# 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



- 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  $\{x \mid 2^{15} 1 \ge x \ge -2^{15}\}$
  - A value in this set is said to have type t
- Type system: rules of a language assigning types to expressions

10/15/19



## 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

4

#### 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 5

10/15/19

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



#### 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(arg1,...,argn) 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



#### 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



#### 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 11

10/15/19



#### 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



# 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
  - Eq: 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



13

17

## 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
  - Haskle, OCAML, SML all use type inference
    - Records are a problem for type inference

10/15/19



#### Format of Type Judgments

A *type judgement* has the form

 $\Gamma \mid -\exp : \tau$ 

- $\Gamma$  is a typing environment
  - Supplies the types of variables (and function names when function names are not variables)
  - $\Gamma$  is a set of the form  $\{x:\sigma,\ldots\}$
  - For any x at most one  $\sigma$  such that  $(x : \sigma \in \Gamma)$
- exp is a program expression
- τ is a type to be assigned to exp
- |- pronounced "turnstyle", or "entails" (or "satisfies" or, informally, "shows")

18

10/15/19



#### **Axioms - Constants**

 $\Gamma \mid -n : int$  (assuming *n* is an integer constant)

 $\Gamma$  |- true : bool  $\Gamma$  |- false : bool

- These rules are true with any typing environment
- $\Gamma$ , 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  $\sigma$  exits, its unique

Variable axiom:

$$\Gamma \mid -x : \sigma$$
 if  $\Gamma(x) = \sigma$ 

10/15/19 20



# Simple Rules - Arithmetic

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

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

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

$$\frac{\Gamma \mid -e_1 : \tau \qquad \Gamma \mid -e_2 : \tau}{\Gamma \mid -e_1 \sim e_2 : \mathsf{bool}}$$

For the moment, think  $\tau$  is int

10/15/19

21



# Example: $\{x:int\} | -x + 2 = 3 : bool$

What do we need to show first?

$$\{x:int\} \mid -x + 2 = 3 : bool$$

10/15/19 22



#### Example: $\{x:int\} | -x + 2 = 3 : bool$

What do we need for the left side?

$$\frac{\{x: int\} \mid -x+2: int \qquad \{x: int\} \mid -3: int \\ \{x: int\} \mid -x+2=3: bool}{\text{Rel}}$$

10/15/19

23

# Example: $\{x:int\} | -x + 2 = 3 : bool$

How to finish?

$$\frac{\{x: int\} \mid - x: int \ \{x: int\} \mid - 2: int}{\{x: int\} \mid - x + 2: int} AO \{x: int\} \mid - 3: int Rel$$

$$\frac{\{x: int\} \mid - x + 2: int}{\{x: int\} \mid - x + 2 = 3: bool}$$



Example:  $\{x:int\} | -x + 2 = 3 :bool$ 

Complete Proof (type derivation)

$$\frac{\frac{\text{Var}}{\{x:\text{int}\}\mid - x:\text{int}} \frac{\text{Const}}{\{x:\text{int}\}\mid - 2:\text{int}}}{\frac{\{x:\text{int}\}\mid - x + 2:\text{int}}{\{x:\text{int}\}\mid - x + 2 = 3:\text{bool}}} \frac{\text{Const}}{\{x:\text{int}\}\mid - 3:\text{int}}$$

10/15/19



Simple Rules - Booleans

#### Connectives

$$\frac{\Gamma \mid -e_1 : \mathsf{bool} \quad \Gamma \mid -e_2 : \mathsf{bool}}{\Gamma \mid -e_1 \&\& e_2 : \mathsf{bool}}$$

$$\frac{\Gamma \mid -e_1 : \mathsf{bool} \quad \Gamma \mid -e_2 : \mathsf{bool}}{\Gamma \mid -e_1 \mid \mid e_2 : \mathsf{bool}}$$

10/15/19 26



#### Type Variables in Rules

If\_then\_else rule:

$$\frac{\Gamma \mid -e_1 : \mathsf{bool} \quad \Gamma \mid -e_2 : \tau \quad \Gamma \mid -e_3 : \tau}{\Gamma \mid - (\mathsf{if} \ e_1 \ \mathsf{then} \ e_2 \ \mathsf{else} \ e_3) : \tau}$$

- $\tau$  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



25

29

# **Function Application**

Application rule:

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

If you have a function expression e₁ of type τ₁ → τ₂ applied to an argument e₂ of type τ₁, the resulting expression e₁e₂ has type τ₂

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:

$$\frac{\{x:\tau_1\} + \Gamma \mid -e:\tau_2}{\Gamma \mid -\text{fun } x -> e:\tau_1 \to \tau_2}$$

10/15/19

+

#### Fun Examples

$$\frac{\{y : int \} + \Gamma \mid -y + 3 : int}{\Gamma \mid -fun \ y -> y + 3 : int \rightarrow int}$$

$$\frac{\{f: \mathsf{int} \to \mathsf{bool}\} + \Gamma \mid -f \ 2 :: [\mathsf{true}] : \mathsf{bool} \ \mathsf{list}}{\Gamma \mid -(\mathsf{fun} \ f \ -> f \ 2 :: [\mathsf{true}])} \\ : (\mathsf{int} \to \mathsf{bool}) \to \mathsf{bool} \ \mathsf{list}$$

30

10/15/19



# (Monomorphic) Let and Let Rec

let rule:

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

let rec rule:

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

10/15/19 31



Which rule do we apply?

?
$$|- (let rec one = 1 :: one in let x = 2 in fun y -> (x :: y :: one) ) : int  $\rightarrow$  int list$$

10/15/19 32



#### Example

10/15/19 33



#### Proof of 1

Which rule?

{one : int list} |- (1 :: one) : int list

10/15/19 34



#### Proof of 1

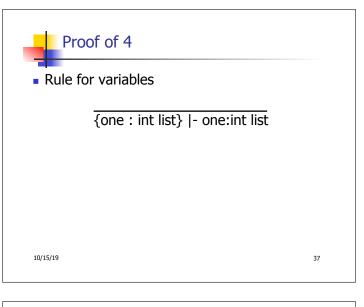
Application

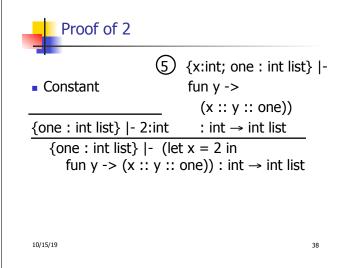
10/15/19 35

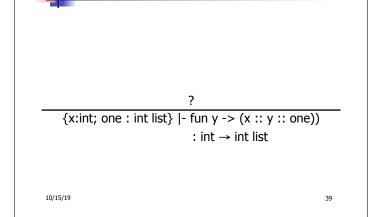


#### Proof of 3

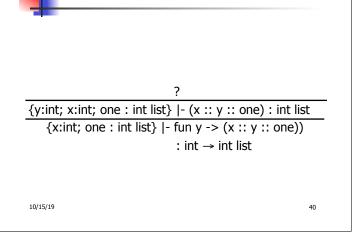
Constants Rule Constants Rule



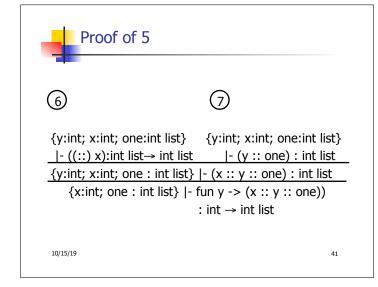


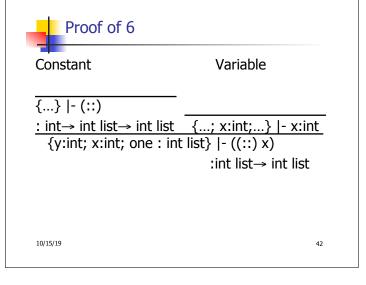


Proof of 5



Proof of 5







#### Proof of 7

Pf of 6 [y/x]

Variable



 $\{\overline{y:int; ...}\}$  |- ((::) y)  $\{\overline{...}; one: int list\}$  |:int list  $\rightarrow$  int list one: int list  $\{\overline{y:int; x:int; one: int list}\}$  |- (y :: one) : 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

$$\frac{\Gamma \mid -e_1 : \alpha \to \beta \quad \Gamma \mid -e_2 : \alpha}{\Gamma \mid -(e_1 e_2) : \beta}$$

10/15/19



## Mea Culpa

- The above system can't handle polymorphism as in OCAML
- No type variables in type language (only metavariable 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

- Monomorpic 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
  - $\forall \alpha_1, \ldots, \alpha_n \cdot \tau$
  - Can think of  $\tau$  as same as  $\forall . \tau$

10/15/19

4

45

47

# Example FreeVars Calculations

- Vars('a -> (int -> 'b) -> 'a) ={'a , 'b}
- FreeVars (All 'b. 'a -> (int -> 'b) -> 'a) =
- {'a , 'b} {'b}= {'a}
- FreeVars {x : All 'b. 'a -> (int -> 'b) -> 'a,
- id: All 'c. 'c -> 'c,
- y: All 'c. 'a -> 'b -> 'c} =
- {'a} U {} U {'a, 'b} = {'a, 'b}

10/15/19



#### 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 FreeVars(τ)
- Free variables of polymorphic type removes variables that are universally quantified
  - FreeVars( $\forall \alpha_1, \dots, \alpha_n \cdot \tau$ ) = FreeVars( $\tau$ )  $\{\alpha_1, \dots, \alpha_n \}$
- FreeVars( $\Gamma$ ) = all FreeVars of types in range of  $\Gamma$

10/15/19



# 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
- Gen( $\tau$ ,  $\Gamma$ ) =  $\forall \alpha_1, ..., \alpha_n$  .  $\tau$  where  $\{\alpha_1, ..., \alpha_n\}$  = freeVars( $\tau$ ) freeVars( $\Gamma$ )

10/15/19 50



#### Polymorphic Typing Rules

- A type judgement has the form
  - $\Gamma$  |- 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

53



# Polymorphic Let and Let Rec

let rule:

$$\frac{\Gamma \mid -e_1 : \tau_1 \mid \{x : \mathsf{Gen}(\tau_1, \Gamma)\} + \Gamma \mid -e_2 : \tau_2}{\Gamma \mid -(\mathsf{let} \mid x = e_1 \mathsf{in} \mid e_2) : \tau_2}$$

let rec rule:

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

10/15/19 52



# Polymorphic Variables (Identifiers)

Variable axiom:

$$\overline{\Gamma \mid - x : \varphi(\tau)}$$
 if  $\Gamma(x) = \forall \alpha_1, \dots, \alpha_n \cdot \tau$ 

- Where  $\varphi$  replaces all occurrences of  $\alpha_1, \ldots, \alpha_n$  by monotypes  $\tau_1, \ldots, \tau_n$
- Note: Monomorphic rule special case:

$$\overline{\Gamma \mid -x : \tau} \quad \text{if } \Gamma(x) = \tau$$

Constants treated same way

10/15/19

4

# Fun Rule Stays the Same

• fun rule:

$$\frac{\{x:\tau_1\} + \Gamma \mid -e:\tau_2}{\Gamma \mid -\text{fun } x -> e:\tau_1 \to \tau_2}$$

- Types τ<sub>1</sub>, τ<sub>2</sub> monomorphic
- Function argument must always be used at same type in function body



## Polymorphic Example

- Assume additional constants:
- hd : $\forall \alpha$ .  $\alpha$  list ->  $\alpha$
- tl:  $\forall \alpha$ .  $\alpha$  list ->  $\alpha$  list
- is\_empty :  $\forall \alpha$ .  $\alpha$  list -> bool
- (::) :  $\forall \alpha. \ \alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}$
- [] : ∀α. α list

10/15/19

55

57



# Polymorphic Example

Show:

2



## Polymorphic Example: Let Rec Rule

• Show: (1) (2) {length: $\alpha$  list -> int} {length: $\forall \alpha$ .  $\alpha$  list -> int}

:  $\alpha$  list -> int length((::) true []) : int

{} |- let rec length =

fun I -> if is\_empty I then 0 else 1 + length (tl I)

in length ((::) 2 []) + length((::) true []) : int

10/15/19



#### Polymorphic Example (1)

Show:

?

 $\begin{aligned} \{ & \text{length:} \alpha \text{ list -> int} \} \mid - \\ & \text{fun I -> if is\_empty I then 0} \\ & \text{else 1 + length (tl I)} \end{aligned}$ 

:  $\alpha$  list -> int

10/15/19 58



## Polymorphic Example (1): Fun Rule

Show: (3)

{length: $\alpha$  list -> int, I:  $\alpha$  list } |-

if is\_empty I then 0

else length (hd l) + length (tl l) : int

{length: $\alpha$  list -> int} |-

fun I -> if is\_empty I then 0

else 1 + length (tl l)

:  $\alpha$  list -> int

10/15/19

50



#### Polymorphic Example (3)

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

7

 $\Gamma$ |- if is\_empty | then 0 else 1 + length (tl |) : int

10/15/19



# Polymorphic Example (3):IfThenElse

- Let  $\Gamma = \{ length : \alpha list -> int, l: \alpha list \}$
- Show
- (4) (5) (6)  $\Gamma$ |- is\_empty |  $\Gamma$ |- 0:int  $\Gamma$ |- 1 + : bool length (tl l) : int  $\Gamma$ |- if is empty | then 0

else 1 + length (tl l) : int

10/15/19 61



# Polymorphic Example (4)

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

.

 $\Gamma$ |- is\_empty | : bool

10/15/19 62



# Polymorphic Example (4):Application

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

?

?

63

 $\frac{\Gamma|\text{- is\_empty}: \alpha \text{ list -> bool} \qquad \Gamma|\text{- I}: \alpha \text{ list}}{\Gamma|\text{- is\_empty I}: \text{bool}}$ 

10/15/19



# Polymorphic Example (4)

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

By Const since  $\alpha$  list -> bool is instance of  $\forall \alpha$ .  $\alpha$  list -> bool ?

 $\frac{\Gamma|\text{- is\_empty}: \alpha \text{ list -> bool} \qquad \Gamma|\text{- I}: \alpha \text{ list}}{\Gamma|\text{- is\_empty I}: \text{bool}}$ 

10/15/19 64



#### Polymorphic Example (4)

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

By Const since  $\alpha$  list -> bool is By Variable instance of  $\forall \alpha$ .  $\alpha$  list -> bool  $\Gamma(I) = \alpha$  list

 $\frac{\Gamma|\text{- is\_empty}: \alpha \text{ list -> bool} \qquad \Gamma|\text{- I}: \alpha \text{ list}}{\Gamma|\text{- is\_empty I}: \text{bool}}$ 

■ This finishes (4)

10/15/19 65



## Polymorphic Example (5):Const

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

By Const Rule

 $\Gamma$ |- 0:int

10/15/19



# Polymorphic Example (6):Arith Op

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

By Variable  $\Gamma$  |- length (7)

By Const :  $\alpha$  list -> int  $\Gamma$ |- (tl l) :  $\alpha$  list  $\Gamma$ |- 1:int  $\Gamma$ |- length (tl l) : int

 $\Gamma$ |-1 + length (tl l) : int

10/15/19

--



# Polymorphic Example (7):App Rule

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

By Const

By Variable

 $\Gamma$ |- tl :  $\alpha$  list ->  $\alpha$  list

 $\Gamma$ |-|:  $\alpha$  list

 $\Gamma$ |- (tl l) :  $\alpha$  list

By Const since  $\alpha$  list ->  $\alpha$  list is instance of  $\forall \alpha$ .  $\alpha$  list ->  $\alpha$  list

10/15/19 68



# Polymorphic Example: (2) by ArithOp

(9)

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

(8)

Γ' |-

length ((::) 2 []) :int length((::) true []) : int

{length: $\forall \alpha$ .  $\alpha$  list -> int}

|- length ((::) 2 []) + length((::) true []) : int

10/15/19 69



# Polymorphic Example: (8)AppRule

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

 $\Gamma'$  |- length : int list ->int  $\Gamma'$  |- ((::)2 []):int list  $\Gamma'$  |- length ((::) 2 []) :int

10/15/19 70



# Polymorphic Example: (8)AppRule

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

By Var since int list -> int is instance of

 $\forall \alpha. \ \alpha \ \text{list} \rightarrow \text{int} \ (\text{by } \alpha \rightarrow \text{int})$ 

(10)

 $\Gamma'$  |- length : int list ->int  $\Gamma'$  |- ((::)2 []):int list  $\Gamma'$  |- length ((::) 2 []) :int

10/15/19

71



# Polymorphic Example: (10)AppRule

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

(11)

 $\frac{\Gamma' \mid -((::) \ 2) : \text{int list } -> \text{int list} \quad \Gamma' \mid -[] : \text{int list}}{\Gamma' \mid -((::) \ 2[]) : \text{int list}}$ 

10/15/19



# Polymorphic Example: (11)AppRule

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

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

10/15/19 73



# Polymorphic Example: (9)AppRule

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

 $\Gamma'$  |-  $\Gamma'$  |- length:bool list ->int  $\Gamma'$  |- length ((::) true []):bool list  $\Gamma'$  |- length ((::) true []) :int

10/15/19 74



# Polymorphic Example: (9)AppRule

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

By Var since bool list -> int  $\$ is instance of  $\forall \alpha. \ \alpha \$ list -> int

 $\begin{array}{c|cccc} & & & & & \\ \hline \Gamma' & | - & & & & \\ \hline \text{length:bool list ->int} & & & & \\ \hline \Gamma' & | - & & & \\ \hline \Gamma' & | - & & \\ \hline \end{array}$ 

10/15/19 75



## Polymorphic Example: (12)AppRule

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

(13)  $\frac{\Gamma' \mid -((::)\text{true}):\text{bool list} \quad \overline{\Gamma' \mid -\ []:\text{bool list}}}{\Gamma' \mid -\ ((::)\ \text{true}\ []):\text{bool list}}$ 

10/15/19 76



# Polymorphic Example: (13)AppRule

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

By Const since

bool ->bool list ->bool list is instance

(::):bool ->bool list ->bool list true : bool

 $\Gamma'$  |- ((::) true) : bool list -> bool list