universität innsbruck

SS 2021

Program Verification

Part 5 – Reasoning about Functional Programs

René Thiemann

Department of Computer Science

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
    - $\forall y.\ \mathsf{plus}(\mathsf{Zero}, y) =_{\mathsf{Nat}} y$
    - $\forall x, y.\ \mathsf{plus}(\mathsf{Succ}(x), y) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(x, y))$
    - ...
  - axioms about equality of constructors
    - $\forall x, y.\ \mathsf{Succ}(x) =_{\mathsf{Nat}} \mathsf{Succ}(y) \longleftrightarrow x =_{\mathsf{Nat}} y$
    - $\forall x.\ \mathsf{Succ}(x) =_{\mathsf{Nat}} \mathsf{Zero} \longleftrightarrow \mathsf{false}$
    - ...
  - 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 $\rightsquigarrow$: $\varphi \rightsquigarrow \psi$ implies $\mathcal{M} \models_\alpha \varphi \longleftrightarrow \psi$

by induction on $\rightsquigarrow$ for arbitrary α

- case $\dfrac{\varphi \rightsquigarrow \varphi'}{\varphi \wedge \psi \rightsquigarrow \varphi' \wedge \psi}$
  - IH: $\mathcal{M} \models_\alpha \varphi \longleftrightarrow \varphi'$ for arbitrary α
  - 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 $\dfrac{\vec\forall\,(\ell =_\tau r \longleftrightarrow \varphi) \in AX}{\ell\sigma =_\tau r\sigma \rightsquigarrow \varphi\sigma}$
  - 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)
    $= C\alpha[\ell\sigma\alpha]\Downarrow = C\alpha[\ell\sigma\alpha\Downarrow]\Downarrow$
    $\overset{(*)}{=} C\alpha[r\sigma\alpha\Downarrow]\Downarrow = C\alpha[r\sigma\alpha]\Downarrow$
    $= C[r\sigma]\alpha\Downarrow$
    $= [\![C[r\sigma]]\!]_\alpha$ (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 defined 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 $\leadsto$

- $\leadsto$ only works with universally quantified properties
  - defined equations
  - equivalences to simplify equalities $=_\tau$
  - newly derived properties such as $\forall xs.\ \mathsf{reverse}(\mathsf{reverse}(xs)) =_{\mathsf{List}} xs$
  - $\leadsto$ 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, $\leadsto$ 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 $\leadsto$ 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 wrt. slide 3/71

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

    (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 11
  - 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 $$\forall x, xs. (\forall ys, zs.\ \mathsf{app}(\mathsf{app}(xs, ys), zs) =_{\mathsf{List}} \mathsf{app}(xs, \mathsf{app}(ys, zs))) \longrightarrow$$ $$(\forall ys, zs.\ \mathsf{app}(\mathsf{app}(\mathsf{Cons}(x, xs), ys), zs) =_{\mathsf{List}} \mathsf{app}(\mathsf{Cons}(x, xs), \mathsf{app}(ys, zs)))$$

    so we try to prove the rhs of $\longrightarrow$ via $\rightsquigarrow$ 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 16

Soundness of Partially Quantified Equation Application

- case $\dfrac{\forall \vec{x}.\ \ell =_\tau r \in AX \quad s \hookrightarrow_{\{\ell = r\}} s'}{s =_\tau t \rightsquigarrow s' =_\tau t}$ 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 9 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 $\vec{x} = x_1, \dots, x_k$
    - premise implies $[\![\ell]\!]_{\alpha[x_1 := a_1, \dots, x_k := a_k]} = [\![r]\!]_{\alpha[x_1 := a_1, \dots, x_k := a_k]}$ for arbitrary a_i, so in particular for $a_i = [\![\sigma(x_i)]\!]_\alpha$
    - it now suffices to prove that $\alpha[x_1 := a_1, \dots, x_k := a_k] = \beta$
    - consider two cases
    - for variables x_i we have

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

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

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

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:

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

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

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

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

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

- 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

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

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

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

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 wrt. 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 wrt. Algorithm

- induction wrt. algorithm was informally performed on slides 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

$$\mathsf{half}(\mathsf{Zero}) = \mathsf{Zero}$$
$$\mathsf{half}(\mathsf{Succ}(\mathsf{Zero})) = \mathsf{Zero}$$
$$\mathsf{half}(\mathsf{Succ}(\mathsf{Succ}(x))) = \mathsf{Succ}(\mathsf{half}(x))$$

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 wrt. Algorithm

- induction wrt. algorithm formally defined
  - let f be n-ary defined function within well-defined program
  - let there be k defined 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

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

  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:

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

- 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 $\forall x, xs.\ \mathsf{sorted}(\mathsf{insort}(x, xs)) =_{\mathsf{Bool}} \mathsf{sorted}(xs)$ 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 $\forall x, xs.$ sorted(insort(x, xs)) $=_{\mathsf{Bool}}$ sorted(xs) by induction on xs
- step case with IH $\forall x.$ sorted(insort(x, ys)) $=_{\mathsf{Bool}}$ 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 le(x, y), i.e., le(x, y) $=_{\mathsf{Bool}}$ 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 $\forall x, xs.$ sorted(insort(x, xs)) $=_{\mathsf{Bool}}$ sorted(xs) by ind. on xs
- step case with IH $\forall x.$ sorted(insort(x, ys)) $=_{\mathsf{Bool}}$ 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 $\neg$le(x, y), i.e., le(x, y) $=_{\mathsf{Bool}}$ 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

    - $\vec{\forall}$ all_le(y, insort(x, ys)) $=_{\mathsf{Bool}}$ all_le(y, Cons(x, ys))
    - $\vec{\forall}$ le(x, y) $=_{\mathsf{Bool}}$ False $\longrightarrow$ le(y, x) $=_{\mathsf{Bool}}$ True

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

Summary

- equational properties can often conveniently be proved via induction and equational reasoning via $\rightsquigarrow$
- induction wrt. 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)