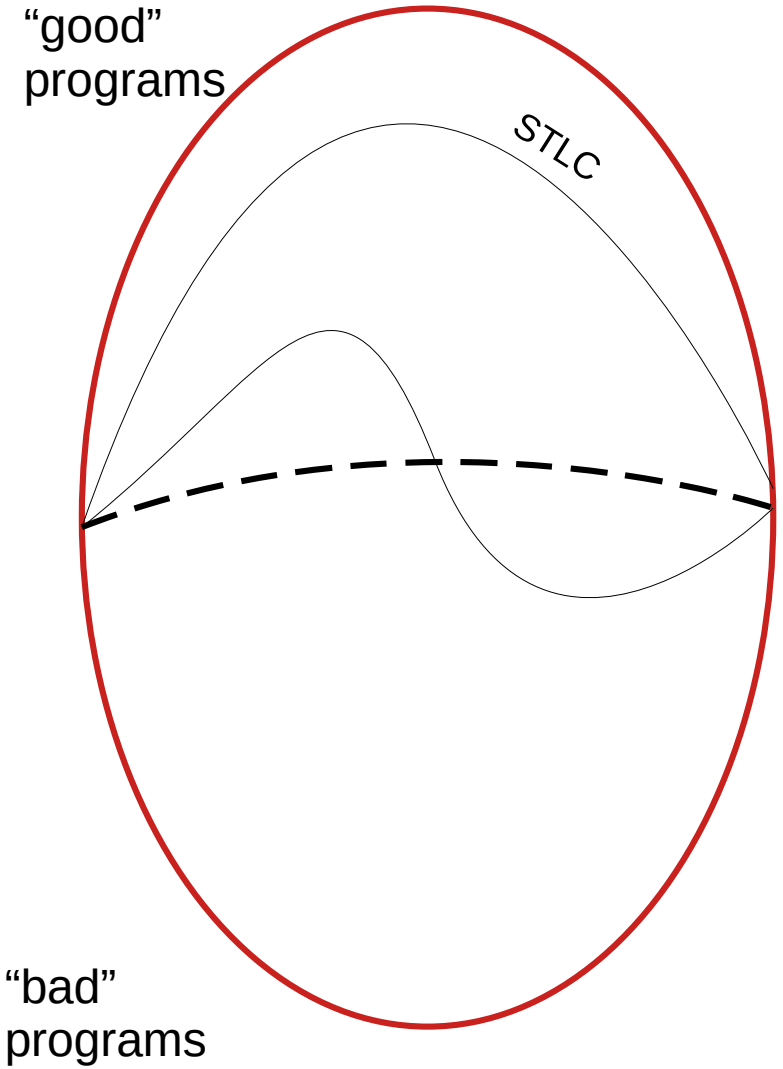
Typing Ω fails

$$\dfrac{\dfrac{\dfrac{?}{\Gamma, x{:}\mathrm{bool} \vdash x : \tau \to \sigma} \quad \Gamma, x{:}\mathrm{bool} \vdash x : \tau}{\Gamma, x{:}\mathrm{bool} \vdash x\ x : \sigma}}{\Gamma \vdash (\lambda x{:}\mathrm{bool}.x\ x) : \mathrm{bool} \to \sigma} \qquad \Gamma \vdash (\lambda x{:}\mathrm{bool}.x\ x) : \mathrm{bool}}{\Gamma \vdash (\lambda x{:}\mathrm{bool}.x\ x)\ (\lambda x{:}\mathrm{bool}.x\ x) : \sigma}$$

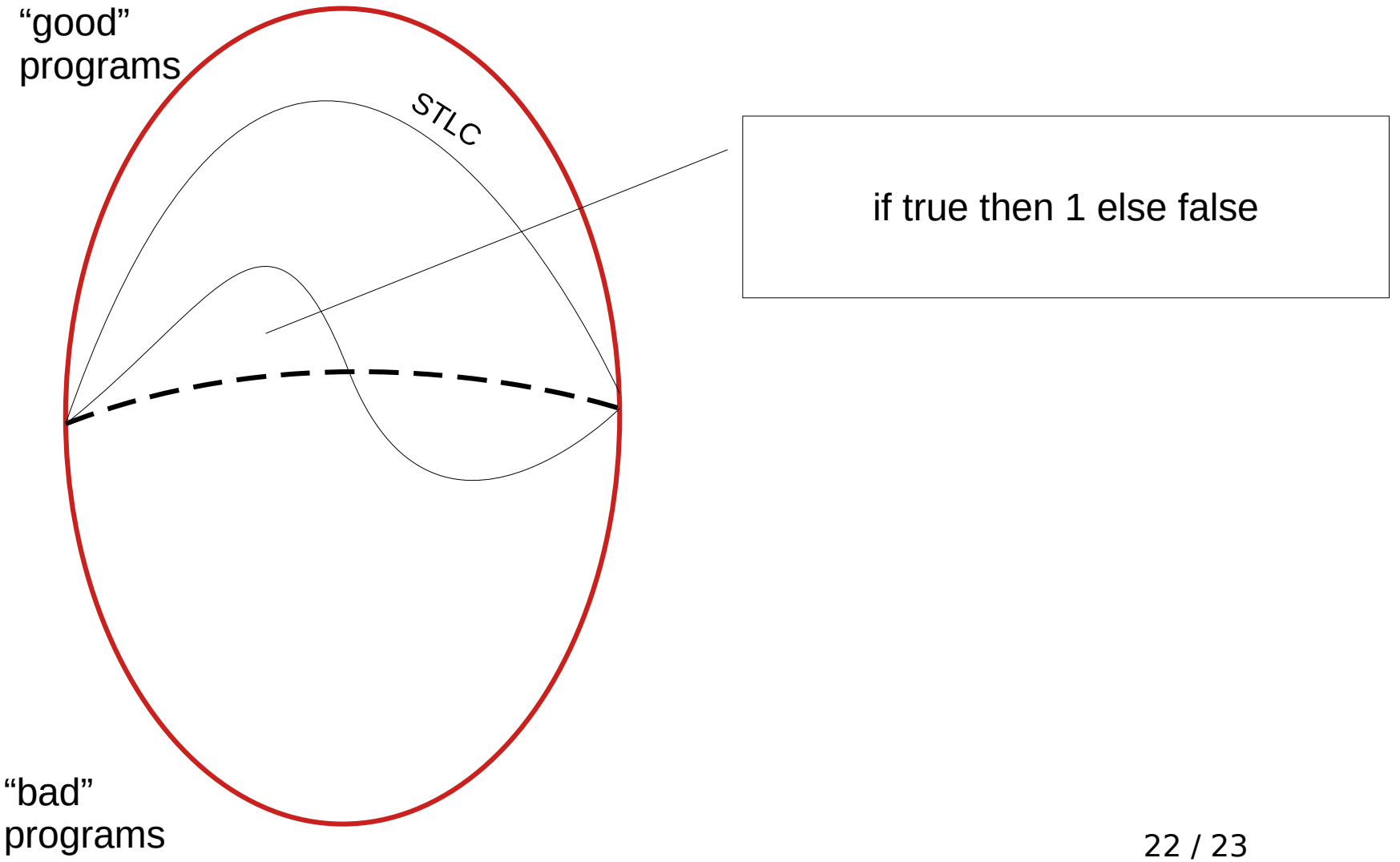
Well-typed terms



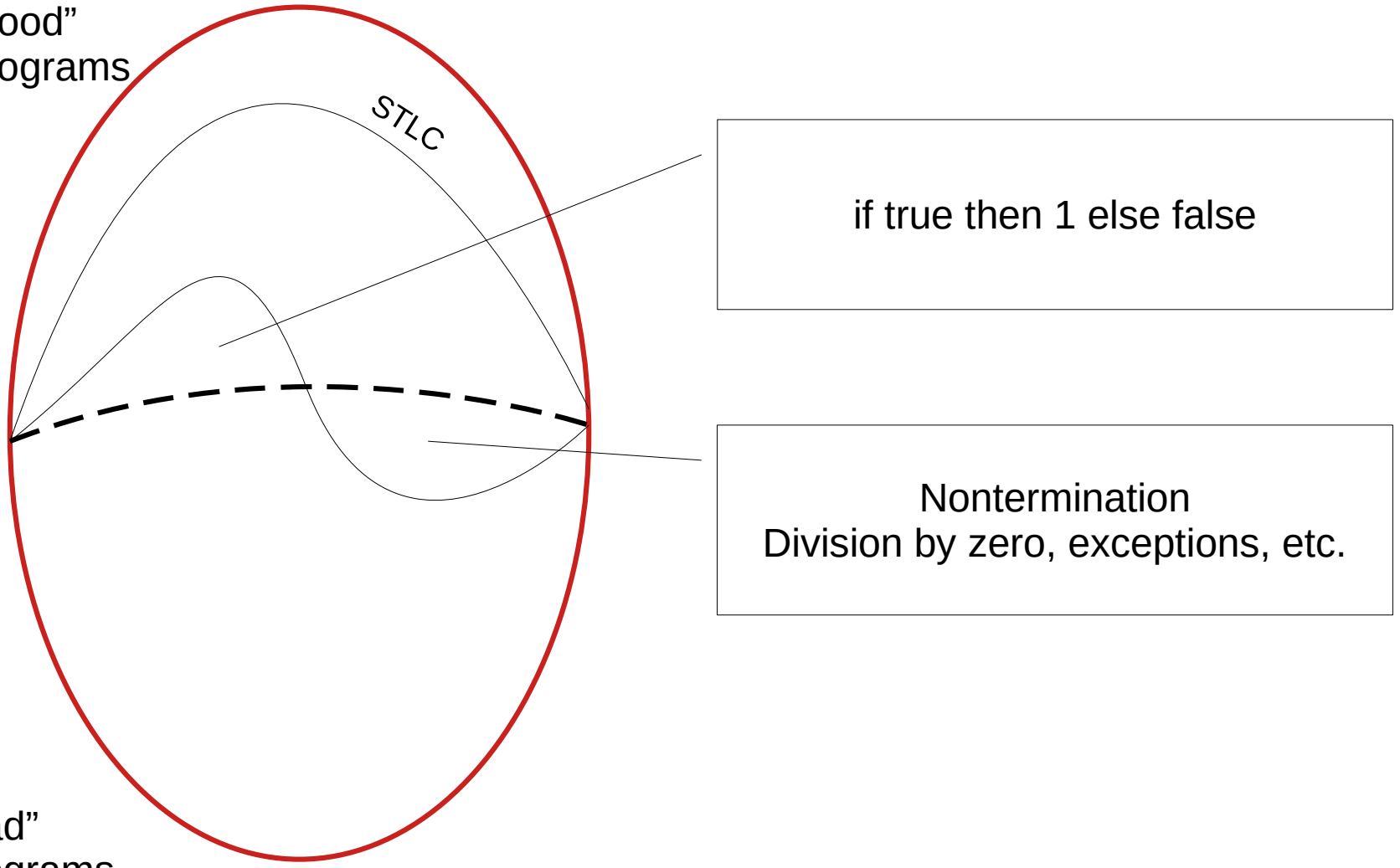
Well-typed terms



Well-typed terms



Well-typed terms



Summary

- Lambda calculus extensions
- Simply Typed Lambda Calculus
  - Statically avoid nonsensical terms
  - Metatheory:

    CR Type soundness (Progress+Preservation)

    Strong normalisation

    Lost: fixed point theorem (thus gen. recursion)

Towards more realistic typed languages

- 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
- separate typing relations for:
  - patterns in pattern matching
  - expressions
  - definitions (of functions, etc.)
  - programs

Example Language: OCaml Light (2008)

OCaml Light semantics is a core ML, excluding only modules and objects

- variant data types, `type t = I of int | C of char`
- record types, `type t = { f : int ; g : bool }`
- parametric type constructors, `type 0 a t = C of 0 a`
- recursive and mutually recursive combinations of the above
- exceptions
- values

Example Language: CakeML (2012-)

- a substantial subset of Standard ML
- modules and signatures, but no functors
- bootstrapped compiler with type checking

Properties of the CakeML Language

- impure language in the SML family
- eagerly evaluated
- semantics given in functional big-step style
- supports IO and FFI
- all integers are unbounded

Syntax of CakeML vs. Standard ML

- CakeML has curried Haskell-style constructor syntax
- constructors in CakeML must begin with an uppercase letter
- constructors must be fully applied
- alpha-numeric variable and function names begin with a lowercase letter
- CakeML lacks SML's records, functors, open and (at present) signatures
- CakeML capitalises `True`, False and `Ref`

Semantics of CakeML vs. Standard ML

- right-to-left evaluation order
- CakeML has no equality types
- semantics of equality is different from SML and OCaml
- multi-argument functions

Example CakeML Programs

Hello World:

```
print "Hello world!\n";
```

Fibonacci with argument from CLI:

```
fun fiba i j n = if n = 0 then i else fiba j (i+j) (n-1);
let
  val v = Option.valOf (Int.fromString
            (List.hd (CommandLine.arguments())))
in
  TextIO.print ((Int.toString (fiba 0 1 v)) ^ "\n")
end
handle _ => TextIO.print_err ("usage: "
              ^ (CommandLine.name()) ^ " <n>\n");
```

CakeML List Functions

```
fun foldl f e xs =
  case xs of [] => e
  | (x::xs) => foldl f (f e x) xs;

fun reverse xs =
  let
    fun append xs ys =
      case xs of [] => ys
      | (x::xs) => x :: append xs ys;
    fun rev xs =
      case xs of [] => xs
      | (x::xs) => append (rev xs) [x]
  in
    rev xs
  end;
```

Example Using CakeML as Compiler

Download the pre-compiled CakeML, and put "hello world" program in `hello.cml`:

```
$ make hello.cake
$ ./hello.cake
Hello world!
```

Compiler takes 20+ hours to bootstrap in HOL4!

Verifying Programs Using CakeML

- imperative programs handled via monads in HOL4
- proof-producing synthesis in HOL4 via **translator**
- post-hoc verification using separation logic

See CakeML journal paper for overview:
`https://cakeml.org/jfp19.pdf`