High Integrity Computing (F23PS2)

High Integrity Software Development (F29HD2)

Proof By Mathematical Induction

Andrew Ireland

Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Program Proof Motivations

- Programs are often specified in terms of recursively defined functions, *e.g.*

$$\begin{array}{rcl} odd(X) & \equiv & if\ \ X = 0\ \ then\ \ false \\ & & if\ \ X = 1\ \ then\ \ true \\ & & else\ \ odd(X-2) \end{array}$$

- Program proofs often require properties of the recursively defined functions, *e.g.*

$$\begin{array}{rcl} odd(X) & \leftrightarrow & \neg(even(X)) \\ even(X) & \leftrightarrow & odd(X+1) \end{array}$$

- Proving such properties requires **mathematical induction**.

More General Motivations

If you want to reason about:

- Recursive datatype, *e.g.* numbers, lists, sets, trees.
- Iterative code, *e.g.* proving a while-loop correct involves reasoning about arbitrary loop iterations.
- Electronic circuits with feedback loops.
- Parameterized designs, *e.g.* N-bit ripple-carry adder.
- Temporal aspects of reactive systems.

Then you require **proof by mathematical induction!**

Recursive Datatypes

- Defined in terms of base and step constructor functions.
- Natural numbers: $0 : nat$ and $s : nat \mapsto nat$

  $nat = \{0, s(0), s(s(0)), \ldots\}$
- Some examples:

$$\begin{array}{ll} s(s(s(0))) & (\equiv 3) \\ s(s(X)) & (\equiv X + 2) \end{array}$$

- Note that by using explicit constructor functions we have direct access to the recursive structure of the datatypes.
- Other recursive datatypes: Lists, Trees, *etc*

Functions via Recursive Equations

- **Addition:**

$$\begin{aligned} \forall Y : nat.\ 0 + Y &= Y \\ \forall X : nat.\ Y : nat.\ s(X) + Y &= s(X + Y) \end{aligned}$$

- **Even:**

$$\begin{aligned} even(0) &= true \\ even(s(0)) &= false \\ \forall X : nat.\ even(s(s(X))) &= even(X) \end{aligned}$$

- **Odd:**

$$\begin{aligned} odd(0) &= false \\ odd(s(0)) &= true \\ \forall X : nat.\ odd(s(s(X))) &= odd(X) \end{aligned}$$

Rewriting Proof Strategy

- The substitution proof rule coupled with matching provides a powerful proof strategy called **rewriting** that involves the application of **rewrite rules**:

  $C \rightarrow L \Rightarrow R$

  Note that the condition C may not occur.
- To be a valid rewrite rule there exists a **similarity relation** between L and R, *e.g.* the $\Rightarrow$ will be justified by an **equality**, **equivalence** or **implication**.
- Note that an equality/equivalence provides bi-directional rewrite rules, while an implication provides a rewrite rule which only applies to a goal in one direction, *i.e.* the opposite direction to the implication.

Generating Rewrite Rules

- **Multiplication:**

$$\begin{aligned} \forall Y : nat.\, 0 * Y &= 0 \\ \forall X : nat.\, Y : nat.\, s(X) * Y &= Y + (X * Y) \end{aligned}$$

- **Rewrite rules:**

$$\begin{aligned} 0 * Y &\Rightarrow 0 \\ s(X) * Y &\Rightarrow Y + (X * Y) \\ Y + (X * Y) &\Rightarrow s(X) * Y \end{aligned}$$

- The application of a rewrite rule requires that the left-hand side of the rule matches with a subterm of the goal.
- Note that all the match variables that appear on the right-hand side of a rewrite **must** also appear on the left-hand side. This means that $0 \Rightarrow 0 * Y$ is **NOT a rewrite rule!**

Proof by Rewriting

$$\begin{aligned} 0 + Y &\Rightarrow Y \qquad (1) \\ s(X) + Y &\Rightarrow s(X + Y) \end{aligned}$$

Givens:

Goal: $\underline{0 + (B + C)} = (0 + B) + C$

by (1)

$B + C = (\underline{0 + B}) + C$

by (1)

$B + C = B + C$

by reflexivity

true

How Rewriting Alone May Fail

$$\begin{aligned} 0 + Y &\Rightarrow Y \\ s(X) + Y &\Rightarrow s(X + Y) \end{aligned}$$

Givens:

Goal: $\forall A, B, C : nat.\ A + (B + C) = (A + B) + C$

The rewrite rules corresponding to the definition of addition are not applicable. Note that

$$\forall A, B, C : nat. \ldots$$

abbreviates

$$\forall A : nat.\ \forall B : nat.\ \forall C : nat. \ldots$$

Proof by Mathematical Induction

- If your goal is to prove $P(A)$, where A denotes a universally quantified variable of the natural numbers, then induction is required.
- The recursive structure of the natural numbers gives rise to a proof rule for induction of the form:

$$\frac{P(0) \quad \forall X : nat.\ (P(X) \rightarrow P(s(X)))}{\forall X : nat.\ P(X)}$$

- To prove $\forall A : nat.\ P(A)$ you must prove subgoals:

  Base case: $P(0)$

  Step case: Given $P(X)$ prove the goal $P(s(X))$

Note that $P(X)$ is known as the **induction hypothesis**.

Failed Proof Revisited

$$\frac{P(0) \quad \forall X : nat.\ (P(X) \rightarrow P(s(X)))}{\forall N : nat.\ P(N)}$$

Goal: $\forall A, B, C : nat.\ A + (B + C) = (A + B) + C$

By induction on A :

Base: $0 + (B + C) = (0 + B) + C$

Step: $\forall B', C' : nat.\ A + (B' + C') = (A + B') + C' \rightarrow$

$$s(A) + (B + C) = (s(A) + B) + C$$

Note: proof of base case corresponds to slide 8.

Step Case Proof

$$\begin{aligned} 0 + Y &\Rightarrow Y \\ s(X) + Y &\Rightarrow s(X + Y) \end{aligned} \tag{2}$$

Decomposing the step case goal gives:

Givens: $\forall B', C' : nat.\ A + (B' + C') = (A + B') + C'$

Goal: $s(A) + (B + C) = (s(A) + B) + C$

and the proof follows ...

Givens: $\forall B', C' : nat.\ A + (B' + C') = (A + B') + C'$

Goal:

$$\underline{s(A) + (B + C)} = (s(A) + B) + C$$

by (2)

$$s(A + (B + C)) = (\underline{s(A) + B}) + C$$

by (2)

$$s(A + (B + C)) = \underline{s(A + B) + C}$$

by (2)

$$s(\underline{A + (B + C)}) = s((A + B) + C)$$

by given left-to-right

$$s((A + B) + C) = s((A + B) + C)$$

by reflexivity

A High-Level Proof Pattern

$$\begin{aligned}
\forall B', C' : nat.\ A + (B' + C') &= (A + B') + C' \\
\boxed{s(A)}^{\uparrow} + (B + C) &= (\boxed{s(A)}^{\uparrow} + B) + C \\
\boxed{s(A + (B + C))}^{\uparrow} &= (\boxed{s(A)}^{\uparrow} + B) + C \\
\boxed{s(A + (B + C))}^{\uparrow} &= (\boxed{s(A + B)}^{\uparrow}) + C \\
\boxed{s(A + (B + C))}^{\uparrow} &= \boxed{s((A + B) + C)}^{\uparrow} \\
s((A + B) + C) &= s((A + B) + C)
\end{aligned}$$

Note that the rewriting **preserves** the parts of the goal that correspond to the given **induction hypothesis** while **removing** the **differences**, *i.e.* enabling a match between goal and given induction hypothesis (see `http://www.rippling.org/`).

Inductive Proof Challenges

- **Conjecture generalization**, *e.g.* to prove:

$$\forall A : nat.\ A + (A + A) = (A + A) + A$$

  requires us to generalize, *e.g.*

$$\forall A, B : nat.\ A + (B + B) = (A + B) + B$$

- **Lemma discovery**, *e.g.* to prove:

$$A^{B*C} = A^{B^C}$$

  requires us to discover and prove:

$$A * (B + C) = (A * B) + (A * C)$$

- **Induction selection**
- **Rewriting**
- **Induction hypothesis instantiation**

Induction Variable Selection

$$\forall A, B, C : nat.$$
$$A + (B + C) = (A + B) + C$$

$$\swarrow \qquad\qquad \downarrow \qquad\qquad \searrow$$

$$\boxed{\mathrm{A}} + (B + C) = (\boxed{\mathrm{A}} + B) + C \qquad A + (\boxed{\mathrm{B}} + C) = (A + \boxed{\mathrm{B}}) + C \qquad A + (B + \boxed{\mathrm{C}}) = (A + B) + \boxed{\mathrm{C}}$$

Induction Rule Selection

$\boxed{s(\boxed{N})}^{\uparrow}$

$$\frac{\begin{array}{l} P(0) \\ \forall N : nat.\ (P(N) \rightarrow P(s(N))) \end{array}}{\forall X : nat.\ P(X)}$$

$\boxed{s(s(\boxed{N}))}^{\uparrow}$

$$\frac{\begin{array}{l} P(0) \\ P(s(0)) \\ \forall N : nat.\ (P(N) \rightarrow P(s(s(N)))) \end{array}}{\forall X : nat.\ P(X)}$$

$\boxed{\boxed{M} + \boxed{N}}^{\uparrow}$

$$\frac{\begin{array}{l} P(0) \\ P(s(0)) \\ \forall M, N : nat.\ (P(M) \land P(N) \rightarrow P(M + N)) \end{array}}{\forall X : nat.\ P(X)}$$

$\vdots \qquad \vdots$

Induction Selection: +-associativity

$$\forall A, B, C : nat.\ \boxed{\mathrm{A}} + (B + C) = (\boxed{\mathrm{A}} + B) + C$$

- A on the left-hand side occurs as the first argument of $+$ so we are looking for a rewrite rule that:
  - removes term structure on the first argument of $+$ and
  - preserves the pattern $A + (B + C) = \ldots$

  the definition of $+$ provides such a rewrite rule, *i.e.*

  $$\boxed{s(\boxed{X})}^{\uparrow} + Y \Rightarrow \boxed{s(\boxed{X + Y})}^{\uparrow}$$

  so this occurrence of A is good for a $\boxed{s(\boxed{A})}^{\uparrow}$ induction.
- The same **rewrite rule analysis** suggests that the A on the right-hand side is also good for a $\boxed{s(\boxed{A})}^{\uparrow}$ induction.
- **All occurrences of A are good candidates.**

Induction Selection: +-associativity

$$\forall A, B, C : nat.\ A + (\boxed{\text{B}} + C) = (A + \boxed{\text{B}}) + C$$

- B on the left-hand side occurs as the first argument of $+$ so we are looking for rewrite rule that:
  - removes term structure on the first argument of $+$ and
  - preserves the pattern $A + (B + C) = \ldots$

  the definition of $+$ provides such a rewrite rule, *i.e.*

  $$\boxed{s(\boxed{X})}^{\uparrow} + Y \Rightarrow \boxed{s(\boxed{X + Y})}^{\uparrow}$$

  so this occurrence of B is good for a $\boxed{s(\boxed{B})}^{\uparrow}$ induction.
- However the same **rewrite rule analysis** fails for B on the right-hand, *i.e.* B occurs in the second argument position.
- **Some occurrences of B are good candidates.**

Induction Selection: +-associativity

$$\forall A, B, C : nat.\ A + (B + \boxed{\text{C}}) = (A + B) + \boxed{\text{C}}$$

- C on the left-hand side occurs as the second argument of $+$ so we are looking for rewrite rule that:
  - removes term structure on the second argument of $+$ and
  - preserves the pattern $A + (B + C) = \ldots$

  No such rewrite rule exists.
- The **rewrite rule analysis** fails for C on the right-hand for the same reason.
- **No occurrences of C are good candidates.**

Another Example

- **Goal:**

$$\forall A : nat.\ odd(A) \leftrightarrow \neg(even(A))$$

- **Rewrite rules:**

$$\begin{array}{rcl} even(0) & \Rightarrow & true \\ even(s(0)) & \Rightarrow & false \\ even(s(s(X))) & \Rightarrow & even(X) \end{array}$$

$$\begin{array}{rcl} odd(0) & \Rightarrow & false \\ odd(s(0)) & \Rightarrow & true \\ odd(s(s(X))) & \Rightarrow & odd(X) \end{array}$$

Note that these are a subset of the rewrite rules that are generated from the definitions given on slide 5.

Induction Selection: *odd-even*

$$\forall A : nat.\ odd(\boxed{\mathrm{A}}) \leftrightarrow \neg(even(\boxed{\mathrm{A}}))$$

- A on the left-hand side occurs as the argument of *odd* so we are looking for a rewrite rule that:
  - removes term structure on the argument of *odd* and
  - preserves the pattern $odd(A) \leftrightarrow \ldots$

  the definition of *odd* provides such a rewrite rule, *i.e.*

$$odd(\ s(s(\underline{X}))^{\uparrow}\ ) \Rightarrow odd(X)$$

so this occurrence of A is good for a $s(s(\underline{A}))^{\uparrow}$ induction.

Induction Selection: *odd-even*

$$\forall A : nat.\ odd(\boxed{\mathrm{A}}) \leftrightarrow \neg(even(\boxed{\mathrm{A}}))$$

- A on the right-hand side occurs as the argument of *even* so we are looking for a rewrite rule that:
  - removes term structure on the argument of *even* and
  - preserves the pattern $\ldots \leftrightarrow \neg(even(A))$

  the definition of *even* provides such a rewrite rule, *i.e.*

  $$even(\ \boxed{s(s(\boxed{X}))}^{\uparrow}\ ) \Rightarrow even(X)$$

  so this occurrence of A is good for a $\boxed{s(s(\boxed{A}))}^{\uparrow}$ induction.
- **All occurrences of A are good candidates.**

Induction Proof: *odd-even*

$$\frac{P(0) \quad P(s(0)) \quad \forall X : nat.\ (P(X) \rightarrow P(s(s(X))))}{\forall N : nat.\ P(N)}$$

Goal: $\forall A : nat.\ odd(\boxed{\mathrm{A}}) \leftrightarrow \neg(even(\boxed{\mathrm{A}}))$

By induction on A :

Base 1: $odd(0) \leftrightarrow \neg(even(0))$

Base 2: $odd(s(0)) \leftrightarrow \neg(even(s(0)))$

Step: $odd(A) \leftrightarrow \neg(even(A)) \rightarrow$
$odd(s(s(A))) \leftrightarrow \neg(even(s(s(A))))$

Exercise: complete proofs of the base and step cases.

And Finally ...

Consider the following definitions and properties:

- **Addition:**

$$
\begin{aligned}
\forall Y : nat.\ 0 + Y &= Y \\
\forall X : nat.\ Y : nat.\ s(X) + Y &= s(X + Y)
\end{aligned}
$$

- **Add:**

$$
\begin{aligned}
\forall Y : nat.\ add(0, Y) &= Y \\
\forall X : nat.\ Y : nat.\ add(s(X), Y) &= add(X, s(Y))
\end{aligned}
$$

- **Properties:**

$$
\begin{aligned}
\forall Y : nat.\ Y + 0 &= Y \\
\forall X : nat.\ Y : nat.\ X + s(Y) &= s(X + Y)
\end{aligned}
$$

And the corresponding rewrites rules ...

Rewrite Rules

$$
\begin{aligned}
0 + Y &\Rightarrow Y \\
Y &\Rightarrow 0 + Y \\
s(X) + Y &\Rightarrow s(X + Y) \qquad (3) \\
s(X + Y) &\Rightarrow s(X) + Y \\
add(0, Y) &\Rightarrow Y \\
Y &\Rightarrow add(0, Y) \\
add(s(X), Y) &\Rightarrow add(X, s(Y)) \qquad (4) \\
add(X, s(Y)) &\Rightarrow add(s(X), Y) \\
Y + 0 &\Rightarrow Y \\
Y &\Rightarrow Y + 0 \\
X + s(Y) &\Rightarrow s(X + Y) \\
s(X + Y) &\Rightarrow X + s(Y) \qquad (5)
\end{aligned}
$$

Induction Hypothesis Instantiation

- **Goal:**
$$\forall A, B : nat.\ add(A, B) = A + B$$
- **Step:**
Given:
$$\forall B' : nat.\ add(A, B') = A + B' \quad (6)$$
Goal:
$$\begin{array}{rcl} add(s(A), B) & = & s(A) + B \\ \vdots & & \vdots \\ add(A, s(B)) & = & A + s(B) \quad (7) \end{array}$$

Note: matching goal (7) with the given induction hypothesis (6) involves instantiating B' to be $s(B)$.

Filling In The Gaps

Givens: $\forall B' : nat.\ add(A, B') = A + B'$

Goal:
$$\begin{array}{rcll} \underline{add(s(A), B)} & = & s(A) + B & \\ & & & \text{by } (4) \\ add(A, s(B)) & = & \underline{s(A) + B} & \\ & & & \text{by } (3) \\ add(A, s(B)) & = & \underline{s(A + B)} & \\ & & & \text{by } (5) \\ add(A, s(B)) & = & A + s(B) & \\ & & & \text{by given IH, where} \\ & true & & B' \text{ becomes } s(B) \end{array}$$

Note: IH denotes Induction Hypothesis.

High-Level Proof Pattern Again

$$
\begin{array}{rcl}
\forall B' : nat.\ add(A, B') & = & A + B' \\
\\
add(\boxed{s(\boxed{A})}^{\uparrow}, B) & = & \boxed{s(\boxed{A})}^{\uparrow} + B \\
add(A, \boxed{s(\boxed{B})}^{\downarrow}) & = & \boxed{s(\boxed{A})}^{\uparrow} + B \\
add(A, \boxed{s(\boxed{B})}^{\downarrow}) & = & \boxed{s(\boxed{A+B})}^{\uparrow} \\
add(A, \boxed{s(\boxed{B})}^{\downarrow}) & = & A + \boxed{s(\boxed{B})}^{\downarrow} \\
 & true &
\end{array}
$$

Note that the rewriting **preserves** the parts of the goal that correspond to the given **induction hypothesis** while **removing** the **differences**, *i.e.* enabling a match between goal and given induction hypothesis.

Learning Outcomes

- The importance of proof by mathematical induction when reasoning about recursively defined structures.
- How to formalize and prove simple properties of recursively defined functions using mathematical induction.
- Challenges of inductive proof:
  - Conjecture generalization and lemma discovery
  - Induction selection
  - Guidance for rewriting
  - Induction hypothesis instantiation

Recommended Reading

Bundy, A. *The Computer Modelling of Mathematical Reasoning*, Academic Press, 1983. **(chapter 11)**

Bundy, A. and Basin, D. and Hutter, D. and Ireland, A. *Rippling: Meta-level Guidance for Mathematical Reasoning*, Cambridge Univeristy Press, 2005.

Broda, K. Eisenbach, S. Khoshnevisan, H. Vikers, S. *Reasoned Programming*, Prentice Hall, 1994. **(chapter 5)**