universität innsbruck

Summer Term 2024

Program Verification

Part 5 – Reasoning about Functional Programs

René Thiemann

Department of Computer Science

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
    - $\forall x, y.\ \mathsf{Succ}(x) =_{\mathsf{Nat}} \mathsf{Succ}(y) \longleftrightarrow x =_{\mathsf{Nat}} y$
    - $\forall x.\ \neg\, \mathsf{Succ}(x) =_{\mathsf{Nat}} \mathsf{Zero}$
  - getting defining equations of functional programs as theorems
    - $\forall x, xs, ys.\, \mathsf{append}(\mathsf{Cons}(x, xs), ys) =_{\mathsf{List}} \mathsf{Cons}(x, \mathsf{append}(xs, ys))$
  - induction schemes
    - $\dfrac{\varphi(\mathsf{Zero}) \quad \forall x.\, \varphi(x) \longrightarrow \varphi(\mathsf{Succ}(x))}{\forall x.\, \varphi(x)}$

Notation – The Normal Form

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

$$t\downarrow := \text{the unique normal of } t \text{ w.r.t. } \hookrightarrow$$

- 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 τ is $\mathcal{T}(\mathcal{C})_\tau$, so $t \in \mathcal{T}(\mathcal{C})_\tau$ implies $t \in NF(\hookrightarrow)$
  - if $F \in \mathcal{C}$, then $F^{\mathcal{M}}(t_1, \dots, t_n) \stackrel{def}{=} F(t_1, \dots, t_n) = F(t_1, \dots, t_n)\downarrow$
  - if $F \in \mathcal{D}$, then $F^{\mathcal{M}}(t_1, \dots, t_n) \stackrel{def}{=} F(t_1, \dots, t_n)\downarrow$

The Substitution Lemma

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

$$[\![t\sigma]\!]_\alpha = [\![t]\!]_\beta$$

  where $\beta(x) := [\![\sigma(x)]\!]_\alpha$
- proof by structural induction on t
  - $[\![x\sigma]\!]_\alpha = [\![\sigma(x)]\!]_\alpha = \beta(x) = [\![x]\!]_\beta$
  - $$\begin{aligned} [\![F(t_1, \dots, t_n)\sigma]\!]_\alpha &= [\![F(t_1\sigma, \dots, t_n\sigma)]\!]_\alpha \\ &= F^{\mathcal{M}}([\![t_1\sigma]\!]_\alpha, \dots, [\![t_n\sigma]\!]_\alpha) \\ &\overset{IH}{=} F^{\mathcal{M}}([\![t_1]\!]_\beta, \dots, [\![t_n]\!]_\beta) \\ &= [\![F(t_1, \dots, t_n)]\!]_\beta \end{aligned}$$

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 $\alpha : \mathcal{V}_\tau \to \mathcal{T}(\mathcal{C})_\tau$ 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
  - $\llbracket x \rrbracket_\alpha = \alpha(x) \overset{(*)}{=} \alpha(x)\downarrow = x\alpha\downarrow$ where $(*)$ holds, since $\alpha(x) \in \mathcal{T}(\mathcal{C})$
  - $$\begin{aligned} \llbracket F(t_1,\ldots,t_n) \rrbracket_\alpha &= F^{\mathcal{M}}(\llbracket t_1 \rrbracket_\alpha,\ldots,\llbracket t_n \rrbracket_\alpha) \\ &\overset{IH}{=} F^{\mathcal{M}}(t_1\alpha\downarrow,\ldots,t_n\alpha\downarrow) = F(t_1\alpha\downarrow,\ldots,t_n\alpha\downarrow)\downarrow \\ &\overset{(confl.)}{=} F(t_1\alpha,\ldots,t_n\alpha)\downarrow = F(t_1,\ldots,t_n)\alpha\downarrow \end{aligned}$$

Defining Equations are Theorems in Standard Model

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

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

- theorem: if $\ell = r$ is defining equation of program (of type τ), 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 $=^{\mathcal{M}}_\tau$ we have to show $[\![\ell]\!]_\alpha = [\![r]\!]_\alpha$ for all α
  - via reverse substitution lemma this is equivalent to $\ell\alpha\downarrow = r\alpha\downarrow$
  - easily follows from confluence, since $\ell\alpha \hookrightarrow r\alpha$

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 φ
- added axioms are theorems of standard model, so they are consistent
- example $AX = \{\vec{\forall}\, \ell =_\tau r \mid \ell = r \text{ is def. eqn.}\}$
- 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 φ 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 φ does not hold!
  - example on next slide

Axiomatic Reasoning – Problematic Model

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

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

- we want to prove associativity of plus, so let φ be

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

- consider the following model $\mathcal{M}'$
  - $\mathcal{A}_{\mathsf{Nat}} = \mathbb{N} \cup \{x + \frac{1}{2} \mid x \in \mathbb{Z}\} = \{\dots, -1\frac{1}{2}, -\frac{1}{2}, 0, \frac{1}{2}, 1, 1\frac{1}{2}, 2, 2\frac{1}{2}, \dots\}$
  - $\mathsf{Zero}^{\mathcal{M}'} = 0$
  - $\mathsf{Succ}^{\mathcal{M}'}(n) = n + 1$
  - $\mathsf{plus}^{\mathcal{M}'}(n, m) = \begin{cases} n + m, & \text{if } n \in \mathbb{N} \text{ or } m \in \mathbb{N} \\ n - m + \frac{1}{2}, & \text{otherwise} \end{cases}$
  - ${=_{\mathsf{Nat}}}^{\mathcal{M}} = \{(n, n) \mid n \in \mathcal{A}_{\mathsf{Nat}}\}$
  - $\mathcal{M}' \models \bigwedge AX$, 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 α 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 φ such that $\mathcal{M} \models \varphi$, but $AX \not\models \varphi$
- note: adding φ to AX does not fix the problem, since then there is another formula φ' such that $\mathcal{M} \models \varphi'$ and $AX \cup \{\varphi\} \not\models \varphi'$
- consequence: "proving φ 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 $c : \tau_1 \times \ldots \times \tau_n \to \tau \in \mathcal{C}$ we define its decomposition theorem as

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

  and for all $d : \tau'_1 \times \ldots \times \tau'_k \to \tau \in \mathcal{C}$ with $c \neq d$ we define the disjointness theorem as

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

- proof of validity of decomposition theorem:

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

Axioms about Equality – Example

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

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

Induction Theorems

- current axioms are not even strong enough to prove simple theorems, e.g., $\forall x.\ \mathsf{plus}(x, \mathsf{Zero}) =_{\mathsf{Nat}} x$
- 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 φ

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

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

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

    is a valid theorem; quantifying over φ results in induction scheme

Induction Theorems – Example Instances

- induction scheme

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

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

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

- example with quantifiers and free variables:
  $\varphi(x) := \forall y.\, \mathsf{plus}(\mathsf{plus}(x, y), z) =_{\mathsf{Nat}} \mathsf{plus}(x, \mathsf{plus}(y, z))$

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

Preparing Induction Theorems – Substitutions in Formulas

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

Examples

- substitution of formulas
  - $(\forall x.\, \varphi)\sigma = \forall x.\, (\varphi\sigma)$ — if x does not occur in σ
  - $(\forall x.\, \varphi)\sigma = (\forall y.\, \varphi[x/y])\sigma$ — if x occurs in σ where y is fresh
- example substitution applications
  - $\varphi := \forall x.\, \neg\, x =_{\mathsf{Nat}} y$
  - $\varphi[y/\mathsf{Zero}] = \forall x.\, \neg\, x =_{\mathsf{Nat}} \mathsf{Zero}$ — no renaming required
  - $\varphi[y/\mathsf{Succ}(z)] = \forall x.\, \neg\, x =_{\mathsf{Nat}} \mathsf{Succ}(z)$ — no renaming required
  - $\varphi[y/\mathsf{Succ}(x)] = \forall z.\, \neg\, z =_{\mathsf{Nat}} \mathsf{Succ}(x)$ — renaming $[x/z]$ required
    without renaming meaning will change: $\forall x.\, \neg\, x =_{\mathsf{Nat}} \mathsf{Succ}(x)$
  - $\varphi[x/\mathsf{Succ}(y)] = \forall z.\, \neg\, z =_{\mathsf{Nat}} y$ — renaming $[x/z]$ required
    without renaming meaning will change: $\forall x.\, \neg\, \mathsf{Succ}(y) =_{\mathsf{Nat}} y$
- example theorems involving substitutions

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

Substitution Lemma for Formulas

- example induction formula

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

- 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 $\beta(x) := [\![\sigma(x)]\!]_\alpha$
- proof by structural induction on φ for arbitrary α and σ
  - $\mathcal{M} \models_\alpha P(t_1, \dots, t_n)\sigma$
    iff $\mathcal{M} \models_\alpha P(t_1\sigma, \dots, t_n\sigma)$
    iff $([\![t_1\sigma]\!]_\alpha, \dots, [\![t_n\sigma]\!]_\alpha) \in P^\mathcal{M}$
    iff $([\![t_1]\!]_\beta, \dots, [\![t_n]\!]_\beta) \in P^\mathcal{M}$
    iff $\mathcal{M} \models_\beta P(t_1, \dots, t_n)$
    where we use the substitution lemma of slide 5 to conclude $[\![t_i\sigma]\!]_\alpha = [\![t_i]\!]\beta$
  - $\mathcal{M} \models_\alpha (\neg\varphi)\sigma$ 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 $\beta(x) := [\![\sigma(x)]\!]_\alpha$
- proof by structural induction on φ for arbitrary α and σ
  - 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 $\beta'(z) := [\![([x/y]\sigma)(z)]\!]_{\alpha[y:=a]}$ (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 β' and $\beta[x := a]$ on variables of φ
    - $\beta'(x) = [\![([x/y]\sigma)(x)]\!]_{\alpha[y:=a]} = [\![\sigma(y)]\!]_{\alpha[y:=a]} = [\![y]\!]_{\alpha[y:=a]} = a$ and $\beta[x := a](x) = a$
    - z is variable of φ, $z \neq x$:
      by freshness condition conclude $z \neq y$ and $y \notin \mathcal{V}ars(\sigma(z))$; hence
      $\beta'(z) = [\![([x/y]\sigma)(z)]\!]_{\alpha[y:=a]} = [\![\sigma(z)]\!]_{\alpha[y:=a]} = [\![\sigma(z)]\!]_\alpha$ and
      $\beta[x := a](z) = \beta(z) = [\![\sigma(z)]\!]_\alpha$

Substitution Lemma in Standard Model

- substitution lemma: $\mathcal{M} \models_\alpha \varphi\sigma$ iff $\mathcal{M} \models_\beta \varphi$ where $\beta(x) := [\![\sigma(x)]\!]_\alpha$
- 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
  1. $\mathcal{M} \models_{\alpha[x:=t]} \varphi$ iff $\mathcal{M} \models_\alpha \varphi[x/t]$
  2. $\mathcal{M} \models_\alpha \forall x.\, \varphi$ iff $\mathcal{M} \models_\alpha \varphi[x/t]$ for all $t \in \mathcal{T}(\mathcal{C})_\tau$
- proof
  1. 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 $\beta(z) = [\![[x/t](z)]\!]_\alpha = \alpha[x := [\![t]\!]_\alpha](z)$
     iff $\mathcal{M} \models_{\alpha[x:=t]} \varphi$ $\qquad$ ($[\![t]\!]_\alpha = t$, since $t \in \mathcal{T}(\mathcal{C})$)
  2. 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 ψ for lists with fresh x, xs

$$\psi := \underbrace{\varphi[ys/\mathsf{Nil}]}_{1} \longrightarrow \underbrace{(\forall x, xs.\, \varphi[ys/xs] \longrightarrow \varphi[ys/\mathsf{Cons}(x, xs)])}_{2} \longrightarrow \forall ys.\, \varphi$$

- proof of $\mathcal{M} \models_\alpha \psi$:
  assume premises 1 ($\mathcal{M} \models_\alpha \varphi[ys/\mathsf{Nil}]$) 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 $\ell \in \mathcal{T}(\mathcal{C})_{\mathsf{List}}$";
  prove this statement by structural induction on lists
  - Nil: showing $\mathcal{M} \models_\alpha \varphi[ys/\mathsf{Nil}]$ is easy: it is exactly premise 1
  - $\mathsf{Cons}(n, \ell)$: use SL on premise 2 to conclude

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

    hence

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

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

Freshness of Variables

- example: structural induction formula for lists with fresh x, xs

$$\varphi[ys/\mathsf{Nil}] \longrightarrow (\forall x, xs.\, \varphi[ys/xs] \longrightarrow \varphi[ys/\mathsf{Cons}(x, xs)]) \longrightarrow \forall ys.\, \varphi$$

- why freshness required? isn't name of quantified variables irrelevant?
- problem: substitution is applied below quantifier!
- example: let us drop freshness condition and "prove" non-theorem

$$\mathcal{M} \models \forall x, xs, ys.\, ys =_{\mathsf{List}} \mathsf{Nil} \vee ys =_{\mathsf{List}} \mathsf{Cons}(x, xs)$$

  - by semantics of $\forall x, xs. \ldots$ it suffices to prove

$$\mathcal{M} \models_\alpha \forall ys.\, \underbrace{ys =_{\mathsf{List}} \mathsf{Nil} \vee ys =_{\mathsf{List}} \mathsf{Cons}(x, xs)}_{\varphi}$$

  - apply above induction formula and obtain two subgoals $\mathcal{M} \models_\alpha \ldots$ for
    - $\varphi[ys/\mathsf{Nil}]$ which is $\mathsf{Nil} =_{\mathsf{List}} \mathsf{Nil} \vee \mathsf{Nil} =_{\mathsf{List}} \mathsf{Cons}(x, xs)$
    - $\forall x, xs.\, \varphi[ys/xs] \longrightarrow \varphi[ys/\mathsf{Cons}(x, xs)]$ which is $\forall x, xs.\, \ldots \longrightarrow \mathsf{Cons}(x, xs) =_{\mathsf{List}} \mathsf{Nil} \vee \mathsf{Cons}(x, xs) =_{\mathsf{List}} \mathsf{Cons}(x, xs)$
- solution: rename variables in induction formula whenever required

Structural Induction Formula

- finally definition of induction formula for data structures is possible
- consider
$$\begin{aligned}\text{data } \tau = {} & c_1 : \tau_{1,1} \times \ldots \times \tau_{1,m_1} \to \tau \\ \mid {} & \ldots \\ \mid {} & c_n : \tau_{n,1} \times \ldots \times \tau_{n,m_n} \to \tau\end{aligned}$$
- let $x \in \mathcal{V}_\tau$, let φ be a formula, let variables $x_1, x_2, \ldots$ be fresh w.r.t. φ
- for each c_i define

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

- the induction formula is $\vec{\forall}\, (\varphi_1 \longrightarrow \ldots \longrightarrow \varphi_n \longrightarrow \forall x.\, \varphi)$
- theorem: $\mathcal{M} \models \vec{\forall}\, (\varphi_1 \longrightarrow \ldots \longrightarrow \varphi_n \longrightarrow \forall x.\, \varphi)$

Proof of Structural Induction Formula

- to prove: $\mathcal{M} \models \vec{\forall}\ (\varphi_1 \longrightarrow \ldots \longrightarrow \varphi_n \longrightarrow \forall x.\, \varphi)$
- $\forall$-intro: $\mathcal{M} \models_\alpha (\varphi_1 \longrightarrow \ldots \longrightarrow \varphi_n \longrightarrow \forall x.\, \varphi)$ for arbitrary α
- $\longrightarrow$-intro: assume $\mathcal{M} \models_{\alpha} \varphi_i$ for all i and show $\mathcal{M} \models_\alpha \forall x.\, \varphi$
- $\forall$-intro via SL: show $\mathcal{M} \models_\alpha \varphi[x/t]$ for all $t \in \mathcal{T}(\mathcal{C})_\tau$
- prove this by structural induction on t w.r.t. induction rule of $\mathcal{T}(\mathcal{C})_\tau$ (for precisely this α, not for arbitrary α)
- induction step for each constructor $c_i : \tau_{i,1} \times \ldots \times \tau_{i,m_i} \to \tau$
  - aim: $\mathcal{M} \models_\alpha \varphi[x/c_i(t_1, \ldots, t_{m_i})]$ IH: $\mathcal{M} \models_\alpha \varphi[x/t_j]$ for all j such that $\tau_{i,j} = \tau$
  - use assumption $\mathcal{M} \models_{\alpha} \varphi_i$, i.e., (here important: same α)

$$\mathcal{M} \models_\alpha \forall x_1, \ldots, x_{m_i}.\, (\bigwedge_{j, \tau_{i,j} = \tau} \varphi[x/x_j]) \longrightarrow \varphi[x/c_i(x_1, \ldots, x_{m_i})]$$

  - use SL as $\forall$-elimination with substitution $[x_1/t_1, \ldots, x_{m_i}/t_{m_i}]$, obtain

$$\mathcal{M} \models_\alpha (\bigwedge_{j, \tau_{i,j} = \tau} \varphi[x/t_j]) \longrightarrow \varphi[x/c_i(t_1, \ldots, t_{m_i})]$$

  - combination with IH yields desired $\mathcal{M} \models_\alpha \varphi[x/c_i(t_1, \ldots, t_{m_i})]$

Summary: Axiomatic Proofs of Functional Programs

- given a well-defined functional program, define a set of axioms AX consisting of
  - equations of defined symbols (slide 7)
  - axioms about equality of constructors (slide 11)
  - structural induction formulas (slide 22)
- instead of proving $\mathcal{M} \models \varphi$ deduce $AX \models \varphi$
- fact: standard model is ignored in previous step
- question: why all these efforts and not just state AX?
- reason:

  having proven $\mathcal{M} \models \psi$ for all $\psi \in AX$
  implies that AX is consistent!
- recall: already just converting functional program equations naively into theorems led to proof of $0 = 1$ on slide 1/20, i.e., inconsistent axioms,
  and AX now contains more complex axioms than just equalities

Example: Attempt to Prove Associativity of Append via AX

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

Equational Reasoning and Induction

Reasoning about Functional Programs: Current State

- given well-defined functional program, extract set of axioms AX that are satisfied in standard model $\mathcal{M}$
  - equations of defined symbols
  - equivalences regarding equality of constructors
  - structural induction formulas
- 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
  - $\forall xs.\ \mathsf{reverse}(\mathsf{reverse}(xs)) =_{\mathsf{List}} xs$
  - $\forall xs, ys.\ \mathsf{reverse}(\mathsf{append}(xs, ys)) =_{\mathsf{List}} \mathsf{append}(\mathsf{reverse}(ys), \mathsf{reverse}(xs))$
  - $\forall x, y.\ \mathsf{plus}(x, y) =_{\mathsf{Nat}} \mathsf{plus}(y, x)$

  but not
  - $\forall x.\ \exists y.\ \mathsf{greater}(y, x) =_{\mathsf{Bool}} \mathsf{True}$
- universally quantified axioms
  - equations of defined symbols
    - $\forall y.\ \mathsf{plus}(\mathsf{Zero}, y) =_{\mathsf{Nat}} y$
    - $\forall x, y.\ \mathsf{plus}(\mathsf{Succ}(x), y) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(x, y))$
    - $\dots$
  - axioms about equality of constructors
    - $\forall x, y.\ \mathsf{Succ}(x) =_{\mathsf{Nat}} \mathsf{Succ}(y) \longleftrightarrow x =_{\mathsf{Nat}} y$
    - $\forall x.\ \mathsf{Succ}(x) =_{\mathsf{Nat}} \mathsf{Zero} \longleftrightarrow \mathsf{false}$
    - $\dots$
  - but not: structural induction formulas
    - $\varphi[y/\mathsf{Zero}] \longrightarrow (\forall x.\ \varphi[y/x] \longrightarrow \varphi[y/\mathsf{Succ}(x)]) \longrightarrow \forall y.\ \varphi$

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

$$\frac{}{\mathsf{true} \wedge \varphi \rightsquigarrow_{AX} \varphi} \qquad \frac{}{\varphi \wedge \mathsf{true} \rightsquigarrow_{AX} \varphi} \qquad \frac{}{\mathsf{false} \wedge \varphi \rightsquigarrow_{AX} \mathsf{false}}$$

$$\frac{}{\neg\mathsf{false} \rightsquigarrow_{AX} \mathsf{true}} \qquad \frac{}{\neg\mathsf{true} \rightsquigarrow_{AX} \mathsf{false}}$$

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

$$\frac{\vec\forall\, (\ell =_\tau r \longleftrightarrow \varphi) \in AX}{\ell\sigma =_\tau r\sigma \rightsquigarrow_{AX} \varphi\sigma} \qquad \frac{}{t =_\tau t \rightsquigarrow_{AX} \mathsf{true}}$$

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

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 α
  - so in particular $\mathcal{M} \models \vec{\forall}\, \varphi \longleftrightarrow \psi$
- immediate consequence: $\varphi \rightsquigarrow^*_{AX}$ true implies $\mathcal{M} \models \vec{\forall}\, \varphi$
- define calculus: $\vdash \vec{\forall}\, \varphi$ if $\varphi \rightsquigarrow^*_{AX}$ true
- example

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

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

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

by induction on $\leadsto$ for arbitrary α

- case $\dfrac{\varphi \leadsto \varphi'}{\varphi \land \psi \leadsto \varphi' \land \psi}$
  - IH: $\mathcal{M} \models_\alpha \varphi \longleftrightarrow \varphi'$ for arbitrary α
  - conclude $\mathcal{M} \models_\alpha \varphi \land \psi$
    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 $\mathcal{M} \models_\alpha \varphi' \land \psi$
  - in total: $\mathcal{M} \models_\alpha \varphi \land \psi \longleftrightarrow \varphi' \land \psi$
- 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 $\dfrac{\vec{\forall}\,(\ell =_\tau r \longleftrightarrow \varphi) \in AX}{\ell\sigma =_\tau r\sigma \rightsquigarrow \varphi\sigma}$
  - premise $\mathcal{M} \models \vec{\forall}\,(\ell =_\tau r \longleftrightarrow \varphi)$,
    so in particular $\mathcal{M} \models_\beta \ell =_\tau r \longleftrightarrow \varphi$ for $\beta(x) = [\![\sigma(x)]\!]_\alpha$
  - conclude $\mathcal{M} \models_\alpha \ell\sigma =_\tau r\sigma$
    iff $[\![\ell]\!]_\beta = [\![r]\!]_\beta$ (by SL)
    iff $\mathcal{M} \models_\beta \varphi$ (by premise)
    iff $\mathcal{M} \models_\alpha \varphi\sigma$ (by SL)
  - in total: $\mathcal{M} \models_\alpha \ell\sigma =_\tau r\sigma \longleftrightarrow \varphi\sigma$

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

- case $\dfrac{\vec{\forall}\, \ell =_\tau r \in AX \quad s \hookrightarrow_{\{\ell = r\}} s'}{s =_\tau t \rightsquigarrow s' =_\tau t}$
  - premise $\mathcal{M} \models \vec{\forall}\, \ell =_\tau r$, 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 $[\![s]\!]_\alpha$
    $= [\![C[\ell\sigma]]\!]_\alpha$
    $= C[\ell\sigma]\alpha\!\downarrow$ (by reverse SL)
    $= C\alpha[\ell\sigma\alpha]\!\downarrow = C\alpha[\ell\sigma\alpha\!\downarrow]\!\downarrow$
    $\stackrel{(*)}{=} C\alpha[r\sigma\alpha\!\downarrow]\!\downarrow = C\alpha[r\sigma\alpha]\!\downarrow$
    $= C[r\sigma]\alpha\!\downarrow$
    $= [\![C[r\sigma]]\!]_\alpha$ (by reverse SL)
    $= [\![s']\!]_\alpha$
  - reason for $(*)$: premise implies
    $[\![\ell]\!]_\beta = [\![r]\!]_\beta$ for $\beta(x) = [\![\sigma(x)]\!]_\alpha$,
    hence $[\![\ell\sigma]\!]_\alpha = [\![r\sigma]\!]_\alpha$ (by SL),
    and thus, $\ell\sigma\alpha\!\downarrow = r\sigma\alpha\!\downarrow$ (by reverse SL)
  - in total: $\mathcal{M} \models_\alpha s =_\tau t \longleftrightarrow s' =_\tau t$

Comparing $\rightsquigarrow$ with $\hookrightarrow$

- $\hookrightarrow$ rewrites on terms whereas $\rightsquigarrow$ also simplifies Boolean connectives and uses axioms about equality $=_\tau$
- $\hookrightarrow$ uses defining equations of program whereas $\rightsquigarrow_{AX}$ is parametrized by set of axioms
  - in particular proven properties like $\forall xs.\ \mathsf{reverse}(\mathsf{reverse}(xs)) =_{\mathsf{List}} xs$ 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 $\forall xs.\ xs =_{\mathsf{List}} \mathsf{reverse}(\mathsf{reverse}(xs))$ 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: $\forall xs.\ \mathsf{reverse}(\mathsf{reverse}(xs)) =_{\mathsf{List}} xs$ is intended direction
    - direction sometimes not obvious for distributive laws

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

      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 $\forall xs.\ \mathsf{reverse}(\mathsf{reverse}(xs)) =_{\mathsf{List}} xs$
  - $\rightsquigarrow$ can not deal with induction axioms such as the one for associativity of append (app)

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

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

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

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

Induction in Combination with Equational Reasoning

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

$$\varphi_1 \longrightarrow \ldots \longrightarrow \varphi_n \longrightarrow \forall x.\, \varphi$$

    (no outer universal quantifier, since by construction above formula has no free variables)
  - try to prove each φ_i via $\rightsquigarrow$

Example: Associativity of Append

- aim: prove equality $\forall xs, ys, zs.\ \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))$
- approach:
  - select induction variable xs
  - reordering of quantifiers not required
  - the induction formula is presented on slide 35
  - φ_1 is

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

so we simply evaluate

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

Example: Associativity of Append, Continued

- proving $\forall xs, ys, zs.\ \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))$
- approach: ...
  - φ_2 is
$$\forall x, xs.(\forall ys, zs.\ \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))) \longrightarrow$$
$$(\forall ys, zs.\ \mathsf{app}(\mathsf{app}(\mathsf{Cons}(x, xs), ys), zs) =_{\mathsf{List}} \mathsf{app}(\mathsf{Cons}(x, xs), \mathsf{app}(ys, zs)))$$

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

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

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

Integrating IHs into Equational Reasoning

- recall structure of induction formula for formula φ 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 φ_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:
    $$\mathsf{app}(\mathsf{app}(\mathsf{Cons}(x, xs), ys), zs) =_{\mathsf{List}} \mathsf{app}(\mathsf{Cons}(x, xs), \mathsf{app}(ys, zs)) \rightsquigarrow^* \mathsf{true}$$
  - add IH $\forall ys, zs.\ \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))$ 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

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

  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 σ 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 $\forall xs, ys, zs.\ \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))$
- approach: ...
  - φ_2 is
$$\begin{aligned}&\forall x, xs.(\forall ys, zs.\ \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))) \longrightarrow\\ &\quad(\forall ys, zs.\ \mathsf{app}(\mathsf{app}(\mathsf{Cons}(x, xs), ys), zs) =_{\mathsf{List}} \mathsf{app}(\mathsf{Cons}(x, xs), \mathsf{app}(ys, zs)))\end{aligned}$$

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

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

    to the set of axioms (only for the proof of φ_2); then

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

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

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

Integrating IHs into Equational Reasoning, Soundness

- aim: prove $\mathcal{M} \models \varphi_i$ for
$$\varphi_i := \vec{\forall}\ \underbrace{\bigwedge_j \psi_j}_{\text{IHs}} \longrightarrow \psi$$
where we assume that $\psi \leadsto^* \mathsf{true}$ with the additional local axioms of the IHs ψ_j
- hence show $\mathcal{M} \models_\alpha \psi$ under the assumptions $\mathcal{M} \models_\alpha \psi_j$ for all IHs ψ_j
- by existing soundness proof of $\leadsto$ we can nearly conclude $\mathcal{M} \models_\alpha \psi$ from $\psi \leadsto^* \mathsf{true}$
- only gap: proof needs to cover new inference rules on slide 40

Soundness of Partially Quantified Equation Application

- case $\dfrac{\forall \vec{x}.\ \ell =_\tau r \in AX \quad s \hookrightarrow_{\{\ell = r\}} s'}{s =_\tau t \leadsto s' =_\tau t}$ with $\sigma(y) = y$ for all $y \notin \vec{x}$
  - premise is $\mathcal{M} \models_\alpha \forall \vec{x}.\ \ell =_\tau r$ (and not $\mathcal{M} \models \vec{\forall}\, \ell =_\tau r$) and $s = C[\ell\sigma]$ and $s' = C[r\sigma]$ as before
  - conclude $[\![s]\!]_\alpha = [\![s']\!]_\alpha$ 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 $[\![\ell]\!]_\beta = [\![r]\!]_\beta$ for $\beta(x) = [\![\sigma(x)]\!]_\alpha$
  - new proof
    - let $\vec{x} = x_1, \ldots, x_k$
    - premise implies $[\![\ell]\!]_{\alpha[x_1 := a_1, \ldots, x_k := a_k]} = [\![r]\!]_{\alpha[x_1 := a_1, \ldots, x_k := a_k]}$ for arbitrary a_i, so in particular for $a_i = [\![\sigma(x_i)]\!]_\alpha$
    - it now suffices to prove that $\alpha[x_1 := a_1, \ldots, x_k := a_k] = \beta$
    - consider two cases
    - for variables x_i we have
      $$\alpha[x_1 := a_1, \ldots, x_k := a_k](x_i) = a_i = [\![\sigma(x_i)]\!]_\alpha = \beta(x_i)$$
    - for all other variables $y \notin \vec{x}$ we have
      $$\alpha[x_1 := a_1, \ldots, x_k := a_k](y) = \alpha(y) = [\![y]\!]_\alpha = [\![\sigma(y)]\!]_\alpha = \beta(y)$$

Summary

- framework for inductive proofs combined with equational reasoning
- apply induction first
- then prove each case $\vec{\forall} \bigwedge \psi_j \longrightarrow \psi$ via evaluation $\psi \leadsto^* \mathsf{true}$ where IHs ψ_j become local axioms
- free variables in IHs (induction variables) may not be instantiated by $\leadsto$, 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

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

- formula

$$\vec{\forall}\, \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))$$

- induction on xs works successfully
- what about induction on ys (or zs)?
- base case already gets stuck

$$\mathsf{app}(\mathsf{app}(xs, \mathsf{Nil}), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(\mathsf{Nil}, zs))$$
$$\rightsquigarrow \mathsf{app}(\mathsf{app}(xs, \mathsf{Nil}), zs) =_{\mathsf{List}} \mathsf{app}(xs, zs)$$

- 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
$$\mathsf{plus}(\mathsf{Succ}(x), y) = \mathsf{Succ}(\mathsf{plus}(x, y))$$
$$\mathsf{plus}(\mathsf{Zero}, y) = y$$
- formula
$$\vec{\forall}\, \mathsf{plus}(x, y) =_{\mathsf{Nat}} \mathsf{plus}(y, x)$$
- let us try induction on x
- base case already gets stuck
$$\mathsf{plus}(\mathsf{Zero}, y) =_{\mathsf{Nat}} \mathsf{plus}(y, \mathsf{Zero})$$
$$\rightsquigarrow y =_{\mathsf{Nat}} \mathsf{plus}(y, \mathsf{Zero})$$
- final result suggests required lemma: Zero is also right neutral
- $\forall x.\ \mathsf{plus}(x, \mathsf{Zero}) =_{\mathsf{Nat}} x$ 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

$$\mathsf{plus}(\mathsf{Succ}(x), y) = \mathsf{Succ}(\mathsf{plus}(x, y))$$
$$\mathsf{plus}(\mathsf{Zero}, y) = y$$

- formula

$$\vec{\forall}\, \mathsf{plus}(x, \mathsf{Zero}) =_{\mathsf{Nat}} x$$

- only one possible induction variable: x
- base case:

$$\mathsf{plus}(\mathsf{Zero}, \mathsf{Zero}) =_{\mathsf{Nat}} \mathsf{Zero} \rightsquigarrow \mathsf{Zero} =_{\mathsf{Nat}} \mathsf{Zero} \rightsquigarrow \mathsf{true}$$

- step case adds IH $\mathsf{plus}(x, \mathsf{Zero}) =_{\mathsf{Nat}} x$ as axiom and we get

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

Commutativity of Addition

- formula

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

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

$$\begin{aligned}
&\mathsf{plus}(\mathsf{Succ}(x), y) =_{\mathsf{Nat}} \mathsf{plus}(y, \mathsf{Succ}(x))\\
\rightsquigarrow\ &\mathsf{Succ}(\mathsf{plus}(x, y)) =_{\mathsf{Nat}} \mathsf{plus}(y, \mathsf{Succ}(x))\\
\rightsquigarrow\ &\mathsf{Succ}(\mathsf{plus}(y, x)) =_{\mathsf{Nat}} \mathsf{plus}(y, \mathsf{Succ}(x))
\end{aligned}$$

- final result suggests required lemma: Succ on second argument can be moved outside
- $\forall x, y.\ \mathsf{plus}(x, \mathsf{Succ}(y)) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(x, y))$ 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

$$
\begin{aligned}
\mathsf{app}(\mathsf{Cons}(x, xs), ys) &= \mathsf{Cons}(x, \mathsf{app}(xs, ys)) \\
\mathsf{app}(\mathsf{Nil}, ys) &= ys \\
\mathsf{rev}(\mathsf{Cons}(x, xs)) &= \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, \mathsf{Nil})) \\
\mathsf{rev}(\mathsf{Nil}) &= \mathsf{Nil} \\
\mathsf{r}(\mathsf{Cons}(x, xs), ys) &= \mathsf{r}(xs, \mathsf{Cons}(x, ys)) \\
\mathsf{r}(\mathsf{Nil}, ys) &= ys \\
\mathsf{rev\_fast}(xs) &= \mathsf{r}(xs, \mathsf{Nil})
\end{aligned}
$$

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

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

- applying $\rightsquigarrow$ first yields desired lemma

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

Generalizations Required

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

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

- step-case gets stuck

$$\begin{aligned} & \mathsf{r}(\mathsf{Cons}(x, xs), \mathsf{Nil}) =_{\mathsf{List}} \mathsf{rev}(\mathsf{Cons}(x, xs)) \\ \leadsto^* \ & \mathsf{r}(xs, \mathsf{Cons}(x, \mathsf{Nil})) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, \mathsf{Nil})) \\ \leadsto \ & \mathsf{r}(xs, \mathsf{Cons}(x, \mathsf{Nil})) =_{\mathsf{List}} \mathsf{app}(\mathsf{r}(xs, \mathsf{Nil}), \mathsf{Cons}(x, \mathsf{Nil})) \end{aligned}$$

- 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

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

- creativity required

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

Fast Implementation of Reversal, Continued

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

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

- base-case

$$\begin{aligned} & \mathsf{r}(\mathsf{Nil}, ys) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(\mathsf{Nil}), ys) \\ \rightsquigarrow^* \ & ys =_{\mathsf{List}} ys \rightsquigarrow \mathsf{true} \end{aligned}$$

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

$$\begin{aligned} & \mathsf{r}(\mathsf{Cons}(x, xs), ys) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(\mathsf{Cons}(x, xs)), ys) \\ \rightsquigarrow\ & \mathsf{r}(xs, \mathsf{Cons}(x, ys)) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(\mathsf{Cons}(x, xs)), ys) \\ \rightsquigarrow\ & \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, ys)) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(\mathsf{Cons}(x, xs)), ys) \\ \rightsquigarrow\ & \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, ys)) =_{\mathsf{List}} \mathsf{app}(\mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, \mathsf{Nil})), ys) \\ \rightsquigarrow\ & \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, ys)) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(xs), \mathsf{app}(\mathsf{Cons}(x, \mathsf{Nil}), ys)) \\ \rightsquigarrow\ & \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, ys)) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, \mathsf{app}(\mathsf{Nil}, ys))) \\ \rightsquigarrow\ & \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, ys)) =_{\mathsf{List}} \mathsf{app}(\mathsf{rev}(xs), \mathsf{Cons}(x, ys)) \rightsquigarrow \mathsf{true} \end{aligned}$$

Fast Implementation of Reversal, Finalized

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

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

- then for our initial aim we get

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

- at this point one easily identifies a missing property

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

  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

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

- and the desired property

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

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

Induction w.r.t. Algorithm

- induction w.r.t. algorithm was informally performed on slide 4/36
  - 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

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

the induction rule for half is

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

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 φ be some formula which has exactly n free variables $x_1, \dots, x_n$
  - then the induction rule for f is

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

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

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

  where $\vec{\forall}$ ranges over all variables in the equation
- properties
  - $\mathcal{M} \models \varphi_{ind,f}$; reason: pattern-completeness and termination $(SN(\hookrightarrow \circ \trianglerighteq))$
  - heuristic: good idea to prove properties $\vec{\forall}\varphi$ about function f via $\varphi_{f,ind}$
  - reason: structure will always allow one evaluation step of f-invocation

Back to Example

- consider program

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

- for property

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

chose 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
$$\forall x.\ \mathsf{le}(\mathsf{half}(x), x) =_{\mathsf{Bool}} \mathsf{True}$$
turns this problem into three new proof obligations
  - $\mathsf{le}(\mathsf{half}(\mathsf{Zero}), \mathsf{Zero}) =_{\mathsf{Bool}} \mathsf{True}$
  - $\mathsf{le}(\mathsf{half}(\mathsf{Succ}(\mathsf{Zero})), \mathsf{Succ}(\mathsf{Zero})) =_{\mathsf{Bool}} \mathsf{True}$
  - $\mathsf{le}(\mathsf{half}(\mathsf{Succ}(\mathsf{Succ}(x))), \mathsf{Succ}(\mathsf{Succ}(x))) =_{\mathsf{Bool}} \mathsf{True}$
    where $\mathsf{le}(\mathsf{half}(x), x) =_{\mathsf{Bool}} \mathsf{True}$ can be assumed as IH
- the first two are easy, the third one works as follows
$$\begin{aligned} & \mathsf{le}(\mathsf{half}(\mathsf{Succ}(\mathsf{Succ}(x))), \mathsf{Succ}(\mathsf{Succ}(x))) =_{\mathsf{Bool}} \mathsf{True} \\ \rightsquigarrow\ & \mathsf{le}(\mathsf{Succ}(\mathsf{half}(x)), \mathsf{Succ}(\mathsf{Succ}(x))) =_{\mathsf{Bool}} \mathsf{True} \\ \rightsquigarrow\ & \mathsf{le}(\mathsf{half}(x), \mathsf{Succ}(x)) =_{\mathsf{Bool}} \mathsf{True} \end{aligned}$$
- here there is another problem, namely that the IH is not applicable
- problem solvable by proving an implication like
  $\mathsf{le}(x, y) =_{\mathsf{Bool}} \mathsf{True} \longrightarrow \mathsf{le}(x, \mathsf{Succ}(y)) =_{\mathsf{Bool}} \mathsf{True}$;
  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 $\psi \longrightarrow \varphi\sigma \rightsquigarrow^* \mathsf{true}$
- $\rightsquigarrow$ must be extended to perform more Boolean reasoning
- not done formally at this point

Equational Reasoning with Conditions, Example

- property

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

- apply induction on le
- first case

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

- second case

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

Equational Reasoning with Conditions, Example

- property

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

- third case has IH

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

  and we reason as follows

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

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

Final Example: Insertion Sort

- consider insertion sort

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

- 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

$$\ldots = \ldots$$
$$\mathsf{sort}(\mathsf{Cons}(x, xs)) = \mathsf{insort}(x, \mathsf{sort}(xs))$$

algorithm above, properties for specification below

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

- example properties (where $b =_{\mathsf{Bool}} \mathsf{True}$ is written just as b)
  - $\mathsf{sorted}(\mathsf{insort}(x, xs)) =_{\mathsf{Bool}} \mathsf{sorted}(xs)$
  - $\mathsf{sorted}(\mathsf{sort}(xs))$
- 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:

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

  speculative proofs are risky: conjectures might be wrong
- property $\forall xs.\ \mathsf{sorted}(\mathsf{sort}(xs))$ is shown by induction on xs
- base case:

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

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

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

Example: Soundness of insort

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

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

Example: Soundness of insort, Step Case

- prove $\forall x, xs.\ \mathsf{sorted}(\mathsf{insort}(x, xs)) =_{\mathsf{Bool}} \mathsf{sorted}(xs)$ by induction on xs
- step case with IH $\forall x.\ \mathsf{sorted}(\mathsf{insort}(x, ys)) =_{\mathsf{Bool}} \mathsf{sorted}(ys)$:

$$\begin{aligned} & \mathsf{sorted}(\mathsf{insort}(x, \mathsf{Cons}(y, ys))) =_{\mathsf{Bool}} \mathsf{sorted}(\mathsf{Cons}(y, ys)) \\ \rightsquigarrow\ & \mathsf{sorted}(\mathsf{if}(\mathsf{le}(x, y), \mathsf{Cons}(x, \mathsf{Cons}(y, ys)), \mathsf{Cons}(y, \mathsf{insort}(x, ys)))) =_{\mathsf{Bool}} \dots \end{aligned}$$

now perform case analysis on first argument of if

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

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

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

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

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

Example: Soundness of insort, Final Part

- prove $\forall x, xs.\ \mathsf{sorted}(\mathsf{insort}(x, xs)) =_{\mathsf{Bool}} \mathsf{sorted}(xs)$ by ind. on xs
- step case with IH $\forall x.\ \mathsf{sorted}(\mathsf{insort}(x, ys)) =_{\mathsf{Bool}} \mathsf{sorted}(ys)$:

$$\begin{aligned} & \mathsf{sorted}(\mathsf{insort}(x, \mathsf{Cons}(y, ys))) =_{\mathsf{Bool}} \mathsf{sorted}(\mathsf{Cons}(y, ys)) \\ \rightsquigarrow\ & \mathsf{sorted}(\mathsf{if}(\mathsf{le}(x, y), \mathsf{Cons}(x, \mathsf{Cons}(y, ys)), \mathsf{Cons}(y, \mathsf{insort}(x, ys)))) =_{\mathsf{Bool}} \dots \end{aligned}$$

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

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

  at this point identify further required auxiliary properties
    - $\vec{\forall}\ \mathsf{all\_le}(y, \mathsf{insort}(x, ys)) =_{\mathsf{Bool}} \mathsf{all\_le}(y, \mathsf{Cons}(x, ys))$
    - $\vec{\forall}\ \mathsf{le}(x, y) =_{\mathsf{Bool}} \mathsf{False} \longrightarrow \mathsf{le}(y, x) =_{\mathsf{Bool}} \mathsf{True}$

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

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 $c_i(x_1, \ldots, x_n)$ 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