



# Program Verification

## Part 5 – Reasoning about Functional Programs

René Thiemann

Department of Computer Science

## Inference Rules for the Standard Model

Inference Rules for the Standard Model

Inference Rules for the Standard Model

### Plan

- only consider well-defined functional programs, so that standard model is well-defined
- aim
  - derive theorems and inference rules which are valid in the standard model
  - these can be used to formally reason about functional programs as on slide 1/18 where associativity of `append` was proven
- examples
  - reasoning about constructors
    - $\forall x, y. \text{Succ}(x) =_{\text{Nat}} \text{Succ}(y) \iff x =_{\text{Nat}} y$
    - $\forall x. \neg \text{Succ}(x) =_{\text{Nat}} \text{Zero}$
  - getting defining equations of functional programs as theorems
    - $\forall x, xs, ys. \text{append}(\text{Cons}(x, xs), ys) =_{\text{List}} \text{Cons}(x, \text{append}(xs, ys))$
  - induction schemes
    - $\frac{\varphi(\text{Zero}) \quad \forall x. \varphi(x) \longrightarrow \varphi(\text{Succ}(x))}{\forall x. \varphi(x)}$

### Notation – The Normal Form

- when speaking about  $\leftrightarrow$ , we always consider some fixed well-defined functional program
- since every term has a unique normal form w.r.t.  $\leftrightarrow$ , we can define a function  $\Downarrow : \mathcal{T}(\Sigma, \mathcal{V})_{\tau} \rightarrow \mathcal{T}(\Sigma, \mathcal{V})_{\tau}$  which returns this normal form and write it in postfix notation:

$t \Downarrow :=$  the unique normal of  $t$  w.r.t.  $\leftrightarrow$

- using  $\Downarrow$ , the meaning of symbols in the standard model can concisely be written as

$$F^{\mathcal{M}}(t_1, \dots, t_n) = F(t_1, \dots, t_n) \Downarrow$$

- proof
  - universe of type  $\tau$  is  $\mathcal{T}(\mathcal{C})_{\tau}$ , so  $t \in \mathcal{T}(\mathcal{C})_{\tau}$  implies  $t \in \text{NF}(\leftrightarrow)$
  - if  $F \in \mathcal{C}$ , then  $F^{\mathcal{M}}(t_1, \dots, t_n) \stackrel{\text{def}}{=} F(t_1, \dots, t_n) = F(t_1, \dots, t_n) \Downarrow$
  - if  $F \in \mathcal{D}$ , then  $F^{\mathcal{M}}(t_1, \dots, t_n) \stackrel{\text{def}}{=} F(t_1, \dots, t_n) \Downarrow$

## The Substitution Lemma

- there are two possibilities to plug in objects into variables
  - as assignment:  $\alpha : \mathcal{V}_\tau \rightarrow \mathcal{A}_\tau$   
result of  $\llbracket t \rrbracket_\alpha$  is an element of  $\mathcal{A}_\tau$
  - as substitution:  $\sigma : \mathcal{V}_\tau \rightarrow \mathcal{T}(\Sigma, \mathcal{V})_\tau$   
result of  $t\sigma$  is an element of  $\mathcal{T}(\Sigma, \mathcal{V})_\tau$
- **substitution lemma**: substitutions can be moved into assignment:

$$\llbracket t\sigma \rrbracket_\alpha = \llbracket t \rrbracket_\beta$$

where  $\beta(x) := \llbracket \sigma(x) \rrbracket_\alpha$

- proof by structural induction on  $t$ 
  - $\llbracket x \rrbracket_\alpha = \llbracket \sigma(x) \rrbracket_\alpha = \beta(x) = \llbracket x \rrbracket_\beta$

$$\begin{aligned} \llbracket F(t_1, \dots, t_n)\sigma \rrbracket_\alpha &= \llbracket F(t_1\sigma, \dots, t_n\sigma) \rrbracket_\alpha \\ &= F^{\mathcal{M}}(\llbracket t_1\sigma \rrbracket_\alpha, \dots, \llbracket t_n\sigma \rrbracket_\alpha) \\ &\stackrel{IH}{=} F^{\mathcal{M}}(\llbracket t_1 \rrbracket_\beta, \dots, \llbracket t_n \rrbracket_\beta) \\ &= \llbracket F(t_1, \dots, t_n) \rrbracket_\beta \end{aligned}$$

## Reverse Substitution Lemma in the Standard Model

- the substitution lemma holds independently of the model
- in case of the standard model, we have the special condition that  $\mathcal{A}_\tau = \mathcal{T}(\mathcal{C})_\tau$ , so
  - the universes consist of terms
  - hence, each **assignment**  $\alpha : \mathcal{V}_\tau \rightarrow \mathcal{T}(\mathcal{C})_\tau$  is a special kind of **substitution** (constructor ground substitution)
- consequence: possibility to encode assignment as substitution
- **reverse substitution lemma**:

$$\llbracket t \rrbracket_\alpha = t\alpha \downarrow$$

- proof by structural induction on  $t$ 
  - $\llbracket x \rrbracket_\alpha = \alpha(x) \stackrel{(*)}{=} \alpha(x) \downarrow = x\alpha \downarrow$  where  $(*)$  holds, since  $\alpha(x) \in \mathcal{T}(\mathcal{C})$

$$\begin{aligned} \llbracket F(t_1, \dots, t_n) \rrbracket_\alpha &= F^{\mathcal{M}}(\llbracket t_1 \rrbracket_\alpha, \dots, \llbracket t_n \rrbracket_\alpha) \\ &\stackrel{IH}{=} F^{\mathcal{M}}(t_1\alpha \downarrow, \dots, t_n\alpha \downarrow) = F(t_1\alpha \downarrow, \dots, t_n\alpha \downarrow) \downarrow \\ &\stackrel{(conf.)}{=} F(t_1\alpha, \dots, t_n\alpha) \downarrow = F(t_1, \dots, t_n)\alpha \downarrow \end{aligned}$$

## Defining Equations are Theorems in Standard Model

- notation:  $\vec{\forall} \varphi$  means that universal quantification ranges over all free variables that occur in  $\varphi$
- example: if  $\varphi$  is  $\text{append}(\text{Cons}(x, xs), ys) =_{\text{List}} \text{Cons}(x, \text{append}(xs, ys))$  then  $\vec{\forall} \varphi$  is

$$\forall x, xs, ys. \text{append}(\text{Cons}(x, xs), ys) =_{\text{List}} \text{Cons}(x, \text{append}(xs, ys))$$

- theorem: if  $\ell = r$  is defining equation of program (of type  $\tau$ ), then

$$\mathcal{M} \models \vec{\forall} \ell =_\tau r$$

- consequence: conversion of well-defined functional programs into equations is now possible, cf. previous problem on slide 1/20
- proof of theorem
  - by definition of  $\models$  and  $=_\tau^{\mathcal{M}}$  we have to show  $\llbracket \ell \rrbracket_\alpha = \llbracket r \rrbracket_\alpha$  for all  $\alpha$
  - via reverse substitution lemma this is equivalent to  $\ell\alpha \downarrow = r\alpha \downarrow$
  - easily follows from confluence, since  $\ell\alpha \leftrightarrow r\alpha$

## Axiomatic Reasoning

- previous slide already provides us with some theorems that are satisfied in standard model
- **axiomatic reasoning**:  
take those theorems as axioms to show property  $\varphi$
- added axioms are theorems of standard model, so they are **consistent**
- example  $AX = \{\vec{\forall} \ell =_\tau r \mid \ell = r \text{ is defining equation}\}$
- **show  $AX \models \varphi$  using first-order reasoning in order to prove  $\mathcal{M} \models \varphi$**  (and forget standard model  $\mathcal{M}$  during the reasoning!)
- question: is it possible to prove every property  $\varphi$  in this way for which  $\mathcal{M} \models \varphi$  holds?
- answer for above example is “no”
  - reason: there are models different than the standard model in which all axioms of  $AX$  are satisfied, but where  $\varphi$  does not hold!
  - example on next slide

## Axiomatic Reasoning – Problematic Model

- consider addition program, then example  $AX$  consists of two axioms

$$\begin{aligned} \forall y. \text{plus}(\text{Zero}, y) &=_{\text{Nat}} y \\ \forall x, y. \text{plus}(\text{Succ}(x), y) &=_{\text{Nat}} \text{Succ}(\text{plus}(x, y)) \end{aligned}$$

- we want to prove associativity of **plus**, so let  $\varphi$  be

$$\forall x, y, z. \text{plus}(\text{plus}(x, y), z) =_{\text{Nat}} \text{plus}(x, \text{plus}(y, z))$$

- consider the following model  $\mathcal{M}'$

- $\mathcal{A}_{\text{Nat}} = \mathbb{N} \cup \{x + \frac{1}{2} \mid x \in \mathbb{Z}\} = \{\dots, -1\frac{1}{2}, -\frac{1}{2}, 0, \frac{1}{2}, 1, 1\frac{1}{2}, 2, 2\frac{1}{2}, \dots\}$
- $\text{Zero}^{\mathcal{M}'} = 0$
- $\text{Succ}^{\mathcal{M}'}(n) = n + 1$
- $\text{plus}^{\mathcal{M}'}(n, m) = \begin{cases} n + m, & \text{if } n \in \mathbb{N} \text{ or } m \in \mathbb{N} \\ n - m + \frac{1}{2}, & \text{otherwise} \end{cases}$
- $=_{\text{Nat}}^{\mathcal{M}'} = \{(n, n) \mid n \in \mathcal{A}_{\text{Nat}}\}$
- $\mathcal{M}' \models \bigwedge AX$ , but  $\mathcal{M}' \not\models \varphi$ : consider  $\alpha(x) = \frac{19}{2}, \alpha(y) = \frac{9}{2}, \alpha(z) = \frac{7}{2}$
- problem: values in  $\alpha$  do not correspond to constructor ground terms

## Gödel's Incompleteness Theorem

- taking  $AX$  as set of defining equations does not suffice to deduce all valid theorems of standard model
- obvious approach: add more theorems to axioms  $AX$  (theorems about  $=_{\tau}$ , induction rules, ...)
- question: is it then possible to deduce all valid theorems of standard model?
- negative answer by **Gödel's First Incompleteness Theorem**
- theorem**: consider a well-defined functional program that includes addition and multiplication of natural numbers; let  $AX$  be a decidable set of valid theorems in the standard model; then **there is a formula  $\varphi$  such that  $\mathcal{M} \models \varphi$ , but  $AX \not\models \varphi$**
- note: adding  $\varphi$  to  $AX$  does not fix the problem, since then there is another formula  $\varphi'$  such that  $\mathcal{M} \models \varphi'$  and  $AX \cup \{\varphi\} \not\models \varphi'$
- consequence: **"proving  $\varphi$  via  $AX \models \varphi$ " is sound, but never complete**
- upcoming: add more axioms than just defining equations, so that still several proofs are possible

## Axioms about Equality

- we define decomposition theorems and disjointness theorems in the form of logical equivalences
- for each  $c : \tau_1 \times \dots \times \tau_n \rightarrow \tau \in \mathcal{C}$  we define its **decomposition theorem** as

$$\vec{\forall} c(x_1, \dots, x_n) =_{\tau} c(y_1, \dots, y_n) \longleftrightarrow x_1 =_{\tau_1} y_1 \wedge \dots \wedge x_n =_{\tau_n} y_n$$

and for all  $d : \tau_1' \times \dots \times \tau_k' \rightarrow \tau \in \mathcal{C}$  with  $c \neq d$  we define the **disjointness theorem** as

$$\vec{\forall} c(x_1, \dots, x_n) =_{\tau} d(y_1, \dots, y_k) \longleftrightarrow \text{false}$$

- proof of validity of decomposition theorem:

$$\begin{aligned} &\mathcal{M} \models_{\alpha} c(x_1, \dots, x_n) =_{\tau} c(y_1, \dots, y_n) \\ \text{iff } &c(\alpha(x_1), \dots, \alpha(x_n)) = c(\alpha(y_1), \dots, \alpha(y_n)) \\ \text{iff } &\alpha(x_1) = \alpha(y_1) \text{ and } \dots \text{ and } \alpha(x_n) = \alpha(y_n) \\ \text{iff } &\mathcal{M} \models_{\alpha} x_1 =_{\tau_1} y_1 \text{ and } \dots \text{ and } \mathcal{M} \models_{\alpha} x_n =_{\tau_n} y_n \\ \text{iff } &\mathcal{M} \models_{\alpha} x_1 =_{\tau_1} y_1 \wedge \dots \wedge x_n =_{\tau_n} y_n \end{aligned}$$

## Axioms about Equality – Example

- for the datatypes of natural numbers and lists we get the following axioms

$$\begin{aligned} \text{Zero} &=_{\text{Nat}} \text{Zero} \longleftrightarrow \text{true} \\ \forall x, y. \text{Succ}(x) &=_{\text{Nat}} \text{Succ}(y) \longleftrightarrow x =_{\text{Nat}} y \\ \text{Nil} &=_{\text{List}} \text{Nil} \longleftrightarrow \text{true} \\ \forall x, xs, y, ys. \text{Cons}(x, xs) &=_{\text{List}} \text{Cons}(y, ys) \longleftrightarrow x =_{\text{Nat}} y \wedge xs =_{\text{List}} ys \end{aligned}$$

$$\begin{aligned} \forall y. \text{Zero} &=_{\text{Nat}} \text{Succ}(y) \longleftrightarrow \text{false} \\ \forall x. \text{Succ}(x) &=_{\text{Nat}} \text{Zero} \longleftrightarrow \text{false} \\ \forall y, ys. \text{Nil} &=_{\text{List}} \text{Cons}(y, ys) \longleftrightarrow \text{false} \\ \forall x, xs. \text{Cons}(x, xs) &=_{\text{List}} \text{Nil} \longleftrightarrow \text{false} \end{aligned}$$

## Induction Theorems

- current axioms are not even strong enough to prove simple theorems, e.g.,  
 $\forall x. \text{plus}(x, \text{Zero}) =_{\text{Nat}} x$
- problem: proofs by induction are not yet covered in axioms
- since the principle of **induction cannot be defined** in general in a **single first-order formula**, we will add infinitely many induction theorems to the set of axioms, one for each property
- not a problem, since set of axioms stays decidable, i.e., one can see whether some tentative formula is an element of the axiom set or not
- example: induction over natural numbers

- formula below is general, but not first-order as it quantifies over  $\varphi$

$$\forall \varphi(x : \text{Nat}). \varphi(\text{Zero}) \longrightarrow (\forall x. \varphi(x) \longrightarrow \varphi(\text{Succ}(x))) \longrightarrow \forall x. \varphi(x)$$

- quantification can be done on meta-level instead:  
let  $\varphi$  be an arbitrary formula with a free variable of type **Nat**; then

$$\varphi(\text{Zero}) \longrightarrow (\forall x. \varphi(x) \longrightarrow \varphi(\text{Succ}(x))) \longrightarrow \forall x. \varphi(x)$$

is a valid theorem; quantifying over  $\varphi$  results in **induction scheme**

## Induction Theorems – Example Instances

- induction scheme

$$\varphi(\text{Zero}) \longrightarrow (\forall x. \varphi(x) \longrightarrow \varphi(\text{Succ}(x))) \longrightarrow \forall x. \varphi(x)$$

- example: right-neutral element:  $\varphi(x) := \text{plus}(x, \text{Zero}) =_{\text{Nat}} x$

$$\text{plus}(\text{Zero}, \text{Zero}) =_{\text{Nat}} \text{Zero}$$

$$\longrightarrow (\forall x. \text{plus}(x, \text{Zero}) =_{\text{Nat}} x \longrightarrow \text{plus}(\text{Succ}(x), \text{Zero}) =_{\text{Nat}} \text{Succ}(x))$$

$$\longrightarrow \forall x. \text{plus}(x, \text{Zero}) =_{\text{Nat}} x$$

- example with **quantifiers** and **free variables**:

$$\varphi(x) := \forall y. \text{plus}(\text{plus}(x, y), z) =_{\text{Nat}} \text{plus}(x, \text{plus}(y, z))$$

$$\forall y. \text{plus}(\text{plus}(\text{Zero}, y), z) =_{\text{Nat}} \text{plus}(\text{Zero}, \text{plus}(y, z))$$

$$\longrightarrow (\forall x. (\forall y. \text{plus}(\text{plus}(x, y), z) =_{\text{Nat}} \text{plus}(x, \text{plus}(y, z))))$$

$$\longrightarrow (\forall y. \text{plus}(\text{plus}(\text{Succ}(x), y), z) =_{\text{Nat}} \text{plus}(\text{Succ}(x), \text{plus}(y, z))))$$

$$\longrightarrow \forall x. \forall y. \text{plus}(\text{plus}(x, y), z) =_{\text{Nat}} \text{plus}(x, \text{plus}(y, z))$$

## Preparing Induction Theorems – Substitutions in Formulas

- current situation
  - substitutions are functions of type  $\mathcal{V} \rightarrow \mathcal{T}(\Sigma, \mathcal{V})$
  - lifted to functions of type  $\mathcal{T}(\Sigma, \mathcal{V}) \rightarrow \mathcal{T}(\Sigma, \mathcal{V})$ , cf. [slide 3/22](#)
  - substitution of variables of formulas is not yet defined, but is required for induction formulas, cf. notation  $\varphi(x) \longrightarrow \varphi(\text{Succ}(x))$  on previous slide
- formal definition of **applying a substitution  $\sigma$  to formulas**
  - true  $\sigma = \text{true}$
  - $(\neg \varphi)\sigma = \neg(\varphi\sigma)$
  - $(\varphi \wedge \psi)\sigma = \varphi\sigma \wedge \psi\sigma$
  - $P(t_1, \dots, t_n)\sigma = P(t_1\sigma, \dots, t_n\sigma)$
  - $(\forall x. \varphi)\sigma = \forall x. (\varphi\sigma)$  if  $x$  does not occur in  $\sigma$ , i.e.,  $\sigma(x) = x$  and  $x \notin \text{Vars}(\sigma(y))$  for all  $y \neq x$
  - $(\forall x. \varphi)\sigma = (\forall y. \varphi[x/y])\sigma$  if  $x$  occurs in  $\sigma$  where
    - $y$  is a fresh variable, i.e.,  $\sigma(y) = y$ ,  $y \notin \text{Vars}(\sigma(z))$  for all  $z \neq y$ , and  $y$  is not a free variable of  $\varphi$
    - $[x/y]$  is the substitution which just replaces  $x$  by  $y$
    - effect is  **$\alpha$ -renaming**: just rename universally quantified variable before substitution to **avoid variable capture**

## Examples

- substitution of formulas

$$\bullet (\forall x. \varphi)\sigma = \forall x. (\varphi\sigma)$$

$$\bullet (\forall x. \varphi)\sigma = (\forall y. \varphi[x/y])\sigma$$

if  $x$  does not occur in  $\sigma$   
if  $x$  occurs in  $\sigma$  where  $y$  is fresh

- example substitution applications

$$\bullet \varphi := \forall x. \neg x =_{\text{Nat}} y$$

$$\bullet \varphi[y/\text{Zero}] = \forall x. \neg x =_{\text{Nat}} \text{Zero}$$

$$\bullet \varphi[y/\text{Succ}(z)] = \forall x. \neg x =_{\text{Nat}} \text{Succ}(z)$$

$$\bullet \varphi[y/\text{Succ}(x)] = \forall z. \neg z =_{\text{Nat}} \text{Succ}(x)$$

without renaming meaning will change:  $\forall x. \neg x =_{\text{Nat}} \text{Succ}(x)$

$$\bullet \varphi[x/\text{Succ}(y)] = \forall z. \neg z =_{\text{Nat}} y$$

without renaming meaning will change:  $\forall x. \neg \text{Succ}(y) =_{\text{Nat}} y$

- example theorems involving substitutions

$$\varphi[x/\text{Zero}] \longrightarrow (\forall y. \varphi[x/y] \longrightarrow \varphi[x/\text{Succ}(y)]) \longrightarrow \forall x. \varphi$$

## Substitution Lemma for Formulas

- example induction formula

$$\varphi[x/\text{Zero}] \longrightarrow (\forall y. \varphi[x/y] \longrightarrow \varphi[x/\text{Succ}(y)]) \longrightarrow \forall x. \varphi$$

- proving validity of this formula (in standard model) requires another substitution lemma about substitutions in formulas
- lemma:  $\mathcal{M} \models_{\alpha} \varphi\sigma$  iff  $\mathcal{M} \models_{\beta} \varphi$  where  $\beta(x) := \llbracket \sigma(x) \rrbracket_{\alpha}$
- proof by structural induction on  $\varphi$  for arbitrary  $\alpha$  and  $\sigma$ 
  - $\mathcal{M} \models_{\alpha} P(t_1, \dots, t_n)\sigma$   
iff  $\mathcal{M} \models_{\alpha} P(t_1\sigma, \dots, t_n\sigma)$   
iff  $(\llbracket t_1\sigma \rrbracket_{\alpha}, \dots, \llbracket t_n\sigma \rrbracket_{\alpha}) \in P^{\mathcal{M}}$   
iff  $(\llbracket t_1 \rrbracket_{\beta}, \dots, \llbracket t_n \rrbracket_{\beta}) \in P^{\mathcal{M}}$   
iff  $\mathcal{M} \models_{\beta} P(t_1, \dots, t_n)$   
where we use the substitution lemma of slide 5 to conclude  $\llbracket t_i\sigma \rrbracket_{\alpha} = \llbracket t_i \rrbracket_{\beta}$
  - $\mathcal{M} \models_{\alpha} (\neg\varphi)\sigma$  iff  $\mathcal{M} \models_{\alpha} \neg(\varphi\sigma)$  iff  $\mathcal{M} \not\models_{\alpha} \varphi\sigma$   
iff  $\mathcal{M} \not\models_{\beta} \varphi$  (by IH) iff  $\mathcal{M} \models_{\beta} \neg\varphi$
  - cases “true” and conjunction are proved in same way as negation

## Substitution Lemma for Formulas – Proof Continued

- lemma:  $\mathcal{M} \models_{\alpha} \varphi\sigma$  iff  $\mathcal{M} \models_{\beta} \varphi$  where  $\beta(x) := \llbracket \sigma(x) \rrbracket_{\alpha}$
- proof by structural induction on  $\varphi$  for arbitrary  $\alpha$  and  $\sigma$ 
  - for quantification we here only consider the more complex case where renaming is required
  - $\mathcal{M} \models_{\alpha} (\forall x. \varphi)\sigma$   
iff  $\mathcal{M} \models_{\alpha} (\forall y. \varphi[x/y])\sigma$  for fresh  $y$   
iff  $\mathcal{M} \models_{\alpha} \forall y. (\varphi[x/y])\sigma$   
iff  $\mathcal{M} \models_{\alpha[y:=a]} \varphi[x/y]\sigma$  for all  $a \in \mathcal{A}$   
iff  $\mathcal{M} \models_{\beta'} \varphi$  for all  $a \in \mathcal{A}$  where  $\beta'(z) := \llbracket ([x/y]\sigma)(z) \rrbracket_{\alpha[y:=a]}$  (by IH)  
iff  $\mathcal{M} \models_{\beta[x:=a]} \varphi$  for all  $a \in \mathcal{A}$  only non-automatic step  
iff  $\mathcal{M} \models_{\beta} \forall x. \varphi$
  - equivalence of  $\beta'$  and  $\beta[x := a]$  on variables of  $\varphi$ 
    - $\beta'(x) = \llbracket ([x/y]\sigma)(x) \rrbracket_{\alpha[y:=a]} = \llbracket \sigma(y) \rrbracket_{\alpha[y:=a]} = \llbracket y \rrbracket_{\alpha[y:=a]} = a$  and  $\beta[x := a](x) = a$
    - $z$  is variable of  $\varphi$ ,  $z \neq x$ :  
by freshness condition conclude  $z \neq y$  and  $y \notin \text{Vars}(\sigma(z))$ ; hence  
 $\beta'(z) = \llbracket ([x/y]\sigma)(z) \rrbracket_{\alpha[y:=a]} = \llbracket \sigma(z) \rrbracket_{\alpha[y:=a]} = \llbracket \sigma(z) \rrbracket_{\alpha}$  and  
 $\beta[x := a](z) = \beta(z) = \llbracket \sigma(z) \rrbracket_{\alpha}$

## Substitution Lemma in Standard Model

- substitution lemma:  $\mathcal{M} \models_{\alpha} \varphi\sigma$  iff  $\mathcal{M} \models_{\beta} \varphi$  where  $\beta(x) := \llbracket \sigma(x) \rrbracket_{\alpha}$
- lemma is valid for all models
- in standard model, substitution lemma permits to characterize universal quantification by substitutions, similar to reverse substitution lemma on slide 6
- lemma: let  $x : \tau \in \mathcal{V}$ , let  $\mathcal{M}$  be the **standard model**
  - $\mathcal{M} \models_{\alpha[x:=t]} \varphi$  iff  $\mathcal{M} \models_{\alpha} \varphi[x/t]$
  - $\mathcal{M} \models_{\alpha} \forall x. \varphi$  iff  $\mathcal{M} \models_{\alpha} \varphi[x/t]$  for all  $t \in \mathcal{T}(\mathcal{C})_{\tau}$
- proof
  - first note that the usage of  $\alpha[x := t]$  implies  $t \in \mathcal{A}_{\tau} = \mathcal{T}(\mathcal{C})_{\tau}$ ;  
by the substitution lemma obtain  
 $\mathcal{M} \models_{\alpha} \varphi[x/t]$   
iff  $\mathcal{M} \models_{\beta} \varphi$  for  $\beta(z) = \llbracket [x/t](z) \rrbracket_{\alpha} = \alpha[x := \llbracket t \rrbracket_{\alpha}](z)$   
iff  $\mathcal{M} \models_{\alpha[x:=t]} \varphi$  ( $\llbracket t \rrbracket_{\alpha} = t$ , since  $t \in \mathcal{T}(\mathcal{C})$ )
  - immediate by part 1 of lemma

## Substitution Lemma and Induction Formulas

- substitution lemma (SL) is crucial result to **lift** structural **induction rule** of universe  $\mathcal{T}(\mathcal{C})_{\tau}$  to a structural **induction formula**
- example: structural induction formula  $\psi$  for lists with fresh  $x, xs$

$$\psi := \underbrace{\varphi[ys/\text{Nil}]}_1 \longrightarrow \underbrace{(\forall x, xs. \varphi[ys/xs]) \longrightarrow \varphi[ys/\text{Cons}(x, xs)]}_2 \longrightarrow \forall ys. \varphi$$

- proof of  $\mathcal{M} \models_{\alpha} \psi$ :  
assume premises 1 ( $\mathcal{M} \models_{\alpha} \varphi[ys/\text{Nil}]$ ) and 2 and show  $\mathcal{M} \models_{\alpha} \forall ys. \varphi$ :  
by SL the latter is equivalent to “ $\mathcal{M} \models_{\alpha} \varphi[ys/\ell]$  for all  $\ell \in \mathcal{T}(\mathcal{C})_{\text{List}}$ ”;  
prove this statement by structural induction on lists
  - Nil**: showing  $\mathcal{M} \models_{\alpha} \varphi[ys/\text{Nil}]$  is easy: it is exactly premise 1
  - Cons**( $n, \ell$ ): use SL on premise 2 to conclude

$$\mathcal{M} \models_{\alpha} (\varphi[ys/xs] \longrightarrow \varphi[ys/\text{Cons}(x, xs)])[x/n, xs/\ell]$$

hence

$$\mathcal{M} \models_{\alpha} \varphi[ys/\ell] \longrightarrow \varphi[ys/\text{Cons}(n, \ell)]$$

and with IH  $\mathcal{M} \models_{\alpha} \varphi[ys/\ell]$  conclude  $\mathcal{M} \models_{\alpha} \varphi[ys/\text{Cons}(n, \ell)]$



## Example: Attempt to Prove Associativity of Append via AX

- task: prove associativity of append via **natural deduction** and **AX**
- define  $\varphi := \text{append}(\text{append}(xs, ys), zs) =_{\text{List}} \text{append}(xs, \text{append}(ys, zs))$ 
  1. show  $\forall xs, ys, zs. \varphi$
  2.  $\forall$ -intro: show  $\varphi$  where now  $xs, ys, zs$  are fresh variables
  3. to this end prove intermediate goal:  $\forall xs. \varphi$
  4. applying induction axiom  $\varphi[xs/\text{Nil}] \longrightarrow (\forall u, us. \varphi[xs/us] \longrightarrow \varphi[xs/\text{Cons}(u, us)]) \longrightarrow \forall xs. \varphi$   
in combination with modus ponens yields two subgoals, one of them is  $\varphi[xs/\text{Nil}]$ , i.e.,  
 $\text{append}(\text{append}(\text{Nil}, ys), zs) =_{\text{List}} \text{append}(\text{Nil}, \text{append}(ys, zs))$
  5. use axiom  $\forall ys. \text{append}(\text{Nil}, ys) =_{\text{List}} ys$
  6.  $\forall$ -elim:  $\text{append}(\text{Nil}, \text{append}(ys, zs)) =_{\text{List}} \text{append}(ys, zs)$
  7. at this point we would like to **simplify** the rhs in the goal to obtain obligation  
 $\text{append}(\text{append}(\text{Nil}, ys), zs) =_{\text{List}} \text{append}(ys, zs)$
  8. this is not possible at this point: there are missing axioms
    - $=_{\text{List}}$  is an equivalence relation
    - $=_{\text{List}}$  is a congruence; required to simplify the lhs  $\text{append}(\cdot, zs)$  at  $\cdot$
    - ...
- next step: **reconsider** the **reasoning engine** and the available **axioms**

## Equational Reasoning and Induction

## Reasoning about Functional Programs: Current State

- extract set of axioms  $AX$  that are satisfied in standard model  $\mathcal{M}$
- for proving property  $\mathcal{M} \models \varphi$  it suffices to show  $AX \models \varphi$
- problems: **reasoning via natural deduction quite cumbersome**
  - explicit introduction and elimination of quantifiers
  - no direct support for equational reasoning
- aim: equational reasoning
  - implicit transitivity reasoning: from  $a =_{\tau} b =_{\tau} c =_{\tau} d$  conclude  $a =_{\tau} d$
  - equational reasoning in contexts: from  $a =_{\tau} b$  conclude  $f(a) =_{\tau'} f(b)$
- in general: want some calculus  $\vdash$  such that  $\vdash \varphi$  implies  $\mathcal{M} \models \varphi$

## Equational Reasoning with Universally Quantified Formulas

- for now let us restrict to universally quantified formulas
- we can formulate properties like
  - $\forall xs. \text{reverse}(\text{reverse}(xs)) =_{\text{List}} xs$
  - $\forall xs, ys. \text{reverse}(\text{append}(xs, ys)) =_{\text{List}} \text{append}(\text{reverse}(ys), \text{reverse}(xs))$
  - $\forall x, y. \text{plus}(x, y) =_{\text{Nat}} \text{plus}(y, x)$
- but not
  - $\forall x. \exists y. \text{greater}(y, x) =_{\text{Bool}} \text{True}$
- universally quantified axioms
  - equations of defined symbols
    - $\forall y. \text{plus}(\text{Zero}, y) =_{\text{Nat}} y$
    - $\forall x, y. \text{plus}(\text{Succ}(x), y) =_{\text{Nat}} \text{Succ}(\text{plus}(x, y))$
    - ...
  - axioms about equality of constructors
    - $\forall x, y. \text{Succ}(x) =_{\text{Nat}} \text{Succ}(y) \longleftrightarrow x =_{\text{Nat}} y$
    - $\forall x. \text{Succ}(x) =_{\text{Nat}} \text{Zero} \longleftrightarrow \text{false}$
    - ...
  - but not: structural induction formulas
    - $\varphi[y/\text{Zero}] \longrightarrow (\forall x. \varphi[y/x] \longrightarrow \varphi[y/\text{Succ}(x)]) \longrightarrow \forall y. \varphi$

## Equational Reasoning in Formulas

- so far:  $\hookrightarrow_{\mathcal{E}}$  replaces terms by terms using **equations  $\mathcal{E}$  of program**
- upcoming:  $\rightsquigarrow$  to simplify formulas using **universally quantified axioms**
- formal definition: let  $AX$  be a set of axioms; then  $\rightsquigarrow_{AX}$  is defined as

$$\frac{\text{true} \wedge \varphi \rightsquigarrow_{AX} \varphi}{\neg \text{false} \rightsquigarrow_{AX} \text{true}} \quad \frac{\varphi \wedge \text{true} \rightsquigarrow_{AX} \varphi}{\neg \text{true} \rightsquigarrow_{AX} \text{false}} \quad \frac{\text{false} \wedge \varphi \rightsquigarrow_{AX} \text{false}}{\text{true} \rightsquigarrow_{AX} \text{true}}$$

$$\frac{\vec{\forall} \ell =_{\tau} r \in AX \quad s \hookrightarrow_{\{\ell=r\}} s'}{s =_{\tau} t \rightsquigarrow_{AX} s' =_{\tau} t} \quad \frac{\vec{\forall} \ell =_{\tau} r \in AX \quad t \hookrightarrow_{\{\ell=r\}} t'}{s =_{\tau} t \rightsquigarrow_{AX} s =_{\tau} t'}$$

$$\frac{\vec{\forall} (\ell =_{\tau} r \longleftrightarrow \varphi) \in AX}{l\sigma =_{\tau} r\sigma \rightsquigarrow_{AX} \varphi\sigma} \quad \frac{}{t =_{\tau} t \rightsquigarrow_{AX} \text{true}}$$

$$\frac{\varphi \rightsquigarrow_{AX} \varphi'}{\varphi \wedge \psi \rightsquigarrow_{AX} \varphi' \wedge \psi} \quad \frac{\psi \rightsquigarrow_{AX} \psi'}{\varphi \wedge \psi \rightsquigarrow_{AX} \varphi \wedge \psi'} \quad \frac{\varphi \rightsquigarrow_{AX} \varphi'}{\neg \varphi \rightsquigarrow_{AX} \neg \varphi'}$$

consisting of Boolean simplifications, equations, equivalences and congruences;  
often subscript  $AX$  is dropped in  $\rightsquigarrow_{AX}$  when clear from context

## Soundness of Equational Reasoning

- we show that whenever  $AX$  is valid in the standard model  $\mathcal{M}$ , then
  - $\varphi \rightsquigarrow_{AX} \psi$  implies  $\mathcal{M} \models_{\alpha} \varphi \longleftrightarrow \psi$  for all  $\alpha$
  - so in particular  $\mathcal{M} \models \vec{\forall} \varphi \longleftrightarrow \psi$
- immediate consequence:  $\varphi \rightsquigarrow_{AX}^* \text{true}$  implies  $\mathcal{M} \models \vec{\forall} \varphi$
- define calculus:  $\vdash \vec{\forall} \varphi$  if  $\varphi \rightsquigarrow_{AX}^* \text{true}$
- example

$$\begin{aligned} & \text{plus}(\text{Zero}, \text{Zero}) =_{\text{Nat}} \text{times}(\text{Zero}, x) \\ & \rightsquigarrow \text{Zero} =_{\text{Nat}} \text{times}(\text{Zero}, x) \\ & \rightsquigarrow \text{Zero} =_{\text{Nat}} \text{Zero} \\ & \rightsquigarrow \text{true} \end{aligned}$$

and therefore  $\mathcal{M} \models \forall x. \text{plus}(\text{Zero}, \text{Zero}) =_{\text{Nat}} \text{times}(\text{Zero}, x)$

Proving Soundness of  $\rightsquigarrow$ :  $\varphi \rightsquigarrow \psi$  implies  $\mathcal{M} \models_{\alpha} \varphi \longleftrightarrow \psi$ 

by induction on  $\rightsquigarrow$  for arbitrary  $\alpha$

- case  $\frac{\varphi \rightsquigarrow \varphi'}{\varphi \wedge \psi \rightsquigarrow \varphi' \wedge \psi}$ 
  - IH:  $\mathcal{M} \models_{\alpha} \varphi \longleftrightarrow \varphi'$  for arbitrary  $\alpha$
  - conclude  $\mathcal{M} \models_{\alpha} \varphi \wedge \psi$   
iff  $\mathcal{M} \models_{\alpha} \varphi$  and  $\mathcal{M} \models_{\alpha} \psi$   
iff  $\mathcal{M} \models_{\alpha} \varphi'$  and  $\mathcal{M} \models_{\alpha} \psi$  (by IH)  
iff  $\mathcal{M} \models_{\alpha} \varphi' \wedge \psi$
  - in total:  $\mathcal{M} \models_{\alpha} \varphi \wedge \psi \longleftrightarrow \varphi' \wedge \psi$
- all other cases for Boolean simplifications and congruences are similar

Proving Soundness of  $\rightsquigarrow$ :  $\varphi \rightsquigarrow \psi$  implies  $\mathcal{M} \models_{\alpha} \varphi \longleftrightarrow \psi$ 

- case  $\frac{\vec{\forall} (\ell =_{\tau} r \longleftrightarrow \varphi) \in AX}{l\sigma =_{\tau} r\sigma \rightsquigarrow \varphi\sigma}$ 
  - $\mathcal{M} \models_{\alpha} l\sigma =_{\tau} r\sigma \longleftrightarrow \varphi\sigma$   
iff  $\mathcal{M} \models_{\alpha} (\ell =_{\tau} r \longleftrightarrow \varphi)\sigma$   
iff  $\mathcal{M} \models_{\beta} \ell =_{\tau} r \longleftrightarrow \varphi$  for  $\beta(x) = \llbracket \sigma(x) \rrbracket_{\alpha}$  (by SL)
  - the last statement is true since  $\vec{\forall} (\ell =_{\tau} r \longleftrightarrow \varphi) \in AX$  and thus,  
 $\mathcal{M} \models \vec{\forall} (\ell =_{\tau} r \longleftrightarrow \varphi) \in AX$

## Proving Soundness of $\rightsquigarrow$ : $\varphi \rightsquigarrow \psi$ implies $\mathcal{M} \models_{\alpha} \varphi \iff \psi$

- $$\frac{\forall \ell =_{\tau} r \in AX \quad s \leftrightarrow_{\{\ell=r\}} s'}{s =_{\tau} t \rightsquigarrow s' =_{\tau} t}$$
- case
    - premise  $\mathcal{M} \models \forall \ell =_{\tau} r$ , and  $s = C[\ell\sigma]$  and  $s' = C[r\sigma]$  where  $C$  is some context, i.e., term with one hole which can be filled via  $[\cdot]$
    - conclude  $\llbracket s \rrbracket_{\alpha}$ 

$$= \llbracket C[\ell\sigma] \rrbracket_{\alpha}$$

$$= C[\ell\sigma]_{\alpha} \downarrow \text{ (by reverse SL)}$$

$$= C\alpha[\ell\sigma\alpha] \downarrow = C\alpha[\ell\sigma\alpha] \downarrow \downarrow$$

$$\stackrel{(*)}{=} C\alpha[r\sigma\alpha] \downarrow \downarrow = C\alpha[r\sigma\alpha] \downarrow$$

$$= C[r\sigma]_{\alpha} \downarrow$$

$$= \llbracket C[r\sigma] \rrbracket_{\alpha} \text{ (by reverse SL)}$$

$$= \llbracket s' \rrbracket_{\alpha}$$
    - reason for (\*): premise implies  $\llbracket \ell \rrbracket_{\beta} = \llbracket r \rrbracket_{\beta}$  for  $\beta(x) = \llbracket \sigma(x) \rrbracket_{\alpha}$ , hence  $\llbracket \ell\sigma \rrbracket_{\alpha} = \llbracket r\sigma \rrbracket_{\alpha}$  (by SL), and thus,  $\ell\sigma\alpha \downarrow = r\sigma\alpha \downarrow$  (by reverse SL)
    - in total:  $\mathcal{M} \models_{\alpha} s =_{\tau} t \iff s' =_{\tau} t$

## Comparing $\rightsquigarrow$ with $\iff$

- $\iff$  rewrites on terms whereas  $\rightsquigarrow$  also simplifies Boolean connectives and uses axioms about equality  $=_{\tau}$
- $\iff$  uses defining equations of program whereas  $\rightsquigarrow_{AX}$  is parametrized by set of axioms
  - in particular proven properties like  $\forall xs. \text{reverse}(\text{reverse}(xs)) =_{\text{List}} xs$  can be added to set of axioms and then be used for  $\rightsquigarrow$
  - this addition of new knowledge greatly improves power, but can destroy both termination and confluence
    - example: adding  $\forall xs. xs =_{\text{List}} \text{reverse}(\text{reverse}(xs))$  to  $AX$  is bad idea
  - heuristics or user input required to select subset of theorems that are used with  $\rightsquigarrow$
  - new equations should be added in suitable direction
    - obvious:  $\forall xs. \text{reverse}(\text{reverse}(xs)) =_{\text{List}} xs$  is intended direction
    - direction sometimes not obvious for distributive laws

$$\forall x, y, z. \text{times}(\text{plus}(x, y), z) =_{\text{Nat}} \text{plus}(\text{times}(x, z), \text{times}(y, z))$$

reason for left-to-right: more often applicable

reason for right-to-left: term gets smaller

## Limits of $\rightsquigarrow$

- $\rightsquigarrow$  only works with universally quantified properties
  - defining equations
  - equivalences to simplify equalities  $=_{\tau}$
  - newly derived properties such as  $\forall xs. \text{reverse}(\text{reverse}(xs)) =_{\text{List}} xs$
  - $\rightsquigarrow$  can **not** deal with induction axioms such as the one for associativity of append (**app**)

$$\begin{aligned} & (\forall ys, zs. \text{app}(\text{app}(\text{Nil}, ys), zs) =_{\text{List}} \text{app}(\text{Nil}, \text{app}(ys, zs))) \\ \longrightarrow & (\forall x, xs. (\forall ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs)))) \longrightarrow \\ & (\forall ys, zs. \text{app}(\text{app}(\text{Cons}(x, xs), ys), zs) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs))) \\ \longrightarrow & (\forall xs, ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))) \end{aligned}$$

- in particular,  $\rightsquigarrow$  often cannot perform any simplification without induction proving

$$\text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))$$

cannot be simplified by  $\rightsquigarrow$  using the existing axioms

## Induction in Combination with Equational Reasoning

- aim: prove equality  $\forall \ell =_{\tau} r$
- approach:
  - select induction variable  $x$
  - reorder quantifiers such that  $\forall \ell =_{\tau} r$  is written as  $\forall x. \varphi$
  - build induction formula w.r.t. slide 22

$$\varphi_1 \longrightarrow \dots \longrightarrow \varphi_n \longrightarrow \forall x. \varphi$$

(no outer universal quantifier, since by construction above formula has no free variables)

- try to prove each  $\varphi_i$  via  $\rightsquigarrow$

### Example: Associativity of Append

- aim: prove equality  $\forall xs, ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))$
- approach:
  - select induction variable  $xs$
  - reordering of quantifiers not required
  - the induction formula is presented on slide 35
  - $\varphi_1$  is

$$\forall ys, zs. \text{app}(\text{app}(\text{Nil}, ys), zs) =_{\text{List}} \text{app}(\text{Nil}, \text{app}(ys, zs))$$

so we simply evaluate

$$\begin{aligned} & \text{app}(\text{app}(\text{Nil}, ys), zs) =_{\text{List}} \text{app}(\text{Nil}, \text{app}(ys, zs)) \\ \rightsquigarrow & \text{app}(ys, zs) =_{\text{List}} \text{app}(\text{Nil}, \text{app}(ys, zs)) \\ \rightsquigarrow & \text{app}(ys, zs) =_{\text{List}} \text{app}(ys, zs) \\ \rightsquigarrow & \text{true} \end{aligned}$$

### Example: Associativity of Append, Continued

- proving  $\forall xs, ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))$
- approach: . . .
  - $\varphi_2$  is
 
$$\forall x, xs. (\forall ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))) \longrightarrow$$

$$(\forall ys, zs. \text{app}(\text{app}(\text{Cons}(x, xs), ys), zs) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs)))$$

so we try to prove the rhs of  $\longrightarrow$  via  $\rightsquigarrow$

$$\begin{aligned} & \text{app}(\text{app}(\text{Cons}(x, xs), ys), zs) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs)) \\ \rightsquigarrow & \text{app}(\text{Cons}(x, \text{app}(xs, ys)), zs) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs)) \\ \rightsquigarrow & \text{Cons}(x, \text{app}(\text{app}(xs, ys), zs)) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs)) \\ \rightsquigarrow & \text{Cons}(x, \text{app}(\text{app}(xs, ys), zs)) =_{\text{List}} \text{Cons}(x, \text{app}(xs, \text{app}(ys, zs))) \\ \rightsquigarrow & x =_{\text{Nat}} x \wedge \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs)) \\ \rightsquigarrow & \text{true} \wedge \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs)) \\ \rightsquigarrow & \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs)) \\ & \neq \text{true} \end{aligned}$$

- problem: we get stuck, since currently IH is unused

### Integrating IHs into Equational Reasoning

- recall structure of induction formula for formula  $\varphi$  and constructor  $c_i$ :

$$\varphi_i := \forall x_1, \dots, x_{m_i}. \underbrace{\left( \bigwedge_{j, \tau_i, j = \tau} \varphi[x/x_j] \right)}_{\text{IHs for recursive arguments}} \longrightarrow \varphi[x/c_i(x_1, \dots, x_{m_i})]$$

- idea: for proving  $\varphi_i$  try to show  $\varphi[x/c_i(x_1, \dots, x_{m_i})]$  by evaluating it to true via  $\rightsquigarrow$ , where each IH  $\varphi[x/x_j]$  is added as equality
- append-example
  - aim:
 
$$\text{app}(\text{app}(\text{Cons}(x, xs), ys), zs) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs)) \rightsquigarrow^* \text{true}$$
  - add IH  $\forall ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))$  to axioms
- problem IH  $\varphi[x/x_j]$  is not universally quantified equation, since variable  $x_j$  is free (in append example, this would be  $xs$ )

### Integrating IHs into Equational Reasoning, Continued

- to solve problem, extend  $\rightsquigarrow$  to allow evaluation with equations that contain free variables
- add two new inference rules

$$\frac{\forall \vec{x}. \ell =_{\tau} r \in AX \quad s \hookrightarrow_{\{\ell=r\}} s'}{s =_{\tau} t \rightsquigarrow_{AX} s' =_{\tau} t} \quad \frac{\forall \vec{x}. \ell =_{\tau} r \in AX \quad t \hookrightarrow_{\{r=\ell\}} t'}{s =_{\tau} t \rightsquigarrow_{AX} s =_{\tau} t'}$$

where in both inference rules, only the variables of  $\vec{x}$  may be instantiated in the equation  $\ell = r$  when simplifying with  $\hookrightarrow$ ; so the chosen substitution  $\sigma$  must satisfy  $\sigma(y) = y$  for all  $y \notin \vec{x}$

- the **swap of direction**, i.e., the  $r = \ell$  in the second rule is intended and a **heuristic**
  - either apply the IH on some lhs of an equality from left-to-right
  - or apply the IH on some rhs of an equality from right-to-left

in both cases, an application will make both sides on the equality more equal

- another heuristic is to **apply each IH only once**

## Example: Associativity of Append, Continued

- proving  $\forall xs, ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))$
- approach: ...
  - $\varphi_2$  is  $\forall x, xs. (\forall ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))) \longrightarrow$   
 $(\forall ys, zs. \text{app}(\text{app}(\text{Cons}(x, xs), ys), zs) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs)))$

so we try to prove the rhs of  $\longrightarrow$  via  $\rightsquigarrow$  and add

$$\forall ys, zs. \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))$$

to the set of axioms (only for the proof of  $\varphi_2$ ); then

$$\begin{aligned} & \text{app}(\text{app}(\text{Cons}(x, xs), ys), zs) =_{\text{List}} \text{app}(\text{Cons}(x, xs), \text{app}(ys, zs)) \\ \rightsquigarrow^* & \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs)) \\ \rightsquigarrow & \text{app}(xs, \text{app}(ys, zs)) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs)) \\ \rightsquigarrow & \text{true} \end{aligned}$$

here it is important to apply the IH only once, otherwise one would get

$$\text{app}(xs, \text{app}(ys, zs)) =_{\text{List}} \text{app}(\text{app}(xs, ys), zs)$$

## Integrating IHs into Equational Reasoning, Soundness

- aim: prove  $\mathcal{M} \models \varphi_i$  for

$$\varphi_i := \bigvee_j \bigwedge_j \psi_j \longrightarrow \psi$$

IHS

where we assume that  $\psi \rightsquigarrow^*$  true with the additional local axioms of the IHs  $\psi_j$

- hence show  $\mathcal{M} \models_\alpha \psi$  under the assumptions  $\mathcal{M} \models_\alpha \psi_j$  for all IHs  $\psi_j$
- by existing soundness proof of  $\rightsquigarrow$  we can nearly conclude  $\mathcal{M} \models_\alpha \psi$  from  $\psi \rightsquigarrow^*$  true
- only gap: proof needs to cover new inference rules on slide 40

## Soundness of Partially Quantified Equation Application

$$\forall \vec{x}. \ell =_\tau r \in AX \quad s \hookrightarrow_{\{\ell=r\}} s'$$

- case  $\frac{s =_\tau t \rightsquigarrow s' =_\tau t}{s =_\tau t \rightsquigarrow s' =_\tau t}$  with  $\sigma(y) = y$  for all  $y \notin \vec{x}$  (and not  $\mathcal{M} \models \bigvee \ell =_\tau r$ )
  - premise is  $\mathcal{M} \models_\alpha \forall \vec{x}. \ell =_\tau r$  (and not  $\mathcal{M} \models \bigvee \ell =_\tau r$ )  
and  $s = C[\ell\sigma]$  and  $s' = C[r\sigma]$  as before
  - conclude  $\llbracket s \rrbracket_\alpha = \llbracket s' \rrbracket_\alpha$  as on slide 33 as main step to derive  $\mathcal{M} \models_\alpha s =_\tau t \longleftrightarrow s' =_\tau t$
  - only change is how to obtain  $\llbracket \ell \rrbracket_\beta = \llbracket r \rrbracket_\beta$  for  $\beta(x) = \llbracket \sigma(x) \rrbracket_\alpha$
  - new proof
    - let  $\vec{x} = x_1, \dots, x_k$
    - premise implies  $\llbracket \ell \rrbracket_{\alpha[x_1:=a_1, \dots, x_k:=a_k]} = \llbracket r \rrbracket_{\alpha[x_1:=a_1, \dots, x_k:=a_k]}$  for arbitrary  $a_i$ , so in particular for  $a_i = \llbracket \sigma(x_i) \rrbracket_\alpha$
    - it now suffices to prove that  $\alpha[x_1 := a_1, \dots, x_k := a_k] = \beta$
    - consider two cases
    - for variables  $x_i$  we have

$$\alpha[x_1 := a_1, \dots, x_k := a_k](x_i) = a_i = \llbracket \sigma(x_i) \rrbracket_\alpha = \beta(x_i)$$

- for all other variables  $y \notin \vec{x}$  we have

$$\alpha[x_1 := a_1, \dots, x_k := a_k](y) = \alpha(y) = \llbracket y \rrbracket_\alpha = \llbracket \sigma(y) \rrbracket_\alpha = \beta(y)$$

## Summary

- framework for inductive proofs combined with equational reasoning
- apply induction first
- then prove each case  $\bigvee \bigwedge \psi_j \longrightarrow \psi$  via evaluation  $\psi \rightsquigarrow^*$  true where IHs  $\psi_j$  become local axioms
- free variables in IHs (induction variables) may not be instantiated by  $\rightsquigarrow$ , all the other variables may be instantiated (“arbitrary” variables)
- heuristic: apply IHs only once
- upcoming: positive and negative **examples**, guidelines, extensions

## Examples, Guidelines, and Extensions

### Associativity of Append

- program
 
$$\text{app}(\text{Cons}(x, xs), ys) = \text{Cons}(x, \text{app}(xs, ys))$$

$$\text{app}(\text{Nil}, ys) = ys$$
- formula
 
$$\vec{\forall} \text{app}(\text{app}(xs, ys), zs) =_{\text{List}} \text{app}(xs, \text{app}(ys, zs))$$
- induction on  $xs$  works successfully
- what about induction on  $ys$  (or  $zs$ )?
- base case already gets stuck
 
$$\text{app}(\text{app}(xs, \text{Nil}), zs) =_{\text{List}} \text{app}(xs, \text{app}(\text{Nil}, zs))$$

$$\rightsquigarrow \text{app}(\text{app}(xs, \text{Nil}), zs) =_{\text{List}} \text{app}(xs, zs)$$
- problem:  $ys$  is argument on second position of append, whereas case analysis in lhs of append happens on first argument
- guideline: **select variables such that case analysis triggers evaluation**

### Commutativity of Addition

- program
 
$$\text{plus}(\text{Succ}(x), y) = \text{Succ}(\text{plus}(x, y))$$

$$\text{plus}(\text{Zero}, y) = y$$
- formula
 
$$\vec{\forall} \text{plus}(x, y) =_{\text{Nat}} \text{plus}(y, x)$$
- let us try induction on  $x$
- base case already gets stuck
 
$$\text{plus}(\text{Zero}, y) =_{\text{Nat}} \text{plus}(y, \text{Zero})$$

$$\rightsquigarrow y =_{\text{Nat}} \text{plus}(y, \text{Zero})$$
- **final result suggests required lemma:**  $\text{Zero}$  is also right neutral
- $\forall x. \text{plus}(x, \text{Zero}) =_{\text{Nat}} x$  can be proven with our approach
- then this lemma can be added to  $AX$  and base case of commutativity-proof can be completed

### Right-Zero of Addition

- program
 
$$\text{plus}(\text{Succ}(x), y) = \text{Succ}(\text{plus}(x, y))$$

$$\text{plus}(\text{Zero}, y) = y$$
- formula
 
$$\vec{\forall} \text{plus}(x, \text{Zero}) =_{\text{Nat}} x$$
- only one possible induction variable:  $x$
- base case:
 
$$\text{plus}(\text{Zero}, \text{Zero}) =_{\text{Nat}} \text{Zero} \rightsquigarrow \text{Zero} =_{\text{Nat}} \text{Zero} \rightsquigarrow \text{true}$$
- step case adds IH  $\text{plus}(x, \text{Zero}) =_{\text{Nat}} x$  as axiom and we get
 
$$\text{plus}(\text{Succ}(x), \text{Zero}) =_{\text{Nat}} \text{Succ}(x)$$

$$\rightsquigarrow \text{Succ}(\text{plus}(x, \text{Zero})) =_{\text{Nat}} \text{Succ}(x)$$

$$\rightsquigarrow \text{Succ}(x) =_{\text{Nat}} \text{Succ}(x)$$

$$\rightsquigarrow \text{true}$$

## Commutativity of Addition

- formula

$$\vec{\forall} \text{plus}(x, y) =_{\text{Nat}} \text{plus}(y, x)$$

- step case adds IH  $\forall y. \text{plus}(x, y) =_{\text{Nat}} \text{plus}(y, x)$  to axioms and we get

$$\text{plus}(\text{Succ}(x), y) =_{\text{Nat}} \text{plus}(y, \text{Succ}(x))$$

$$\rightsquigarrow \text{Succ}(\text{plus}(x, y)) =_{\text{Nat}} \text{plus}(y, \text{Succ}(x))$$

$$\rightsquigarrow \text{Succ}(\text{plus}(y, x)) =_{\text{Nat}} \text{plus}(y, \text{Succ}(x))$$

- final result suggests required lemma: `Succ` on second argument can be moved outside
- $\forall x, y. \text{plus}(x, \text{Succ}(y)) =_{\text{Nat}} \text{Succ}(\text{plus}(x, y))$  can be proven with our approach (induction on  $x$ )
- then this lemma can be added to  $AX$  and commutativity-proof can be completed

## Fast Implementation of Reversal

- program

$$\text{app}(\text{Cons}(x, xs), ys) = \text{Cons}(x, \text{app}(xs, ys))$$

$$\text{app}(\text{Nil}, ys) = ys$$

$$\text{rev}(\text{Cons}(x, xs)) = \text{app}(\text{rev}(xs), \text{Cons}(x, \text{Nil}))$$

$$\text{rev}(\text{Nil}) = \text{Nil}$$

$$r(\text{Cons}(x, xs), ys) = r(xs, \text{Cons}(x, ys))$$

$$r(\text{Nil}, ys) = ys$$

$$\text{rev\_fast}(xs) = r(xs, \text{Nil})$$

- aim: show that both implementations of reverse are equivalent, so that the naive implementation can be replaced by the faster one

$$\forall xs. \text{rev\_fast}(xs) =_{\text{List}} \text{rev}(xs)$$

- applying  $\rightsquigarrow$  first yields desired lemma

$$\forall xs. r(xs, \text{Nil}) =_{\text{List}} \text{rev}(xs)$$

## Generalizations Required

- for induction for the following formula there is only one choice:  $xs$

$$\forall xs. r(xs, \text{Nil}) =_{\text{List}} \text{rev}(xs)$$

- step-case gets stuck

$$r(\text{Cons}(x, xs), \text{Nil}) =_{\text{List}} \text{rev}(\text{Cons}(x, xs))$$

$$\rightsquigarrow^* r(xs, \text{Cons}(x, \text{Nil})) =_{\text{List}} \text{app}(\text{rev}(xs), \text{Cons}(x, \text{Nil}))$$

$$\rightsquigarrow r(xs, \text{Cons}(x, \text{Nil})) =_{\text{List}} \text{app}(r(xs, \text{Nil}), \text{Cons}(x, \text{Nil}))$$

- problem: the second argument `Nil` of `r` in formula is too specific
- solution: **generalize formula** by replacing constants by variables
- naive replacement does not work, since it does not hold

$$\forall xs, ys. r(xs, ys) =_{\text{List}} \text{rev}(xs)$$

- creativity required

$$\forall xs, ys. r(xs, ys) =_{\text{List}} \text{app}(\text{rev}(xs), ys)$$

## Fast Implementation of Reversal, Continued

- proving main formula by induction on  $xs$ , since recursion is on  $xs$

$$\forall xs, ys. r(xs, ys) =_{\text{List}} \text{app}(\text{rev}(xs), ys)$$

- base-case

$$r(\text{Nil}, ys) =_{\text{List}} \text{app}(\text{rev}(\text{Nil}), ys)$$

$$\rightsquigarrow^* ys =_{\text{List}} ys \rightsquigarrow \text{true}$$

- step-case solved with **associativity** of append and **IH** added to axioms

$$r(\text{Cons}(x, xs), ys) =_{\text{List}} \text{app}(\text{rev}(\text{Cons}(x, xs)), ys)$$

$$\rightsquigarrow r(xs, \text{Cons}(x, ys)) =_{\text{List}} \text{app}(\text{rev}(\text{Cons}(x, xs)), ys)$$

$$\rightsquigarrow \text{app}(\text{rev}(xs), \text{Cons}(x, ys)) =_{\text{List}} \text{app}(\text{rev}(\text{Cons}(x, xs)), ys)$$

$$\rightsquigarrow \text{app}(\text{rev}(xs), \text{Cons}(x, ys)) =_{\text{List}} \text{app}(\text{app}(\text{rev}(xs), \text{Cons}(x, \text{Nil})), ys)$$

$$\rightsquigarrow \text{app}(\text{rev}(xs), \text{Cons}(x, ys)) =_{\text{List}} \text{app}(\text{rev}(xs), \text{app}(\text{Cons}(x, \text{Nil}), ys))$$

$$\rightsquigarrow \text{app}(\text{rev}(xs), \text{Cons}(x, ys)) =_{\text{List}} \text{app}(\text{rev}(xs), \text{Cons}(x, \text{app}(\text{Nil}, ys)))$$

$$\rightsquigarrow \text{app}(\text{rev}(xs), \text{Cons}(x, ys)) =_{\text{List}} \text{app}(\text{rev}(xs), \text{Cons}(x, ys)) \rightsquigarrow \text{true}$$

## Fast Implementation of Reversal, Finalized

- now add main formula to axioms, so that it can be used by  $\rightsquigarrow$

$$\forall xs, ys. r(xs, ys) =_{\text{List}} \text{app}(\text{rev}(xs), ys)$$

- then for our initial aim we get

$$\begin{aligned} \text{rev\_fast}(xs) &=_{\text{List}} \text{rev}(xs) \\ \rightsquigarrow r(xs, \text{Nil}) &=_{\text{List}} \text{rev}(xs) \\ \rightsquigarrow \text{app}(\text{rev}(xs), \text{Nil}) &=_{\text{List}} \text{rev}(xs) \end{aligned}$$

- at this point one easily identifies a missing property

$$\forall xs. \text{app}(xs, \text{Nil}) =_{\text{List}} xs$$

which is proven by induction on  $xs$  in combination with  $\rightsquigarrow$

- afterwards it is trivial to complete the equivalence proof of the two reversal implementations

## Another Problem

- consider the following program

$$\begin{aligned} \text{half}(\text{Zero}) &= \text{Zero} \\ \text{half}(\text{Succ}(\text{Zero})) &= \text{Zero} \\ \text{half}(\text{Succ}(\text{Succ}(x))) &= \text{Succ}(\text{half}(x)) \\ \text{le}(\text{Zero}, y) &= \text{True} \\ \text{le}(\text{Succ}(x), \text{Zero}) &= \text{False} \\ \text{le}(\text{Succ}(x), \text{Succ}(y)) &= \text{le}(x, y) \end{aligned}$$

- and the desired property

$$\forall x. \text{le}(\text{half}(x), x) =_{\text{Bool}} \text{True}$$

- induction on  $x$  will get stuck, since the step-case  $\text{Succ}(x)$  does not permit evaluation w.r.t.  $\text{half}$ -equations
- better induction is desirable, namely rule that corresponds to algorithm definition (e.g. of  $\text{half}$ ) with cases that correspond to patterns in  $\text{lhss}$

## Induction w.r.t. Algorithm

- **induction w.r.t. algorithm** was informally performed on slide 4/28
  - select some  $n$ -ary function  $f$
  - each  $f$ -equation is turned into one case
  - for each **recursive**  $f$ -call in rhs get one IH
- example: for algorithm

$$\begin{aligned} \text{half}(\text{Zero}) &= \text{Zero} \\ \text{half}(\text{Succ}(\text{Zero})) &= \text{Zero} \\ \text{half}(\text{Succ}(\text{Succ}(x))) &= \text{Succ}(\text{half}(x)) \end{aligned}$$

the induction rule for  $\text{half}$  is

$$\begin{aligned} &\varphi[y/\text{Zero}] \\ \longrightarrow &\varphi[y/\text{Succ}(\text{Zero})] \\ \longrightarrow &(\forall x. \varphi[y/x] \longrightarrow \varphi[y/\text{Succ}(\text{Succ}(x))]) \\ \longrightarrow &\forall y. \varphi \end{aligned}$$

## Induction w.r.t. Algorithm

- **induction w.r.t. algorithm** formally defined
  - let  $f$  be  $n$ -ary defined function within **well-defined** program
  - let there be  $k$  defining equations for  $f$
  - let  $\varphi$  be some formula which has exactly  $n$  free variables  $x_1, \dots, x_n$
  - then the **induction rule for  $f$**  is

$$\varphi_{\text{ind},f} := \psi_1 \longrightarrow \dots \longrightarrow \psi_k \longrightarrow \forall x_1, \dots, x_n. \varphi$$

where for the  $i$ -th  $f$ -equation  $f(\ell_1, \dots, \ell_n) = r$  we define

$$\psi_i := \vec{\nabla} \left( \bigwedge_{r \geq f(r_1, \dots, r_n)} \varphi[x_1/r_1, \dots, x_n/r_n] \right) \longrightarrow \varphi[x_1/\ell_1, \dots, x_n/\ell_n]$$

where  $\vec{\nabla}$  ranges over all variables in the equation

- **properties**
  - $\mathcal{M} \models \varphi_{\text{ind},f}$ ; reason: pattern-completeness and termination ( $SN(\leftrightarrow \circ \geq)$ )
  - heuristic: good idea to prove properties  $\vec{\nabla} \varphi$  about function  $f$  via  $\varphi_{f,\text{ind}}$
  - reason: structure will always allow one evaluation step of  $f$ -invocation

## Back to Example

- consider program

$\text{half}(\text{Zero}) = \text{Zero}$   
 $\text{half}(\text{Succ}(\text{Zero})) = \text{Zero}$   
 $\text{half}(\text{Succ}(\text{Succ}(x))) = \text{Succ}(\text{half}(x))$   
 $\text{le}(\text{Zero}, y) = \text{True}$   
 $\text{le}(\text{Succ}(x), \text{Zero}) = \text{False}$   
 $\text{le}(\text{Succ}(x), \text{Succ}(y)) = \text{le}(x, y)$

- for property

$$\forall x. \text{le}(\text{half}(x), x) =_{\text{Bool}} \text{True}$$

choose induction for **half** (and not for **le**), since **half** is inner function call; hopefully evaluation of inner function calls will enable evaluation of outer function calls

## (Nearly) Completing the Proof

- applying induction for **half** on  $\forall x. \text{le}(\text{half}(x), x) =_{\text{Bool}} \text{True}$

turns this problem into three new proof obligations

- $\text{le}(\text{half}(\text{Zero}), \text{Zero}) =_{\text{Bool}} \text{True}$
- $\text{le}(\text{half}(\text{Succ}(\text{Zero})), \text{Succ}(\text{Zero})) =_{\text{Bool}} \text{True}$
- $\text{le}(\text{half}(\text{Succ}(\text{Succ}(x))), \text{Succ}(\text{Succ}(x))) =_{\text{Bool}} \text{True}$  where  $\text{le}(\text{half}(x), x) =_{\text{Bool}} \text{True}$  can be assumed as IH

- the first two are easy, the third one works as follows

$$\begin{aligned} & \text{le}(\text{half}(\text{Succ}(\text{Succ}(x))), \text{Succ}(\text{Succ}(x))) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(\text{Succ}(\text{half}(x)), \text{Succ}(\text{Succ}(x))) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(\text{half}(x), \text{Succ}(x)) =_{\text{Bool}} \text{True} \end{aligned}$$

- here there is another problem, namely that the IH is not applicable
- problem solvable by proving an **implication** like  $\text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(x, \text{Succ}(y)) =_{\text{Bool}} \text{True}$ ; uses **equational reasoning with conditions**; covered informally only

## Equational Reasoning with Conditions

- generalization: instead of pure equalities also support implications
- simplifications with  $\rightsquigarrow$  can happen on **both sides of implication**, since  $\rightsquigarrow$  yields equivalent formulas
- applying conditional equations triggers new proofs: preconditions must be satisfied
- example:
  - assume axioms contain conditional equality  $\varphi \longrightarrow \ell =_{\tau} r$ , e.g., from IH
  - current goal is implication  $\psi \longrightarrow C[\ell\sigma] =_{\tau} t$
  - we would like to replace goal by  $\psi \longrightarrow C[r\sigma] =_{\tau} t$
  - but then we must ensure  $\psi \longrightarrow \varphi\sigma$ , e.g., via  $\psi \longrightarrow \varphi\sigma \rightsquigarrow^* \text{true}$
- $\rightsquigarrow$  must be extended to perform more Boolean reasoning
- not done formally at this point

## Equational Reasoning with Conditions, Example

- property

$$\text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(x, \text{Succ}(y)) =_{\text{Bool}} \text{True}$$

- apply induction on **le**
- first case

$$\begin{aligned} & \text{le}(\text{Zero}, y) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(\text{Zero}, \text{Succ}(y)) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(\text{Zero}, y) =_{\text{Bool}} \text{True} \longrightarrow \text{True} =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(\text{Zero}, y) =_{\text{Bool}} \text{True} \longrightarrow \text{true} \\ \rightsquigarrow & \text{true} \end{aligned}$$

- second case

$$\begin{aligned} & \text{le}(\text{Succ}(x), \text{Zero}) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(\text{Succ}(x), \text{Succ}(\text{Zero})) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{False} =_{\text{Bool}} \text{True} \longrightarrow \text{le}(\text{Succ}(x), \text{Succ}(\text{Zero})) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{false} \longrightarrow \text{le}(\text{Succ}(x), \text{Succ}(\text{Zero})) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{true} \end{aligned}$$

## Equational Reasoning with Conditions, Example

- property

$$\text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(x, \text{Succ}(y)) =_{\text{Bool}} \text{True}$$

- third case has IH

$$\text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(x, \text{Succ}(y)) =_{\text{Bool}} \text{True}$$

and we reason as follows

$$\begin{aligned} & \text{le}(\text{Succ}(x), \text{Succ}(y)) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(\text{Succ}(x), \text{Succ}(\text{Succ}(y))) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(\text{Succ}(x), \text{Succ}(\text{Succ}(y))) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{le}(x, \text{Succ}(y)) =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{True} =_{\text{Bool}} \text{True} \\ \rightsquigarrow & \text{le}(x, y) =_{\text{Bool}} \text{True} \longrightarrow \text{true} \\ \rightsquigarrow & \text{true} \end{aligned}$$

- proof of property  $\forall x. \text{le}(\text{half}(x), x) =_{\text{Bool}} \text{True}$  finished

## Final Example: Insertion Sort

- consider insertion sort

$$\begin{aligned} & \text{le}(\text{Zero}, y) = \text{True} \\ & \text{le}(\text{Succ}(x), \text{Zero}) = \text{False} \\ & \text{le}(\text{Succ}(x), \text{Succ}(y)) = \text{le}(x, y) \\ & \text{if}(\text{True}, xs, ys) = xs \\ & \text{if}(\text{False}, xs, ys) = ys \\ & \text{insort}(x, \text{Nil}) = \text{Cons}(x, \text{Nil}) \\ & \text{insort}(x, \text{Cons}(y, ys)) = \text{if}(\text{le}(x, y), \text{Cons}(x, \text{Cons}(y, ys)), \text{Cons}(y, \text{insort}(x, ys))) \\ & \text{sort}(\text{Nil}) = \text{Nil} \\ & \text{sort}(\text{Cons}(x, xs)) = \text{insort}(x, \text{sort}(xs)) \end{aligned}$$

- aim: prove soundness, e.g., result is sorted
- problem: how to express “being sorted”?
- in general: how to express properties if certain primitives are not available?

## Expressing Properties

- solution: express **properties via functional programs**

$$\begin{aligned} & \dots = \dots \\ & \text{sort}(\text{Cons}(x, xs)) = \text{insort}(x, \text{sort}(xs)) \end{aligned}$$

algorithm above, properties for specification below

$$\begin{aligned} & \text{and}(\text{True}, b) = b \\ & \text{and}(\text{False}, b) = \text{False} \\ & \text{all\_le}(x, \text{Nil}) = \text{True} \\ & \text{all\_le}(x, \text{Cons}(y, ys)) = \text{and}(\text{le}(x, y), \text{all\_le}(x, ys)) \\ & \text{sorted}(\text{Nil}) = \text{True} \\ & \text{sorted}(\text{Cons}(x, xs)) = \text{and}(\text{all\_le}(x, xs), \text{sorted}(xs)) \end{aligned}$$

- example properties (where  $b =_{\text{Bool}} \text{True}$  is written just as  $b$ )
  - $\text{sorted}(\text{insort}(x, xs)) =_{\text{Bool}} \text{sorted}(xs)$
  - $\text{sorted}(\text{sort}(xs))$
- important: functional programs for specifications should be simple; they must be readable for validation and need not be efficient

## Example: Soundness of sort

- already **assume property** of **insort**:

$$\forall x, xs. \text{sorted}(\text{insort}(x, xs)) =_{\text{Bool}} \text{sorted}(xs) \quad (*)$$

speculative proofs are risky: conjectures might be wrong

- property  $\forall xs. \text{sorted}(\text{sort}(xs))$  is shown by induction on  $xs$
- base case:

$$\begin{aligned} & \text{sorted}(\text{sort}(\text{Nil})) \\ \rightsquigarrow & \text{sorted}(\text{Nil}) \\ \rightsquigarrow & \text{True} \quad (\text{recall: syntax omits } =_{\text{Bool}} \text{True}) \\ \rightsquigarrow & \text{true} \end{aligned}$$

- step case with IH  $\text{sorted}(\text{sort}(xs))$ :

$$\begin{aligned} & \text{sorted}(\text{sort}(\text{Cons}(x, xs))) \\ \rightsquigarrow & \text{sorted}(\text{insort}(x, \text{sort}(xs))) \\ \rightsquigarrow & \text{sorted}(\text{sort}(xs)) \quad (*) \\ \rightsquigarrow & \text{True} \end{aligned}$$

### Example: Soundness of `insert`

- prove  $\forall x, xs. \text{sorted}(\text{insert}(x, xs)) =_{\text{Bool}} \text{sorted}(xs)$  by induction on  $xs$
- base case:

$$\begin{aligned} & \text{sorted}(\text{insert}(x, \text{Nil})) =_{\text{Bool}} \text{sorted}(\text{Nil}) \\ \rightsquigarrow & \text{sorted}(\text{Cons}(x, \text{Nil})) =_{\text{Bool}} \text{sorted}(\text{Nil}) \\ \rightsquigarrow & \text{and}(\text{all\_le}(x, \text{Nil}), \text{sorted}(\text{Nil})) =_{\text{Bool}} \text{sorted}(\text{Nil}) \\ \rightsquigarrow & \text{and}(\text{True}, \text{sorted}(\text{Nil})) =_{\text{Bool}} \text{sorted}(\text{Nil}) \\ \rightsquigarrow & \text{sorted}(\text{Nil}) =_{\text{Bool}} \text{sorted}(\text{Nil}) \\ \rightsquigarrow & \text{true} \end{aligned}$$

### Example: Soundness of `insert`, Step Case

- prove  $\forall x, xs. \text{sorted}(\text{insert}(x, xs)) =_{\text{Bool}} \text{sorted}(xs)$  by induction on  $xs$
- step case with IH  $\forall x. \text{sorted}(\text{insert}(x, ys)) =_{\text{Bool}} \text{sorted}(ys)$ :
 
$$\begin{aligned} & \text{sorted}(\text{insert}(x, \text{Cons}(y, ys))) =_{\text{Bool}} \text{sorted}(\text{Cons}(y, ys)) \\ \rightsquigarrow & \text{sorted}(\text{if}(\text{le}(x, y), \text{Cons}(x, \text{Cons}(y, ys)), \text{Cons}(y, \text{insert}(x, ys)))) =_{\text{Bool}} \dots \end{aligned}$$

now perform **case analysis** on first argument of `if`

- case  $\text{le}(x, y)$ , i.e.,  $\text{le}(x, y) =_{\text{Bool}} \text{True}$ 

$$\begin{aligned} & \text{sorted}(\text{if}(\text{le}(x, y), \text{Cons}(x, \text{Cons}(y, ys)), \text{Cons}(y, \text{insert}(x, ys)))) =_{\text{Bool}} \dots \\ \rightsquigarrow & \text{sorted}(\text{if}(\text{True}, \text{Cons}(x, \text{Cons}(y, ys)), \text{Cons}(y, \text{insert}(x, ys)))) =_{\text{Bool}} \dots \\ \rightsquigarrow & \text{sorted}(\text{Cons}(x, \text{Cons}(y, ys))) =_{\text{Bool}} \text{sorted}(\text{Cons}(y, ys)) \\ \rightsquigarrow & \text{and}(\text{all\_le}(x, \text{Cons}(y, ys)), \text{sorted}(\text{Cons}(y, ys))) =_{\text{Bool}} \text{sorted}(\text{Cons}(y, ys)) \end{aligned}$$

the key to resolve this final formula is the following auxiliary property

$$\vec{\forall} \text{le}(x, y) \longrightarrow \text{sorted}(\text{Cons}(y, zs)) \longrightarrow \text{all\_le}(x, \text{Cons}(y, zs))$$

this property can be proved by induction on  $zs$  but it will require a transitivity property for `le`

### Example: Soundness of `insert`, Final Part

- prove  $\forall x, xs. \text{sorted}(\text{insert}(x, xs)) =_{\text{Bool}} \text{sorted}(xs)$  by ind. on  $xs$
- step case with IH  $\forall x. \text{sorted}(\text{insert}(x, ys)) =_{\text{Bool}} \text{sorted}(ys)$ :
 
$$\begin{aligned} & \text{sorted}(\text{insert}(x, \text{Cons}(y, ys))) =_{\text{Bool}} \text{sorted}(\text{Cons}(y, ys)) \\ \rightsquigarrow & \text{sorted}(\text{if}(\text{le}(x, y), \text{Cons}(x, \text{Cons}(y, ys)), \text{Cons}(y, \text{insert}(x, ys)))) =_{\text{Bool}} \dots \end{aligned}$$

- case  $\neg \text{le}(x, y)$ , i.e.,  $\text{le}(x, y) =_{\text{Bool}} \text{False}$ 

$$\begin{aligned} & \text{sorted}(\text{if}(\text{le}(x, y), \text{Cons}(x, \text{Cons}(y, ys)), \text{Cons}(y, \text{insert}(x, ys)))) =_{\text{Bool}} \dots \\ \rightsquigarrow & \text{sorted}(\text{if}(\text{False}, \text{Cons}(x, \text{Cons}(y, ys)), \text{Cons}(y, \text{insert}(x, ys)))) =_{\text{Bool}} \dots \\ \rightsquigarrow & \text{sorted}(\text{Cons}(y, \text{insert}(x, ys))) =_{\text{Bool}} \text{sorted}(\text{Cons}(y, ys)) \\ \rightsquigarrow & \text{and}(\text{all\_le}(y, \text{insert}(x, ys)), \text{sorted}(\text{insert}(x, ys))) =_{\text{Bool}} \text{sorted}(\text{Cons}(y, ys)) \\ \rightsquigarrow & \text{and}(\text{all\_le}(y, \text{insert}(x, ys)), \text{sorted}(ys)) =_{\text{Bool}} \text{sorted}(\text{Cons}(y, ys)) \\ \rightsquigarrow & \text{and}(\text{all\_le}(y, \text{insert}(x, ys)), \text{sorted}(ys)) =_{\text{Bool}} \text{and}(\text{all\_le}(y, ys), \text{sorted}(ys)) \end{aligned}$$

at this point identify further required auxiliary properties

- $\vec{\forall} \text{all\_le}(y, \text{insert}(x, ys)) =_{\text{Bool}} \text{all\_le}(y, \text{Cons}(x, ys))$
- $\vec{\forall} \text{le}(x, y) =_{\text{Bool}} \text{False} \longrightarrow \text{le}(y, x) =_{\text{Bool}} \text{True}$

these allow us to complete this case and hence the overall proof for `sorted`

### Summary

- definition of several axioms (inference rules)
  - all axioms are satisfied in standard model, so they are consistent
- equational properties can often conveniently be proved via induction and equational reasoning via  $\rightsquigarrow$
- induction w.r.t. algorithm preferable whenever algorithms use more complex pattern structure than  $c_i(x_1, \dots, x_n)$  for all constructors  $c_i$
- when getting stuck with  $\rightsquigarrow$  try to detect suitable auxiliary property; after proving it, add it to set of axioms for evaluation
- not every property can be expressed purely equational; e.g., Boolean connectives are sometimes required
- specify properties of functional programs (e.g., `sort`) as functional programs (e.g., `sorted`)
- Demo05.thy: Isabelle formalization of all example proofs