

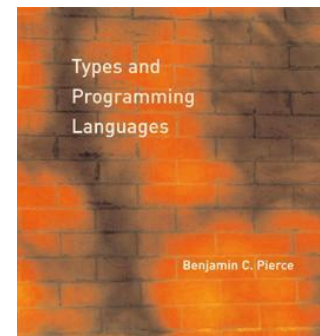
Typed Lambda Calculus



UNIVERSIDADE DA CORUÑA

GRAO EN ENXEÑERÍA INFORMÁTICA **DESEÑO DAS LINGUAXES DE PROGRAMACIÓN**

Based on Chapters 8 and 9 of:
Benjamin C. Pierce, Types and Programming
Languages. MIT Press, 2002.



Outline

- Types for Arithmetic
 - types
 - the typing relation
 - safety = progress + preservation
- The simply typed lambda calculus
 - Function types
 - the typing relation
 - Curry-Howard correspondence
 - Erasure: Curry-style vs Church-style
- Implementation

Terms for arithmetic

Terms

$t ::= \text{true}$
 false
 $\text{if } t \text{ then } t \text{ else } t$
 0
 $\text{succ } t$
 $\text{pred } t$
 $\text{iszero } t$

Values

$v ::= \text{true}$
 false
 nv
 $nv ::= 0$
 $\text{succ } nv$

Boolean and Nat terms

Some terms represent **booleans**, some represent **natural numbers**.

t ::=	true	
	false	
	if t then t else t	if t then t else t
	0	if t then t else t
	succ t	
	pred t	
	iszero t	

Nonsense terms

Some terms don't make sense. They represent neither booleans nor natural numbers.

`succ true`

`iszero false`

`if succ(0) then true else false`

These terms are **stuck** -- no evaluation rules apply, but they are not values.

But what about the following?

`if iszero(0) then true else 0`

Space of terms

Terms

if true then 0 else succ(0)

succ(0)

true

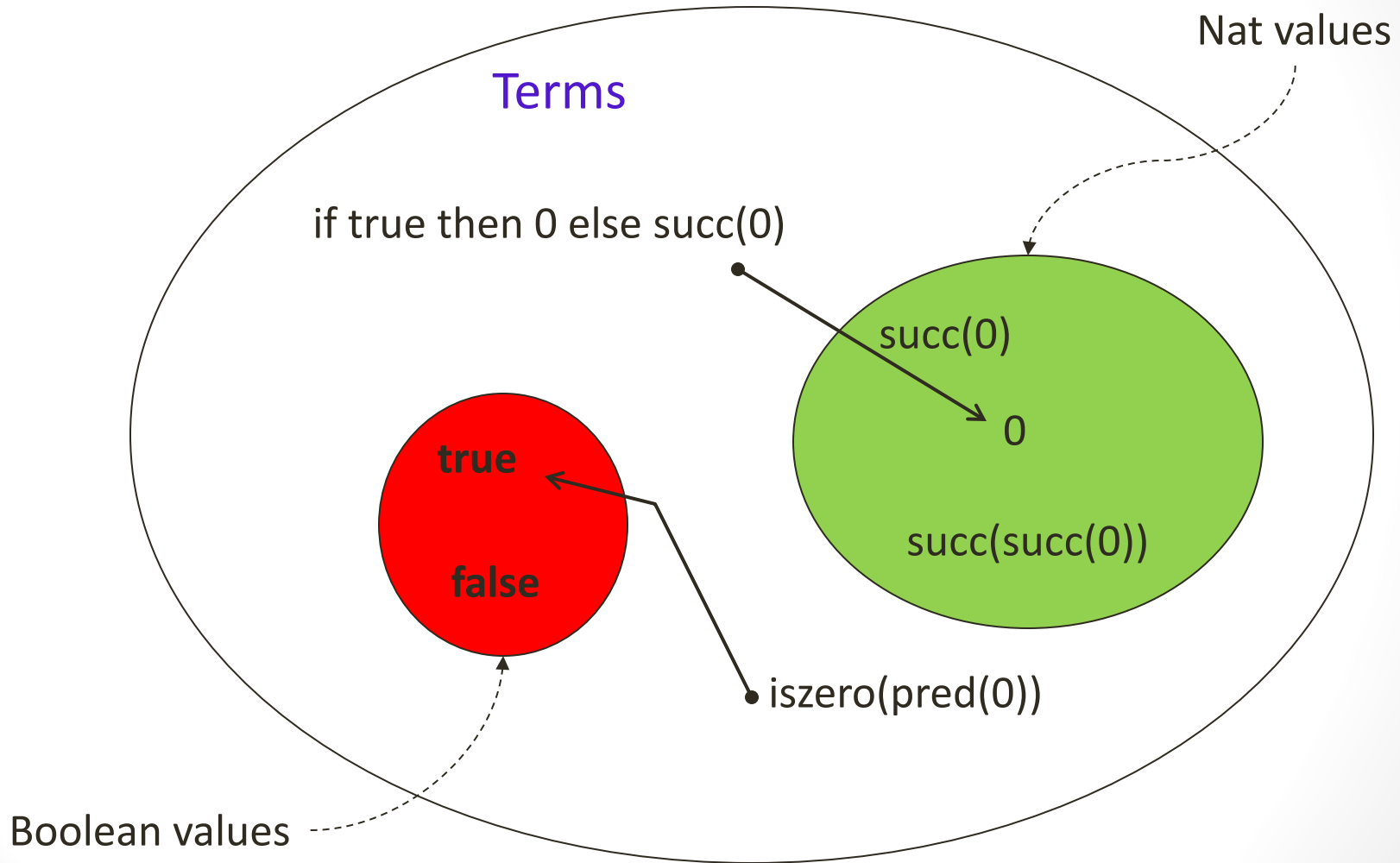
0

false

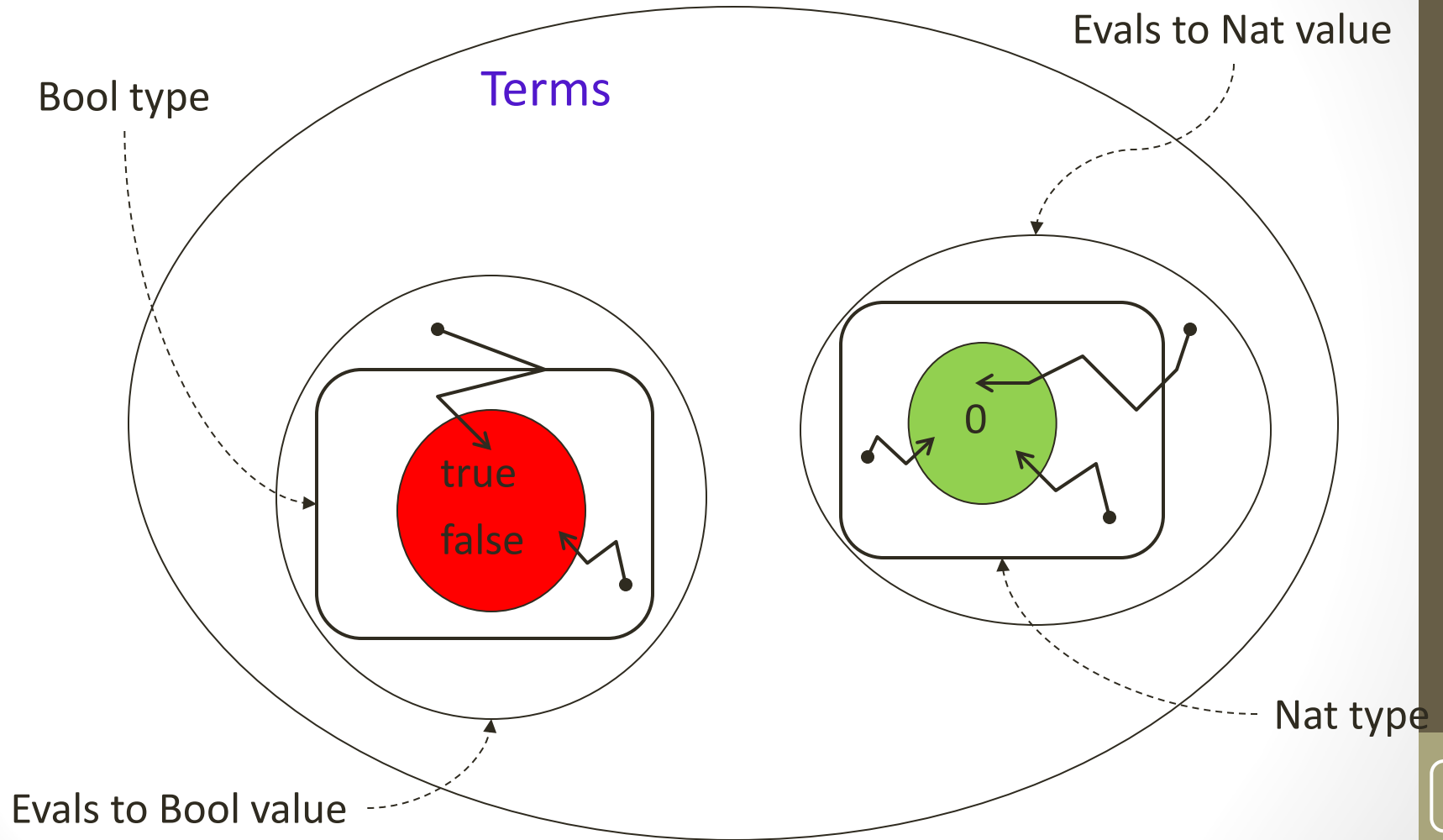
succ(succ(0))

iszero(pred(0))

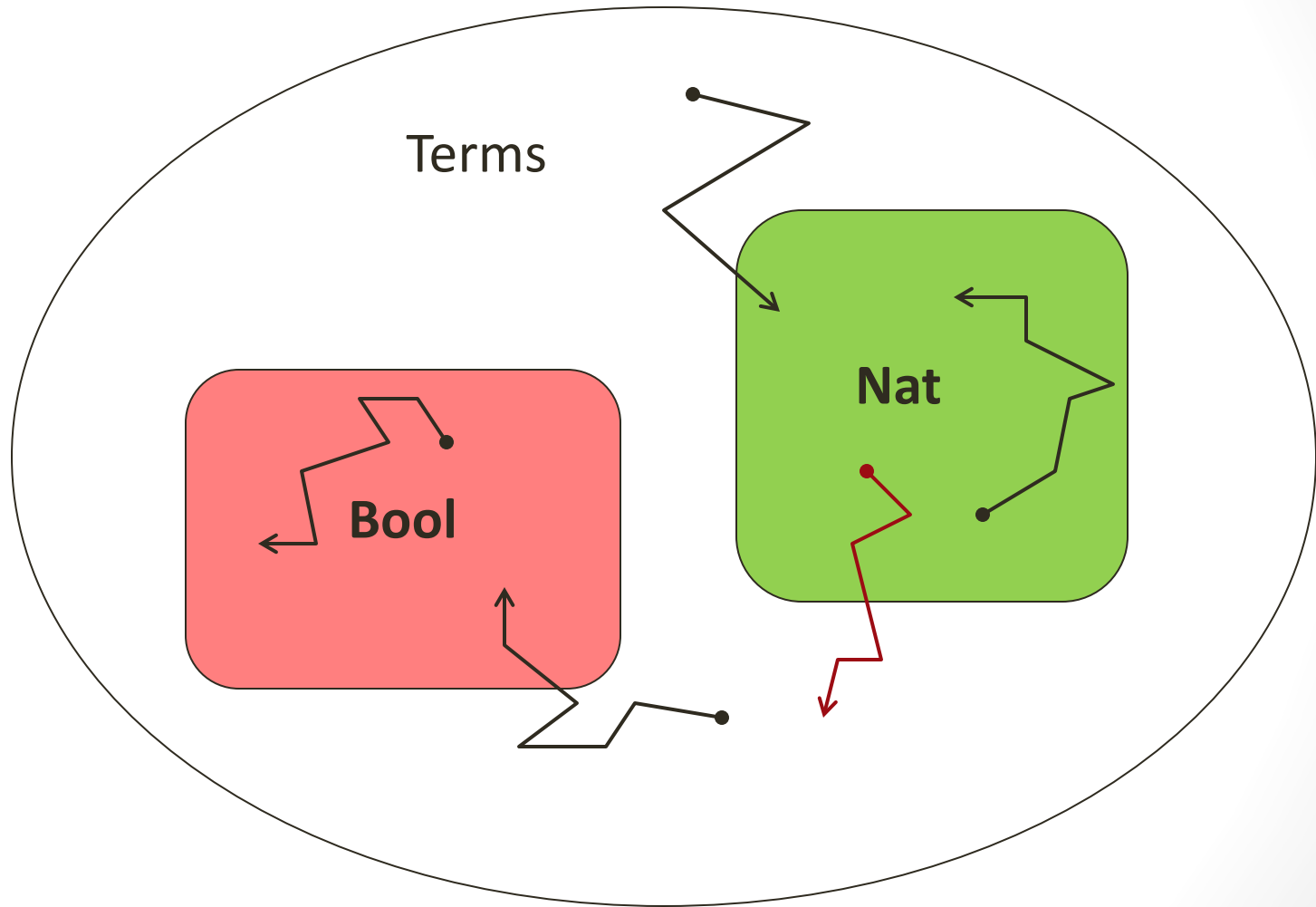
Bool and Nat values



Bool and Nat types



Evaluation preserves type



A Type System

1. type expressions: $T ::= \dots$
2. typing relation : $t : T$
3. typing rules giving an inductive definition of $t : T$

Typing rules for Arithmetic:

$T ::= \text{Bool} \mid \text{Nat}$ (type expressions)

$\text{true} : \text{Bool}$ (T-TRUE)

$\text{false} : \text{Bool}$ (T-FALSE)

$0 : \text{Nat}$ (T-ZERO)

$$\frac{t_1 : \text{Nat}}{\text{succ } t_1 : \text{Nat}} \quad (\text{T-SUCC})$$
$$\frac{t_1 : \text{Nat}}{\text{pred } t_1 : \text{Nat}} \quad (\text{T-PRED})$$
$$\frac{t_1 : \text{Nat}}{\text{iszero } t_1 : \text{Bool}} \quad (\text{T-ISZERO})$$
$$\frac{t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (\text{T-IF})$$

Typing relation

Defn: The typing relation $t : T$ for arithmetic expressions is the smallest binary relation between terms and types satisfying the given rules.

A term t is **typable** (or **well typed**) if there is some T such that $t : T$.

Inversion Lemma

Lemma (8.2.2). [Inversion of the typing relation]

1. If $\text{true} : R$ then $R = \text{Bool}$
2. If $\text{false} : R$ then $R = \text{Bool}$
3. If $\text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$ then $t_1 : \text{Bool}$ and $t_2, t_3 : R$
4. If $0 : R$ then $R = \text{Nat}$
5. If $\text{succ } t_1 : R$ then $R = \text{Nat}$ and $t_1 : \text{Nat}$
6. If $\text{pred } t_1 : R$ then $R = \text{Nat}$ and $t_1 : \text{Nat}$
7. If $\text{iszero } t_1 : R$ then $R = \text{Bool}$ and $t_1 : \text{Nat}$

Typing Derivations

A type derivation is a tree of instances of typing rules with the desired typing as the root.

$$\frac{\frac{\frac{}{0 : \text{Nat}} \text{ (T-ZERO)}}{\text{iszero}(0) : \text{Bool}} \text{ (T-ISZERO)} \quad \frac{\frac{}{0 : \text{Nat}} \text{ (T-ZERO)}}{\text{pred}(0) : \text{Nat}} \text{ (T-PRED)}}{\text{if iszero}(0) \text{ then } 0 \text{ else pred } 0 : \text{Nat}} \text{ (T-IF)}$$

The shape of the derivation tree exactly matches the shape of the term being typed.

Uniqueness of types

Theorem (8.2.4). Each term t has at most one type. That is, if t is typable, then its type is unique, and there is a unique derivation of its type.

Safety (or Soundness)

Safety = Progress + Preservation

Progress: A well-typed term is not stuck -- either it is a value, or it can take a step according to the evaluation rules.

Preservation: If a well-typed term makes a step of evaluation, the resulting term is also well-typed.

Preservation is also known as “subject reduction”

Cannonical forms

Defn: a canonical form is a well-typed value term.

Lemma (8.3.1).

1. If v is a value of type Bool , then v is true or v is false.
2. If v is a value of type Nat , then v is a numeric value, i.e. a term in nv , where
$$\text{nv} ::= 0 \mid \text{succ } \text{nv}.$$

Progress and Preservation for Arithmetic

Theorem (8.3.2) [Progress]

If t is a well-typed term (that is, $t : T$ for some type T), then either t is a value or else $t \rightarrow t'$ for some t' .

Theorem (8.3.3) [Preservation]

If $t : T$ and $t \rightarrow t'$ then $t' : T$.

Proofs are by induction on the derivation of $t : T$.

Simply typed lambda calculus

To type terms of the lambda calculus, we need types for functions (lambda terms):

$$T_1 \rightarrow T_2$$

A function type $T_1 \rightarrow T_2$ specifies the argument type T_1 and the result type T_2 of the function.

Simply typed lambda calculus

The abstract syntax of type terms is

$$\begin{aligned} T &::= \text{base types} \\ &\quad T \rightarrow T \end{aligned}$$

We need base types (e.g Bool) because otherwise we could build no type terms.

We also need terms of these base types, so we have an “applied” lambda calculus. In this case, we will take Bool as the sole base type and add corresponding Boolean terms.

Abstract syntax and values

Terms

$t ::= \text{true}$
 false
 $\text{if } t \text{ then } t \text{ else } t$
 x
 $\lambda x:T. t$
 $t \ t$

Values

$v ::= \text{true}$
 false
 $\lambda x:T. t$

Note that terms contain types! Lambda expressions are *explicitly* typed.

Typing rule for lambda terms

$$\frac{\Gamma, x: T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x: T_1. t_2 : T_1 \rightarrow T_2} \quad (\text{T-Abs})$$

The body of a lambda term (usually) contains free variable occurrences. We need to supply a context (Γ) that gives types for the free variables.

Defn. A **typing context** Γ is a list of free variables with their types. A variable can appear only once in a context.

$$\Gamma ::= \emptyset \mid \Gamma, x: T$$

Typing rule for applications

$$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 t_2 : T_{12}} \quad (\text{T-APP})$$

The type of the argument term must agree with the argument type of the function term.

Typing rule for variables

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad (\text{T-VAR})$$

The type of a variable is taken from the supplied context.

Inversion of typing relation

Lemma (9.3.1). [Inversion of the typing relation]

1. If $\Gamma \vdash x : R$ then $x : R \in \Gamma$
2. If $\Gamma \vdash \lambda x : T_1. t_2 : R$ then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x : T_1 \vdash t_2 : R_2$
3. If $\Gamma \vdash t_1 t_2 : R$, then there is a T_{11} such that $\Gamma \vdash t_1 : T_{11} \rightarrow R$ and $\Gamma \vdash t_2 : T_{11}$
4. If $\Gamma \vdash \text{true} : R$ then $R = \text{Bool}$
5. If $\Gamma \vdash \text{false} : R$ then $R = \text{Bool}$
6. If $\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : R$ then $\Gamma \vdash t_1 : \text{Bool}$ and $\Gamma \vdash t_2, t_3 : R$

Uniqueness of types

Theorem (9.3.3): In a given typing context Γ containing all the free variables of term t , there is at most one type T such that $\Gamma \vdash t : T$.

Canonical Forms (λ_{\rightarrow})

Lemma (9.3.4):

1. If v is a value of type Bool , then v is either `true` or `false`.
2. If v is a value of type $T_1 \rightarrow T_2$, then $v = \lambda x: T_1. t$.

Progress (λ_{\rightarrow})

Theorem (9.3.5): Suppose t is a closed, well-typed term (so $\vdash t : T$ for some T). Then either t is a value, or $t \rightarrow t'$ for some t' .

Proof: by induction on the derivation of $\vdash t : T$.

Note: if t is not closed, e.g. $f \text{ true}$, then it may be in normal form yet not be a value.

Permutation and Weakening

Lemma (9.3.6)[Permutation]: If $\Gamma \vdash t : T$ and Δ is a permutation of Γ , then $\Delta \vdash t : T$.

Lemma (9.3.7)[Weakening]: If $\Gamma \vdash t : T$ and $x \notin \text{dom}(\Gamma)$, then for any type S , $\Gamma, x : S \vdash t : T$, with a derivation of the same depth.

Proof: by induction on the derivation of $\vdash t : T$.

Substitution Lemma

Lemma (9.3.8) [Preservation of types under substitutions]:

If $\Gamma, x: S \vdash t : T$ and $\Gamma \vdash s: S$, then $\Gamma \vdash [x \Rightarrow s]t: T$.

Proof: induction of the derivation of $\Gamma, x: S \vdash t : T$.

Replace leaf nodes for occurrences of x with copies of the derivation of $\Gamma \vdash s: S$.

Substitution Lemma

Lemma (9.3.8) [Preservation of types under substitutions]:

If $\Gamma, x: S \vdash t : T$ and $\Gamma \vdash s: S$, then $\Gamma \vdash [x \Rightarrow s]t: T$.

Proof: induction of the derivation of $\Gamma, x: S \vdash t : T$.

Replace leaf nodes for occurrences of x with copies of the derivation of $\Gamma \vdash s: S$.

Preservation (λ_{\rightarrow})

Theorem (9.3.9) [Preservation]:

If $\Gamma \vdash t : T$ and $t \rightarrow t'$, then $\Gamma \vdash t' : T$.

Proof: induction of the derivation of $\Gamma \vdash t : T$, similar to the proof for typed arithmetic, but requiring the Substitution Lemma for the beta redex case.

Homework: write a detailed proof of Thm 9.3.9.

Introduction and Elimination rules

λ Introduction

$$\frac{\Gamma, x: T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x: T_1. t_2 : T_1 \rightarrow T_2} \quad (\text{T-Abs})$$

λ Elimination

$$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 t_2 : T_{12}} \quad (\text{T-App})$$

Typing rules often come in intro-elim pairs like this.
Sometimes there are multiple intro or elim rules for a construct.

Erasure

Defn: The erasure of a simply typed term is defined by:

$$\text{erase}(x) = x$$

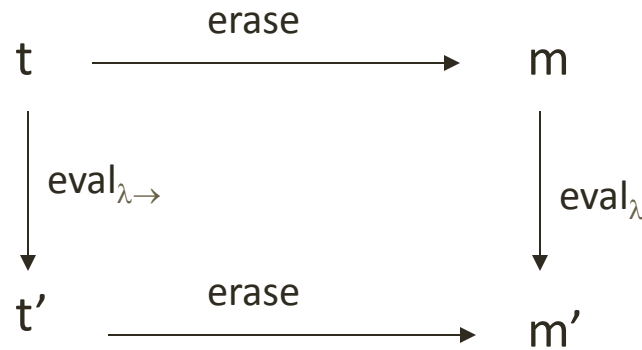
$$\text{erase}(\lambda x: T. t) = \lambda x. \text{erase}(t)$$

$$\text{erase}(t_1 t_2) = (\text{erase}(t_1))(\text{erase}(t_2))$$

erase maps a simply typed term in λ_{\rightarrow} to the corresponding untyped term in λ .

$$\text{erase}(\lambda x: \text{Bool}. \lambda y: \text{Bool} \rightarrow \text{Bool}. y x) = \lambda x. \lambda y. y x$$

Erasure commutes with evaluation



Theorem (9.5.2)

1. if $t \rightarrow t'$ in λ_{\rightarrow} then $\text{erase}(t) \rightarrow \text{erase}(t')$ in λ .
2. if $\text{erase}(t) \rightarrow m$ in λ then there exists t' such that $t \rightarrow t'$ in λ_{\rightarrow} and $\text{erase}(t') = m$.

Curry style and Church style

Curry:

define evaluation for untyped terms, then define the well-typed subset of terms and show that they don't exhibit bad “run-time” behaviors.
Erase and then evaluate.

Church:

define the set of well-typed terms and give evaluation rules only for such well-typed terms.