

Programming Languages and Compilers (CS 421)

Elsa L Gunter
2112 SC, UIUC

<http://courses.engr.illinois.edu/cs421>

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

10/15/19

1

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
 - Data layout (how many words; which are data and which are pointers) dictated by type

10/15/19

2

Terminology

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

10/15/19

3

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

10/15/19

4

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

10/15/19

5

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”

10/15/19

6

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

10/15/19

7

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

10/15/19

8

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

10/15/19

9

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

10/15/19

10

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

10/15/19

11

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)

10/15/19

12

Static Type Checking

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

10/15/19

13

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

10/15/19

14

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

10/15/19

15

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)

10/15/19

16

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

10/15/19

17

Format of Type Judgments

- A **type judgement** has the form
 $\Gamma \vdash \text{exp} : \tau$
- Γ is a typing environment
 - Supplies the types of variables (and function names when function names are not variables)
 - Γ is a set of the form $\{x:\sigma, \dots\}$
 - For any x at most one σ such that $(x:\sigma \in \Gamma)$
- exp is a program expression
- τ is a type to be assigned to exp
- \vdash pronounced “turnstyle”, or “entails” (or “satisfies” or, informally, “shows”)

10/15/19

18

Axioms - Constants

$\Gamma \vdash n : \text{int}$ (assuming n is an integer constant)

$\Gamma \vdash \text{true} : \text{bool}$ $\Gamma \vdash \text{false} : \text{bool}$

- These rules are true with any typing environment
- Γ, n are meta-variables

10/15/19

19

Axioms – Variables (Monomorphic Rule)

Notation: Let $\Gamma(x) = \sigma$ if $x : \sigma \in \Gamma$

Note: if such σ exists, its unique

Variable axiom:

$\Gamma \vdash x : \sigma$ if $\Gamma(x) = \sigma$

10/15/19

20

Simple Rules - Arithmetic

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

$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad (\oplus) : \tau_1 \rightarrow \tau_2 \rightarrow \tau_3}{\Gamma \vdash e_1 \oplus e_2 : \tau_3}$

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

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

For the moment, think τ is `int`

10/15/19

21

Example: $\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}$

What do we need to show first?

$\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}$

10/15/19

22

Example: $\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}$

What do we need for the left side?

$\frac{\{x : \text{int}\} \vdash x + 2 : \text{int} \quad \{x:\text{int}\} \vdash 3 : \text{int}}{\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}} \text{Rel}$

10/15/19

23

Example: $\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}$

How to finish?

$\frac{\frac{\{x:\text{int}\} \vdash x:\text{int} \quad \{x:\text{int}\} \vdash 2:\text{int}}{\{x : \text{int}\} \vdash x + 2 : \text{int}} \text{AO} \quad \{x:\text{int}\} \vdash 3 : \text{int}}{\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}} \text{Rel}$

10/15/19

24

Example: $\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}$

Complete Proof (type derivation)

$$\frac{\frac{\text{Var}}{\{x:\text{int}\} \vdash x:\text{int}} \quad \frac{\text{Const}}{\{x:\text{int}\} \vdash 2:\text{int}}}{\{x:\text{int}\} \vdash x + 2 : \text{int}} \text{AO} \quad \frac{\text{Const}}{\{x:\text{int}\} \vdash 3:\text{int}}}{\{x:\text{int}\} \vdash x + 2 = 3 : \text{bool}} \text{Rel}$$

10/15/19

25

Simple Rules - Booleans

Connectives

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

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

10/15/19

26

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

10/15/19

27

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 e_2 of type τ_1 , the resulting expression $e_1 e_2$ has type τ_2

10/15/19

28

Fun Rule

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

$$\frac{\{x:\tau_1\} + \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

10/15/19

29

Fun Examples

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

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

10/15/19

30

(Monomorphic) Let and Let Rec

- let rule:

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

- let rec rule:

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

10/15/19

31

Example

- Which rule do we apply?

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

10/15/19

32

Example

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

10/15/19

33

Proof of 1

- Which rule?

$$\{one : \text{int list}\} \vdash (1 :: \text{one}) : \text{int list}$$

10/15/19

34

Proof of 1

- Application

$$\frac{\begin{array}{c} \textcircled{3} \\ \{one : \text{int list}\} \vdash \\ ((::) 1) : \text{int list} \rightarrow \text{int list} \end{array} \quad \begin{array}{c} \textcircled{4} \\ \{one : \text{int list}\} \vdash \\ \text{one} : \text{int list} \end{array}}{\{one : \text{int list}\} \vdash (1 :: \text{one}) : \text{int list}}$$

10/15/19

35

Proof of 3

Constants Rule

Constants Rule

$$\frac{\frac{\{one : \text{int list}\} \vdash \quad \{one : \text{int list}\} \vdash}{((::) : \text{int} \rightarrow \text{int list}) \rightarrow \text{int list} \quad 1 : \text{int}}}{\{one : \text{int list}\} \vdash ((::) 1) : \text{int list} \rightarrow \text{int list}}$$

10/15/19

36

Proof of 4

- Rule for variables

$$\frac{}{\{one : int\ list\} \vdash one : int\ list}$$

10/15/19

37

Proof of 2

⑤ $\{x:int; one : int\ list\} \vdash$
 $fun\ y\ \rightarrow$
 $(x :: y :: one)$
 $: int \rightarrow int\ list$

$$\frac{\{one : int\ list\} \vdash 2:int \quad \{x:int; one : int\ list\} \vdash fun\ y\ \rightarrow (x :: y :: one) : int \rightarrow int\ list}{\{one : int\ list\} \vdash (let\ x = 2\ in\ fun\ y\ \rightarrow (x :: y :: one)) : int \rightarrow int\ list}$$

10/15/19

38

Proof of 5

$$\frac{?}{\{x:int; one : int\ list\} \vdash fun\ y\ \rightarrow (x :: y :: one) : int \rightarrow int\ list}$$

10/15/19

39

Proof of 5

$$\frac{?}{\frac{\{y:int; x:int; one : int\ list\} \vdash (x :: y :: one) : int\ list}{\{x:int; one : int\ list\} \vdash fun\ y\ \rightarrow (x :: y :: one) : int \rightarrow int\ list}}$$

10/15/19

40

Proof of 5

⑥ $\frac{\{y:int; x:int; one:int\ list\} \vdash ((::) x) : int\ list \rightarrow int\ list}{\{y:int; x:int; one : int\ list\} \vdash (x :: y :: one) : int\ list}$

⑦ $\frac{\{y:int; x:int; one:int\ list\} \vdash (y :: one) : int\ list}{\{x:int; one : int\ list\} \vdash fun\ y\ \rightarrow (x :: y :: one) : int \rightarrow int\ list}$

10/15/19

41

Proof of 6

Constant Variable

$$\frac{\{...\} \vdash (::)}{: int \rightarrow int\ list \rightarrow int\ list} \quad \frac{\{...\; x:int;...\} \vdash x:int}{\{y:int; x:int; one : int\ list\} \vdash ((::) x) : int\ list \rightarrow int\ list}$$

10/15/19

42

Proof of 7

Pf of 6 [y/x]

Variable

•
•
•

$$\frac{\frac{\{y:\text{int}; \dots\} \vdash ((::) y) \quad \{...\; \text{one: int list}\} \vdash \text{one: int list}}{\text{one: int list}}}{\{y:\text{int}; x:\text{int}; \text{one} : \text{int list}\} \vdash (y :: \text{one}) : \text{int list}}$$

10/15/19

43

Curry - Howard Isomorphism

- Type Systems are logics; logics are type systems
- Types are propositions; propositions are types
- Terms are proofs; proofs are terms
- Function space arrow corresponds to implication; application corresponds to modus ponens

10/15/19

44

Curry - Howard Isomorphism

- Modus Ponens

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

- Application

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

10/15/19

45

Mea Culpa

- 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

10/15/19

46

Support for Polymorphic Types

- Monomorphic Types (τ):
 - Basic Types: `int`, `bool`, `float`, `string`, `unit`, ...
 - Type Variables: $\alpha, \beta, \gamma, \delta, \epsilon$
 - Compound Types: $\alpha \rightarrow \beta$, `int * string`, `bool list`, ...
- Polymorphic Types:
 - Monomorphic types τ
 - Universally quantified monomorphic types
 - $\forall \alpha_1, \dots, \alpha_n. \tau$
 - Can think of τ as same as $\forall. \tau$

10/15/19

47

Example FreeVars Calculations

- $\text{Vars}(\text{'a} \rightarrow (\text{int} \rightarrow \text{'b}) \rightarrow \text{'a}) = \{\text{'a}, \text{'b}\}$
- $\text{FreeVars} (\text{All 'b. 'a} \rightarrow (\text{int} \rightarrow \text{'b}) \rightarrow \text{'a}) = \{\text{'a}, \text{'b}\} - \{\text{'b}\} = \{\text{'a}\}$
- $\text{FreeVars} \{x : \text{All 'b. 'a} \rightarrow (\text{int} \rightarrow \text{'b}) \rightarrow \text{'a},$
- $\text{id} : \text{All 'c. 'c} \rightarrow \text{'c},$
- $y : \text{All 'c. 'a} \rightarrow \text{'b} \rightarrow \text{'c}\} = \{\text{'a}\} \cup \{\} \cup \{\text{'a}, \text{'b}\} = \{\text{'a}, \text{'b}\}$

10/15/19

48

Support for Polymorphic Types

- Typing Environment Γ supplies polymorphic types (which will often just be monomorphic) for variables
- Free variables of monomorphic type just type variables that occur in it
 - Write $\text{FreeVars}(\tau)$
- Free variables of polymorphic type removes variables that are universally quantified
 - $\text{FreeVars}(\forall \alpha_1, \dots, \alpha_n. \tau) = \text{FreeVars}(\tau) - \{\alpha_1, \dots, \alpha_n\}$
- $\text{FreeVars}(\Gamma) =$ all FreeVars of types in range of Γ

10/15/19

49

Monomorphic to Polymorphic

- Given:
 - type environment Γ
 - monomorphic type τ
 - τ shares type variables with Γ
- Want most polymorphic type for τ that doesn't break sharing type variables with Γ
- $\text{Gen}(\tau, \Gamma) = \forall \alpha_1, \dots, \alpha_n. \tau$ where $\{\alpha_1, \dots, \alpha_n\} = \text{freeVars}(\tau) - \text{freeVars}(\Gamma)$

10/15/19

50

Polymorphic Typing Rules

- A *type judgement* has the form $\Gamma \vdash \text{exp} : \tau$
 - Γ uses *polymorphic* types
 - τ still monomorphic
- Most rules stay same (except use more general typing environments)
- Rules that change:
 - Variables
 - Let and Let Rec
 - Allow polymorphic constants
- Worth noting functions again

10/15/19

51

Polymorphic Let and Let Rec

- let rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \{x : \text{Gen}(\tau_1, \Gamma)\} + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let } x = e_1 \text{ in } e_2) : \tau_2}$$
- let rec rule:

$$\frac{\{x : \tau_1\} + \Gamma \vdash e_1 : \tau_1 \quad \{x : \text{Gen}(\tau_1, \Gamma)\} + \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (\text{let rec } x = e_1 \text{ in } e_2) : \tau_2}$$

10/15/19

52

Polymorphic Variables (Identifiers)

Variable axiom:

$$\frac{}{\Gamma \vdash x : \varphi(\tau)} \quad \text{if } \Gamma(x) = \forall \alpha_1, \dots, \alpha_n. \tau$$

- Where φ replaces all occurrences of $\alpha_1, \dots, \alpha_n$ by monotypes τ_1, \dots, τ_n
- Note: Monomorphic rule special case:

$$\frac{}{\Gamma \vdash x : \tau} \quad \text{if } \Gamma(x) = \tau$$
- Constants treated same way

10/15/19

53

Fun Rule Stays the Same

- fun rule:

$$\frac{\{x : \tau_1\} + \Gamma \vdash e : \tau_2}{\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2}$$

- Types τ_1, τ_2 monomorphic
- Function argument must always be used at same type in function body

10/15/19

54

Polymorphic Example

- Assume additional constants:
- $\text{hd} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha$
- $\text{tl} : \forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $\text{is_empty} : \forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$
- $(::) : \forall \alpha. \alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$
- $[] : \forall \alpha. \alpha \text{ list}$

10/15/19

55

Polymorphic Example

- Show:

$$\frac{?}{\{\} \vdash \text{let rec length} = \text{fun } l \rightarrow \text{if is_empty } l \text{ then } 0 \text{ else } 1 + \text{length (tl } l) \text{ in length ((::) 2 [])} + \text{length}((::) \text{true []}) : \text{int}}$$

10/15/19

56

Polymorphic Example: Let Rec Rule

- Show: (1) (2)
- $$\frac{\{\text{length} : \alpha \text{ list} \rightarrow \text{int}\} \vdash \text{fun } l \rightarrow \dots : \alpha \text{ list} \rightarrow \text{int} \quad \{\text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\} \vdash \text{length ((::) 2 [])} + \text{length}((::) \text{true []}) : \text{int}}{\{\} \vdash \text{let rec length} = \text{fun } l \rightarrow \text{if is_empty } l \text{ then } 0 \text{ else } 1 + \text{length (tl } l) \text{ in length ((::) 2 [])} + \text{length}((::) \text{true []}) : \text{int}}$$

10/15/19

57

Polymorphic Example (1)

- Show:

$$\frac{?}{\{\text{length} : \alpha \text{ list} \rightarrow \text{int}\} \vdash \text{fun } l \rightarrow \text{if is_empty } l \text{ then } 0 \text{ else } 1 + \text{length (tl } l) : \alpha \text{ list} \rightarrow \text{int}}$$

10/15/19

58

Polymorphic Example (1): Fun Rule

- Show: (3)
- $$\frac{\{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\} \vdash \text{if is_empty } l \text{ then } 0 \text{ else length (hd } l) + \text{length (tl } l) : \text{int}}{\{\text{length} : \alpha \text{ list} \rightarrow \text{int}\} \vdash \text{fun } l \rightarrow \text{if is_empty } l \text{ then } 0 \text{ else } 1 + \text{length (tl } l) : \alpha \text{ list} \rightarrow \text{int}}$$

10/15/19

59

Polymorphic Example (3)

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

$$\frac{?}{\Gamma \vdash \text{if is_empty } l \text{ then } 0 \text{ else } 1 + \text{length (tl } l) : \text{int}}$$

10/15/19

60

Polymorphic Example (3):IfThenElse

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

$$\frac{\begin{array}{ccc} (4) & (5) & (6) \\ \Gamma \vdash \text{is_empty } l & \Gamma \vdash 0 : \text{int} & \Gamma \vdash 1 + \\ & : \text{bool} & \text{length } (tl \ l) : \text{int} \end{array}}{\Gamma \vdash \text{if is_empty } l \text{ then } 0 \\ \text{else } 1 + \text{length } (tl \ l) : \text{int}}$$

10/15/19

61

Polymorphic Example (4)

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

$$\frac{?}{\Gamma \vdash \text{is_empty } l : \text{bool}}$$

10/15/19

62

Polymorphic Example (4):Application

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

$$\frac{\frac{?}{\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}} \quad \frac{?}{\Gamma \vdash l : \alpha \text{ list}}}{\Gamma \vdash \text{is_empty } l : \text{bool}}$$

10/15/19

63

Polymorphic Example (4)

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

By Const since $\alpha \text{ list} \rightarrow \text{bool}$ is
instance of $\forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$?

$$\frac{\frac{\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}}{\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}} \quad \frac{\Gamma \vdash l : \alpha \text{ list}}{\Gamma \vdash l : \alpha \text{ list}}}{\Gamma \vdash \text{is_empty } l : \text{bool}}$$

10/15/19

64

Polymorphic Example (4)

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

By Const since $\alpha \text{ list} \rightarrow \text{bool}$ is
instance of $\forall \alpha. \alpha \text{ list} \rightarrow \text{bool}$ By Variable
 $\Gamma(l) = \alpha \text{ list}$

$$\frac{\frac{\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}}{\Gamma \vdash \text{is_empty} : \alpha \text{ list} \rightarrow \text{bool}} \quad \frac{\Gamma \vdash l : \alpha \text{ list}}{\Gamma \vdash l : \alpha \text{ list}}}{\Gamma \vdash \text{is_empty } l : \text{bool}}$$

- This finishes (4)

10/15/19

65

Polymorphic Example (5):Const

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

By Const Rule

$$\frac{}{\Gamma \vdash 0 : \text{int}}$$

10/15/19

66

Polymorphic Example (6):Arith Op

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

$$\frac{\text{By Const} \quad \frac{\Gamma \vdash 1 : \text{int}}{\Gamma \vdash 1 : \text{int}} \quad \frac{\text{By Variable} \quad \frac{\Gamma \vdash \text{length} : \alpha \text{ list} \rightarrow \text{int} \quad \Gamma \vdash (tl \ l) : \alpha \text{ list}}{\Gamma \vdash \text{length} (tl \ l) : \text{int}}}{\Gamma \vdash 1 + \text{length} (tl \ l) : \text{int}} \quad (7)$$

10/15/19

67

Polymorphic Example (7):App Rule

- Let $\Gamma = \{\text{length} : \alpha \text{ list} \rightarrow \text{int}, l : \alpha \text{ list}\}$
- Show

$$\frac{\text{By Const} \quad \frac{\Gamma \vdash tl : \alpha \text{ list} \rightarrow \alpha \text{ list}}{\Gamma \vdash tl : \alpha \text{ list} \rightarrow \alpha \text{ list}} \quad \frac{\text{By Variable} \quad \frac{\Gamma \vdash l : \alpha \text{ list}}{\Gamma \vdash l : \alpha \text{ list}}}{\Gamma \vdash (tl \ l) : \alpha \text{ list}}$$

By Const since $\alpha \text{ list} \rightarrow \alpha \text{ list}$ is instance of $\forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$

10/15/19

68

Polymorphic Example: (2) by ArithOp

- Let $\Gamma' = \{\text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:

$$\frac{\text{(8)} \quad \frac{\Gamma' \vdash \text{length} ((::) \ 2 \ []) : \text{int}}{\{\text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}} \quad \text{(9)} \quad \frac{\Gamma' \vdash \text{length} ((::) \ \text{true} \ []) : \text{int}}{\{\text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}}}{\Gamma' \vdash \text{length} ((::) \ 2 \ []) + \text{length} ((::) \ \text{true} \ []) : \text{int}}$$

10/15/19

69

Polymorphic Example: (8)AppRule

- Let $\Gamma' = \{\text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:

$$\frac{\Gamma' \vdash \text{length} : \text{int list} \rightarrow \text{int} \quad \Gamma' \vdash ((::) \ 2 \ []) : \text{int list}}{\Gamma' \vdash \text{length} ((::) \ 2 \ []) : \text{int}}$$

10/15/19

70

Polymorphic Example: (8)AppRule

- Let $\Gamma' = \{\text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
 - Show:
- By Var since $\text{int list} \rightarrow \text{int}$ is instance of $\forall \alpha. \alpha \text{ list} \rightarrow \text{int}$ (by $\alpha \rightarrow \text{int}$)

$$\frac{\Gamma' \vdash \text{length} : \text{int list} \rightarrow \text{int} \quad \Gamma' \vdash ((::) \ 2 \ []) : \text{int list}}{\Gamma' \vdash \text{length} ((::) \ 2 \ []) : \text{int}} \quad (10)$$

10/15/19

71

Polymorphic Example: (10)AppRule

- Let $\Gamma' = \{\text{length} : \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
 - Show:
- By Const since int list is instance of $\forall \alpha. \alpha \text{ list}$ (by $\alpha \rightarrow \text{int}$)

$$\frac{\Gamma' \vdash ((::) \ 2) : \text{int list} \rightarrow \text{int list} \quad \Gamma' \vdash [] : \text{int list}}{\Gamma' \vdash ((::) \ 2 \ []) : \text{int list}} \quad (11)$$

10/15/19

72

Polymorphic Example: (11)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Const since $\text{int} \rightarrow \text{int list} \rightarrow \text{int list}$ is instance of

$$\frac{\frac{\forall \alpha. \alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}}{\Gamma' \vdash (:) : \text{int} \rightarrow \text{int list} \rightarrow \text{int list}} \quad \text{By Const}}{\Gamma' \vdash ((:) 2) : \text{int list} \rightarrow \text{int list}}$$

10/15/19

73

Polymorphic Example: (9)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:

$$\frac{\Gamma' \vdash \text{length} : \text{bool list} \rightarrow \text{int} \quad \Gamma' \vdash ((:) \text{true } []) : \text{bool list}}{\Gamma' \vdash \text{length } ((:) \text{true } []) : \text{int}}$$

10/15/19

74

Polymorphic Example: (9)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Var since $\text{bool list} \rightarrow \text{int}$ is instance of $\forall \alpha. \alpha \text{ list} \rightarrow \text{int}$

$$\frac{\frac{\Gamma' \vdash \text{length} : \text{bool list} \rightarrow \text{int} \quad \Gamma' \vdash ((:) \text{true } []) : \text{bool list}}{\Gamma' \vdash \text{length } ((:) \text{true } []) : \text{int}} \quad (12)}$$

10/15/19

75

Polymorphic Example: (12)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:
- By Const since bool list is instance of $\forall \alpha. \alpha \text{ list}$

$$\frac{\Gamma' \vdash ((:) \text{true}) : \text{bool list} \rightarrow \text{bool list} \quad \Gamma' \vdash [] : \text{bool list}}{\Gamma' \vdash ((:) \text{true } []) : \text{bool list}} \quad (13)$$

10/15/19

76

Polymorphic Example: (13)AppRule

- Let $\Gamma' = \{\text{length} \forall \alpha. \alpha \text{ list} \rightarrow \text{int}\}$
- Show:

By Const since $\text{bool} \rightarrow \text{bool list} \rightarrow \text{bool list}$ is instance of $\forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}$

$$\frac{\frac{\text{of } \forall \alpha. \alpha \text{ list} \rightarrow \alpha \text{ list}}{\Gamma' \vdash (:) : \text{bool} \rightarrow \text{bool list} \rightarrow \text{bool list}} \quad \frac{\text{By Const}}{\Gamma' \vdash \text{true} : \text{bool}}}{\Gamma' \vdash ((:) \text{true}) : \text{bool list} \rightarrow \text{bool list}}$$

10/15/19

77