

Lesson 4 Typed Arithmetic Typed Lambda Calculus

1/28/03
Chapters 8, 9, 10

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

 $nv ::= 0$
 $\text{succ } nv$

1/28/03

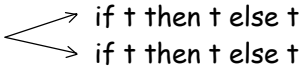
Lesson 4: Typed Arith & Lambda

3

Boolean and Nat terms

Some terms represent booleans, some represent natural numbers.

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



1/28/03

Lesson 4: Typed Arith & Lambda

4

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?

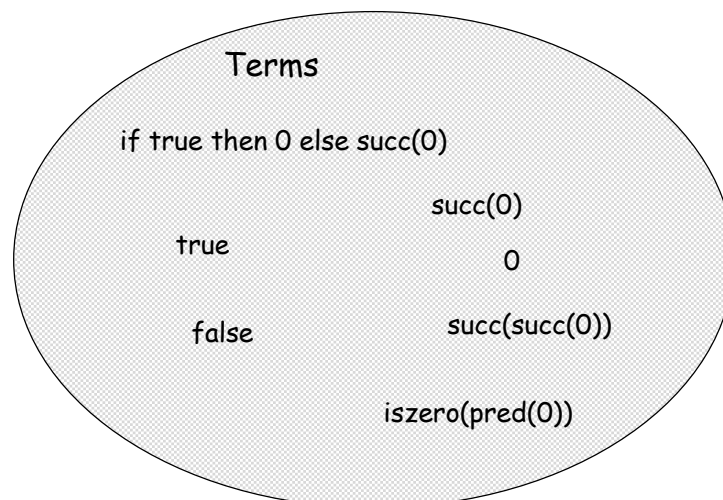
```
if iszero(0) then true else 0
```

1/28/03

Lesson 4: Typed Arith & Lambda

5

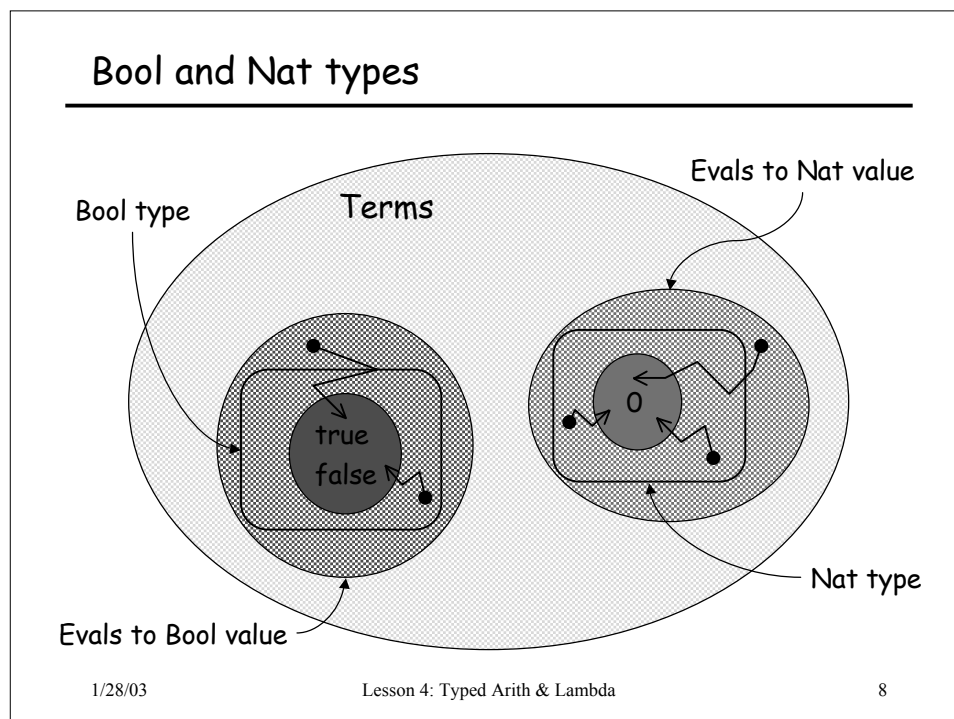
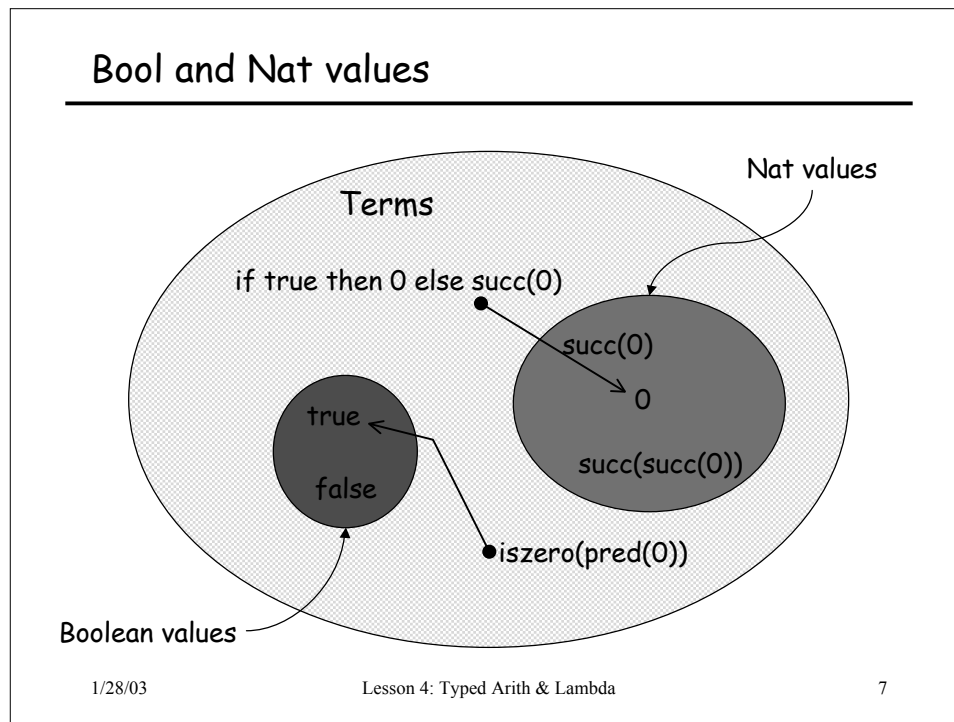
Space of terms



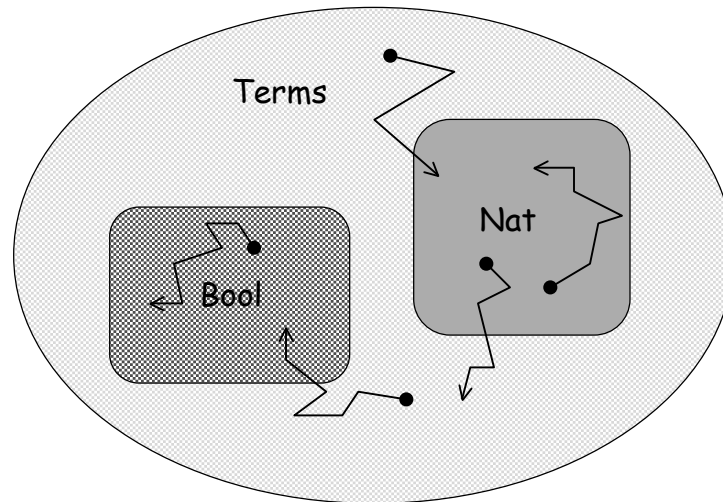
1/28/03

Lesson 4: Typed Arith & Lambda

6



Evaluation preserves type



1/28/03

Lesson 4: Typed Arith & Lambda

9

A Type System

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

1/28/03

Lesson 4: Typed Arith & Lambda

10

Typing rules for Arithmetic: **BN** (typed)

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

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

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

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

$$\frac{t1 : \text{Nat}}{\text{succ } t1 : \text{Nat}} \quad (\text{T-Succ})$$

$$\frac{t1 : \text{Nat}}{\text{pred } t1 : \text{Nat}} \quad (\text{T-Pred})$$

$$\frac{t1 : \text{Nat}}{\text{iszero } t1 : \text{Bool}} \quad (\text{T-IsZero})$$

$$\frac{t1 : \text{Bool} \quad t2 : T \quad t3 : T}{\text{if } t1 \text{ then } t2 \text{ else } t3 : T} \quad (\text{T-If})$$

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 $\text{if } t1 \text{ then } t2 \text{ else } t3 : R$ then $t1 : \text{Bool}$ and $t2, t3 : R$
4. If $0 : R$ then $R = \text{Nat}$
5. If $\text{succ } t1 : R$ then $R = \text{Nat}$ and $t1 : \text{Nat}$
6. If $\text{pred } t1 : R$ then $R = \text{Nat}$ and $t1 : \text{Nat}$
7. If $\text{iszero } t1 : R$ then $R = \text{Bool}$ and $t1 : \text{Nat}$

1/28/03

Lesson 4: Typed Arith & Lambda

13

Typing Derivations

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

$$\begin{array}{c}
 \text{(T-IsZero)} \frac{0 : \text{Nat} \text{ (T-Zero)}}{\text{iszero}(0) : \text{Bool}} \quad 0 : \text{Nat} \quad \frac{0 : \text{Nat} \text{ (T-Zero)}}{\text{pred}(0) : \text{Nat}} \text{ (T-Pred)} \\
 \hline
 \text{if iszero}(0) \text{ then } 0 \text{ else pred } 0 : \text{Nat} \text{ (T-If)}
 \end{array}$$

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

1/28/03

Lesson 4: Typed Arith & Lambda

14

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.

1/28/03

Lesson 4: Typed Arith & Lambda

15

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"

1/28/03

Lesson 4: Typed Arith & Lambda

16

Cannonical forms

Defn: a cannonical 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.

1/28/03

Lesson 4: Typed Arith & Lambda

19

Simply typed lambda calculus

The abstract syntax of type terms is

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

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.

1/28/03

Lesson 4: Typed Arith & Lambda

20

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.

1/28/03

Lesson 4: Typed Arith & Lambda

21

Typing rule for lambda terms

$$\frac{\Gamma, x: T1 \vdash t2 : T2}{\Gamma \vdash \lambda x: T1. t2 : T1 \rightarrow T2} \quad (\text{T-Abs})$$

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.

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

1/28/03

Lesson 4: Typed Arith & Lambda

22

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.

1/28/03

Lesson 4: Typed Arith & Lambda

23

Typing rule for variables

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

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

1/28/03

Lesson 4: Typed Arith & Lambda

24

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 $\Gamma \vdash \lambda x. T1. t2 : R$ then $R = T1 \rightarrow R2$ for some $R2$ with $\Gamma, x : T1 \vdash t2 : R2$.
3. If $\Gamma \vdash t1 t2 : R$, then there is a $T11$ such that $\Gamma \vdash t1 : T11 \rightarrow R$ and $\Gamma \vdash t2 : T11$.
4. If $\Gamma \vdash \text{true} : R$ then $R = \text{Bool}$
5. If $\Gamma \vdash \text{false} : R$ then $R = \text{Bool}$
6. If $\Gamma \vdash \text{if } t1 \text{ then } t2 \text{ else } t3 : R$ then $\Gamma \vdash t1 : \text{Bool}$ and $\Gamma \vdash t2, t3 : R$

1/28/03

Lesson 4: Typed Arith & Lambda

25

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 $\Gamma \vdash t : T$.

1/28/03

Lesson 4: Typed Arith & Lambda

26

Canonical Forms (\square)

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 $v = \lambda x: T_1. t$.

1/28/03

Lesson 4: Typed Arith & Lambda

27

Progress (\square)

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 true`, then it may be in normal form yet not be a value.

1/28/03

Lesson 4: Typed Arith & Lambda

28

Permutation and Weakening

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

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 $\Gamma \vdash t : T$.

1/28/03

Lesson 4: Typed Arith & Lambda

29

Substitution Lemma

Lemma (9.3.8) [Preservation of types under substitutions]:
If $\Gamma, x : S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.

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 $\Gamma \vdash s : S$.

1/28/03

Lesson 4: Typed Arith & Lambda

30

Substitution Lemma

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

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

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 $\Gamma \vdash s: S$.

1/28/03

Lesson 4: Typed Arith & Lambda

31

Preservation (β_η)

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.

1/28/03

Lesson 4: Typed Arith & Lambda

32

Introduction and Elimination rules

□ Introduction

$$\frac{\square, x: T1 \vdash t2 : T2}{\square \vdash \lambda x: T1. t2 : T1 \rightarrow T2} \quad (\text{T-Abs})$$

□ Elimination

$$\frac{\square \vdash t1 : T11 \rightarrow T12 \quad \square \vdash t2 : T11}{\square \vdash t1 \ t2 : T12} \quad (\text{T-App})$$

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

1/28/03

Lesson 4: Typed Arith & Lambda

33

Erasure

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

$$\begin{aligned} \text{erase}(x) &= x \\ \text{erase}(\lambda x: T. t) &= \lambda x. \text{erase}(t) \\ \text{erase}(t1 \ t2) &= (\text{erase}(t1))(\text{erase}(t2)) \end{aligned}$$

erase maps a simply typed term in \square_τ to the corresponding untyped term in \square .

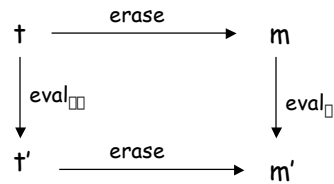
$$\text{erase}(\lambda x: \text{Bool}. \lambda y: \text{Bool} \rightarrow \text{Bool}. y \ x) = \lambda x. \lambda y. y \ x$$

1/28/03

Lesson 4: Typed Arith & Lambda

34

Erasure commutes with evaluation



Theorem (9.5.2)

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

1/28/03

Lesson 4: Typed Arith & Lambda

35

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.

1/28/03

Lesson 4: Typed Arith & Lambda

36

Homework

Modify the simplebool program to add arithmetic terms and a second primitive type Nat.

1. Add Nat, 0, succ, pred, iszero tokens to lexer and parser.
2. Extend the definition of terms in the parser with arithmetic forms (see tyarith)
3. Add type and term constructors to abstract syntax in syntax.sml, and modify print functions accordingly.
3. Modify the eval and typeof functions in core.sml to handle arithmetic expressions.

1/28/03

Lesson 4: Typed Arith & Lambda

37

Optional homework

Can you define the arithmetic plus operation in λ_{BN} (BN)?

1/28/03

Lesson 4: Typed Arith & Lambda

38

Sample

some text

1/28/03

Lesson 4: Typed Arith & Lambda

39

Rules

prem1	prem2	
<hr/>		(Label)
concl		

prem1	
<hr/>	(Label)
concl	

axiom	(Label)
-------	---------

1/28/03

Lesson 4: Typed Arith & Lambda

40

Symbols

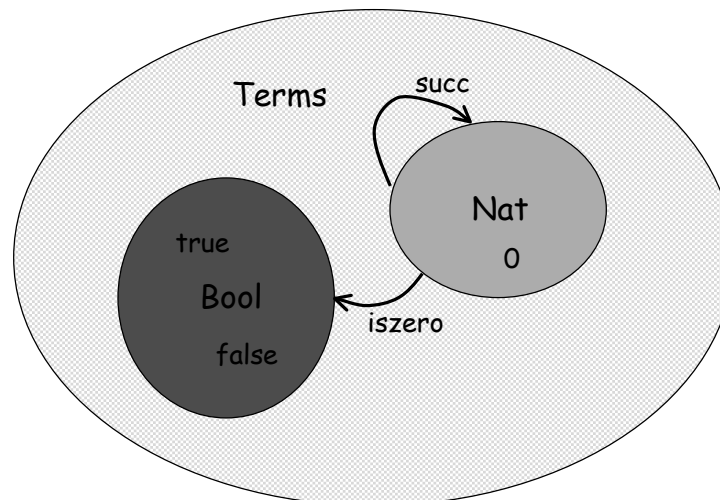
λ λ λ λ λ λ λ λ
 $\lambda \Rightarrow \rightarrow \rightarrow \rightarrow$
 $\emptyset \lambda \lambda \supseteq \lambda \lambda \lambda \lambda \lambda$
 \equiv
 $\lambda \lambda \lambda \lambda \lambda \lambda \lambda \lambda$
 $\lambda \Rightarrow \rightarrow \rightarrow \rightarrow$
 $\emptyset \lambda \lambda \supseteq \lambda \lambda \lambda \lambda \lambda$
 \equiv

1/28/03

Lesson 4: Typed Arith & Lambda

41

Space of terms



1/28/03

Lesson 4: Typed Arith & Lambda

42