

Programming Languages and Compilers (CS 421)

Elsa L Gunter

2112 SC, UIUC

[http://www.cs.uiuc.edu/class
/sp07/cs421/](http://www.cs.uiuc.edu/class/sp07/cs421/)

Based in part on slides by Mattox Beckman, as updated
by Vikram Adve and Gul Agha

Why Data Types?

- Data types play a key role in:
 - *Data abstraction* in the design of programs
 - *Type checking* in the analysis of programs
 - *Compile-time code generation* in the translation and execution of programs

Terminology

- Type: A type t defines a set of possible data values
 - E.g. `short` in C is $\{x \mid 2^{15} - 1 \geq x \geq -2^{15}\}$
 - A value in this set is said to have type t
- Type system: rules of a language assigning types to expressions

Types as Specifications

- Types describe properties
- Different type systems describe different properties, eg
 - Data is read-write versus read-only
 - Operation has authority to access data
 - Data came from “right” source
 - Operation might or could not raise an exception
- Common type systems focus on types describing same data layout and access methods

Sound Type System

- If an expression is assigned type t , and it evaluates to a value v , then v is in the set of values defined by t
- SML, OCAML, Scheme and Ada have sound type systems
- Most implementations of C and C++ do not

Strongly Typed Language

- When no application of an operator to arguments can lead to a run-time type error, language is *strongly typed*
 - Eg: $1 + 2.3;;$
- Depends on definition of “type error”

Strongly Typed Language

- C++ claimed to be “strongly typed”, but
 - Union types allow creating a value at one type and using it at another
 - Type coercions may cause unexpected (undesirable) effects
 - No array bounds check (in fact, no runtime checks at all)
- SML, OCAML “strongly typed” but still must do dynamic array bounds checks, runtime type case analysis, and other checks

Static vs Dynamic Types

- *Static type*: type assigned to an expression at compile time
- *Dynamic type*: type assigned to a storage location at run time
- *Statically typed language*: static type assigned to every expression at compile time
- *Dynamically typed language*: type of an expression determined at run time

Type Checking

- When is $op(arg_1, \dots, arg_n)$ allowed?
- *Type checking* assures that operations are applied to the right number of arguments of the right types
 - Right type may mean same type as was specified, or may mean that there is a predefined implicit coercion that will be applied
- Used to resolve overloaded operations

Type Checking

- Type checking may be done *statically* at compile time or *dynamically* at run time
- Dynamically typed (aka untyped) languages (eg LISP, Prolog) do only dynamic type checking
- Statically typed languages can do most type checking statically

Dynamic Type Checking

- Performed at run-time before each operation is applied
- Types of variables and operations left unspecified until run-time
 - Same variable may be used at different types

Dynamic Type Checking

- Data object must contain type information
- Errors aren't detected until violating application is executed (maybe years after the code was written)

Static Type Checking

- Performed after parsing, before code generation
- Type of every variable and signature of every operator must be known at compile time

Static Type Checking

- Can eliminate need to store type information in data object if no dynamic type checking is needed
- Catches many programming errors at earliest point
- Can't check types that depend on dynamically computed values
 - Eg: array bounds

Static Type Checking

- Typically places restrictions on languages
 - Garbage collection
 - References instead of pointers
 - All variables initialized when created
 - Variable only used at one type
 - Union types allow for work-arounds, but effectively introduce dynamic type checks

Type Declarations

- *Type declarations*: explicit assignment of types to variables (signatures to functions) in the code of a program
 - Must be checked in a strongly typed language
 - Often not necessary for strong typing or even static typing (depends on the type system)

Type Inference

- *Type inference*: A program analysis to assign a type to an expression from the program context of the expression
 - Fully static type inference first introduced by Robin Miller in ML
 - Haskell, OCAML, SML all use type inference
 - Records are a problem for type inference

Format of Type Judgments

- A *type judgment* has the form
$$\Gamma \vdash \text{exp} : \tau$$
- Γ is a typing environment
 - Supplies the types of variables and functions
 - Γ is a list of the form $[x : \sigma, \dots]$
- exp is a program expression
- τ is a type to be assigned to exp
- \vdash pronounced “turnstile”, or “entails” (or “satisfies”)

Example Valid Type Judgments

- $[] \vdash \text{true or false} : \text{bool}$
- $[x : \text{int}] \vdash x + 3 : \text{int}$
- $[p : \text{int} \rightarrow \text{string}] \vdash p(5) : \text{string}$

Format of Typing Rules

Assumptions

$$\frac{\Gamma_1 \vdash \text{exp}_1 : \tau_1 \quad \dots \quad \Gamma_n \vdash \text{exp}_n : \tau_n}{\Gamma \vdash \text{exp} : \tau}$$

Conclusion

- Idea: Type of expression determined by type of components
- Rule without assumptions is called an *axiom*
- Γ may be omitted when not needed

Format of Typing Rules

Assumptions

$$\frac{\Gamma_1 \vdash \text{exp}_1 : \tau_1 \quad \dots \quad \Gamma_n \vdash \text{exp}_n : \tau_n}{\Gamma \vdash \text{exp} : \tau}$$

Conclusion

- Γ , exp , τ are *parameterized* environments, expressions and types - *i.e.* may contain *meta-variables*

Axioms - Constants

$\overline{\text{ |- } n : \text{int}}$ (assuming n is an integer constant)

$\overline{\text{ |- true : bool}}$

$\overline{\text{ |- false : bool}}$

- These rules are true with any typing environment
- n is a meta-variable

Axioms - Variables

Notation: Let $\Gamma(x) = \sigma$ if $x : \sigma \in \Gamma$ and there is no $x : \tau$ to the left of $x : \sigma$ in Γ

Variable axiom:

$$\frac{}{\Gamma \vdash x : \sigma} \quad \text{if } \Gamma(x) = \sigma$$

Simple Rules - Arithmetic

Primitive operators ($\oplus \in \{+, -, *, \dots\}$):

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}}$$

Relations ($\sim \in \{<, >, =, <=, >= \}$):

$$\frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 \sim e_2 : \text{bool}}$$

Simple Rules - Booleans

Connectives

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ \&\& \ e_2 : \text{bool}}$$

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 \ || \ e_2 : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Start building the proof tree from the bottom up

$$\frac{\quad}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}} \text{?}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Which rule has this as a conclusion?

$$\frac{\quad}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}} \text{?}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Booleans: \parallel

$$\frac{\Gamma \vdash y : \text{bool} \quad \Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Pick an assumption to prove

$$\frac{\frac{?}{\Gamma \vdash y : \text{bool}} \quad \Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Which rule has this as a conclusion?

$$\frac{\frac{?}{\Gamma \vdash y : \text{bool}} \quad \Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Axiom for variables

$$\frac{\Gamma \vdash y : \text{bool} \quad \Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Pick an assumption to prove

$$\frac{\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad ?}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Which rule has this as a conclusion?

$$\frac{\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash x + 3 > 6 : \text{bool}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}} \quad ?$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Arithmetic relations

$$\frac{\Gamma \vdash y : \text{bool} \quad \frac{\Gamma \vdash x + 3 : \text{int} \quad \Gamma \vdash 6 : \text{int}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Pick an assumption to prove

$$\frac{\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash x + 3 : \text{int} \quad \frac{\Gamma \vdash 6 : \text{int}}{?}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Which rule has this as a conclusion?

$$\frac{\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash x + 3 : \text{int} \quad \frac{\Gamma \vdash 6 : \text{int}}{?}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Axiom for constants

$$\frac{\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\Gamma \vdash x + 3 : \text{int} \quad \overline{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Pick an assumption to prove

$$\frac{\frac{\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\frac{\Gamma \vdash x + 3 : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 6 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Which rule has this as a conclusion?

$$\frac{\frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y : \text{bool}} \quad \frac{\frac{\Gamma \vdash x + 3 : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 6 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Arithmetic operations

$$\frac{\frac{\frac{\Gamma \vdash x : \text{int} \quad \Gamma \vdash 3 : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \Gamma \vdash 6 : \text{int}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \Gamma \vdash y : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Pick an assumption to prove

$$\frac{\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x : \text{int}} \quad \frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash 3 : \text{int}}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 6 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \frac{\Gamma \vdash y : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Which rule has this as a conclusion?

$$\frac{\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \Gamma \vdash y : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Axiom for constants

$$\frac{\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \Gamma \vdash y : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Pick an assumption to prove

$$\frac{\frac{\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x + 3 : \text{int}}}{\Gamma \vdash y : \text{bool}} \quad \frac{\frac{\frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Which rule has this as a conclusion?

$$\frac{\frac{\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x + 3 : \text{int}}}{\Gamma \vdash y : \text{bool}} \quad \frac{\frac{\frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- Axiom for variables

$$\frac{\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \Gamma \vdash y : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Simple Example

- Let $\Gamma = [x:\text{int} ; y:\text{bool}]$
- Show $\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}$
- No more assumptions! **DONE!**

$$\frac{\frac{\frac{\Gamma \vdash x : \text{int}}{\Gamma \vdash x + 3 : \text{int}} \quad \frac{\Gamma \vdash 3 : \text{int}}{\Gamma \vdash 6 : \text{int}}}{\Gamma \vdash x + 3 > 6 : \text{bool}} \quad \Gamma \vdash y : \text{bool}}{\Gamma \vdash y \parallel (x + 3 > 6) : \text{bool}}$$

Type Variables in Rules

- If_then_else rule:

$$\frac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}$$

- τ is a type variable (meta-variable)
- Can take any type at all
- All instances in a rule application must get same type
- Then branch, else branch and if_then_else must all have same type

Function Application

- Application rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash (e_1 e_2) : \tau_2}$$

- If you have a function expression e_1 of type $\tau_1 \rightarrow \tau_2$ applied to an argument of type τ_1 , the resulting expression has type τ_2

Application Examples

$$\frac{\Gamma \vdash \text{print_int} : \text{int} \rightarrow \text{unit} \quad \Gamma \vdash 5 : \text{int}}{\Gamma \vdash (\text{print_int } 5) : \text{unit}}$$

- $e_1 = \text{print_int}$, $e_2 = 5$,
- $\tau_1 = \text{int}$, $\tau_2 = \text{unit}$

$$\frac{\Gamma \vdash \text{map print_int} : \text{int list} \rightarrow \text{unit list} \quad \Gamma \vdash [3;7] : \text{int list}}{\Gamma \vdash (\text{map print_int } [3; 7]) : \text{unit list}}$$

- $e_1 = \text{map print_int}$, $e_2 = [3; 7]$,
- $\tau_1 = \text{int list}$, $\tau_2 = \text{unit list}$

Fun Rule

- Rules describe types, but also how the environment Γ may change
- Can only do what rule allows!
- fun rule:

$$\frac{[x : \tau_1] \cup \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

Fun Examples

$$\frac{[y : \text{int}] \cup \Gamma \vdash y + 3 : \text{int}}{\Gamma \vdash \text{fun } y \rightarrow y + 3 : \text{int} \rightarrow \text{int}}$$

$$\frac{[f : \text{int} \rightarrow \text{bool}] \cup \Gamma \vdash f\ 2 :: [\text{true}] : \text{bool list}}{\Gamma \vdash (\text{fun } f \rightarrow f\ 2 :: [\text{true}]) : (\text{int} \rightarrow \text{bool}) \rightarrow \text{bool list}}$$

Let and Let Rec

- let rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad [x : \tau_1] \cup \Gamma \vdash e_2 : \tau_2}{(\text{let } x = e_1 \text{ in } e_2) : \tau_2}$$

- let rec rule:

$$\frac{[x : \tau_1] \cup \Gamma \vdash e_1 : \tau_1 \quad [x : \tau_1] \cup \Gamma \vdash e_2 : \tau_2}{(\text{let rec } x = e_1 \text{ in } e_2) : \tau_2}$$

Mia Copa

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only meta-variable in the logic)
- Would need:
 - Object level type variables and some kind of type quantification
 - **let** and **let rec** rules to introduce polymorphism
 - Explicit rule to eliminate (instantiate) polymorphism

Example

- Which rule do we apply?

?

|- (let rec one = 1 :: one in
let x = 2 in
fun y -> (x :: y :: one)) : int → int list

Example

- Let rec rule: $\textcircled{2}$ $[one : \text{int list}] \vdash$
 $\textcircled{1}$ $(\text{let } x = 2 \text{ in}$
 $[one : \text{int list}] \vdash$ $\text{fun } y \rightarrow (x :: y :: one))$
 $(1 :: one) : \text{int list}$ $\quad \quad \quad : \text{int} \rightarrow \text{int list}$

$\vdash (\text{let rec one} = 1 :: one \text{ in}$
 $\text{let } x = 2 \text{ in}$
 $\text{fun } y \rightarrow (x :: y :: one)) : \text{int} \rightarrow \text{int list}$

Proof of 1

- Which rule?

$[one : int\ list] \vdash (1 :: one) : int\ list$

Proof of 1

- Application

③

$[one : int\ list] \vdash$
 $((::) 1) : int\ list \rightarrow int\ list$

④

$[one : int\ list] \vdash$
 $one : int\ list$

$[one : int\ list] \vdash (1 :: one) : int\ list$

Proof of 3

Constants Rule

Constants Rule

$[one : int\ list] \vdash$

$(::) : int \rightarrow int\ list \rightarrow int\ list$

$[one : int\ list] \vdash$

$1 : int$

$[one : int\ list] \vdash ((::) 1) : int\ list \rightarrow int\ list$

Proof of 4

- Rule for variables

$$\frac{}{[\text{one} : \text{int list}] \vdash \text{one}:\text{int list}}$$

Proof of 2

Constant

⑤ $[x:\text{int}; \text{one} : \text{int list}] \vdash$

$\text{fun } y \rightarrow$

$(x :: y :: \text{one}))$

$[\text{one} : \text{int list}] \vdash 2:\text{int}$

$:\text{int} \rightarrow \text{int list}$

$[\text{one} : \text{int list}] \vdash (\text{let } x = 2 \text{ in}$

$\text{fun } y \rightarrow (x :: y :: \text{one})) : \text{int} \rightarrow \text{int list}$

Proof of 5

?

$[x:\text{int}; \text{one} : \text{int list}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one}))$
 $: \text{int} \rightarrow \text{int list}$

Proof of 5

?

$$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash (x :: y :: \text{one}) : \text{int list}$$

$$[x:\text{int}; \text{one} : \text{int list}] \vdash \text{fun } y \text{ -> } (x :: y :: \text{one})) \\ : \text{int} \rightarrow \text{int list}$$

Proof of 5

⑥

$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash$
 $((::) x):\text{int list} \rightarrow \text{int list}$

⑦

$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash$
 $(y :: \text{one}) : \text{int list}$

$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash (x :: y :: \text{one}) : \text{int list}$

$[x:\text{int}; \text{one} : \text{int list}] \vdash \text{fun } y \rightarrow (x :: y :: \text{one})$
 $: \text{int} \rightarrow \text{int list}$

Proof of 6

Constant

Variable

$[\dots] \vdash (::)$

$: \text{int} \rightarrow \text{int list} \rightarrow \text{int list}$

$[\dots; x:\text{int}; \dots] \vdash x:\text{int}$

$[y:\text{int}; x:\text{int}; \text{one} : \text{int list}] \vdash ((::) x)$

$:\text{int list} \rightarrow \text{int list}$

Proof of 7

Pf of 6 [y/x]

•
•
•

[y:int; ...] |- ((::) y)
:int list → int list

Variable

[...; one: int list] |-
one: int list

[y:int; x:int; one : int list] |- (y :: one) : int list

Curry - Howard Isomorphism

- Type Systems are logics; logics are type systems
- Types are propositions; propositions are types
- Terms are proofs; proofs are terms

- Functions space arrow corresponds to implication; application corresponds to modus ponens

Curry - Howard Isomorphism

- Modus Ponens

$$\frac{A \Rightarrow B \quad A}{B}$$

- Application

$$\frac{\Gamma \vdash e_1 : \alpha \rightarrow \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash (e_1 e_2) : \beta}$$