Automated Reasoning

Lecture 10: Inductive Proof (in Isabelle)

Robert Atkey
bob.atkey@ed.ac.uk

based on originals by Alan Bundy

Friday 13th February 2015
Recap

- Previously:
  - Unification and Rewriting

- This time: Proof by Induction (in Isabelle)
  - Proof by Mathematical Induction
  - Structural Recursion and Induction
  - Challenges in Inductive Proof Automation
A Summation Problem

What is

\[ 1 + 2 + 3 + \ldots + 999 + 1000 \]

? 

Is there a general formula for any \( n \)?
A Summation Problem

What is

$$1 + 2 + 3 + \ldots + 999 + 1000$$

Is there a general formula for any $n$?

Gauss’s solution:

$$1 + 2 + \ldots + n = \frac{n(n + 1)}{2}$$
A Summation Problem

What is

\[ 1 + 2 + 3 + \ldots + 999 + 1000 \]

Is there a general formula for any \( n \)?

Gauss’s solution:

\[ 1 + 2 + \ldots + n = \frac{n(n + 1)}{2} \]

How can we prove this? (Automatically?)

- First-order proof search is (generally) unable to prove this
- Rewriting and Model Checking (later in the course!) don’t help
Proof by Induction

To prove $\forall n. \ P(n)$:

\[
\begin{array}{ll}
\text{(base)} & \text{prove } P(0) \\
\text{(step)} & \text{for all } n, \text{ assume } P(n) \text{ and prove } P(n+1)
\end{array}
\]
Proof by Induction

To prove $\forall n. \; P(n)$:

\[ \begin{cases} 
\text{(base)} & \text{prove } P(0) \\
\text{(step)} & \text{for all } n, \text{ assume } P(n) \text{ and prove } P(n+1) 
\end{cases} \]

To prove $\forall n. \; 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$:
Proof by Induction

To prove $\forall n. P_n$: 

\begin{align*}
\text{(base)} & \text{ prove } P_0 \\
\text{(step)} & \text{ for all } n, \text{ assume } P_n \text{ and prove } P(n+1)
\end{align*}

To prove $\forall n. 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$:

(base): $0 = \frac{0 \cdot 1}{2}$, by computation.
Proof by Induction

To prove $\forall n. \ P(n)$:

\[
\begin{align*}
\text{(base)} & \quad \text{prove } P(0) \\
\text{(step)} & \quad \text{for all } n, \text{ assume } P(n) \text{ and prove } P(n+1)
\end{align*}
\]

To prove $\forall n. \ 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$:

\text{(base): } 0 = \frac{0 \cdot 1}{2}, \text{ by computation.}

\text{(step): assume the formula holds for } n, \text{ and:}

\[
\begin{align*}
1 + 2 + \ldots + n + (n+1) \\
= (1 + 2 + \ldots + n) + (n+1) \\
= \frac{n(n+1)}{2} + (n+1) \quad \text{(apply induction hypothesis)} \\
= \ldots \\
= \frac{(n+1)(n+2)}{2}
\end{align*}
\]

as required.
Inductively Defined Data

Induction is especially useful for dealing with Inductive Datatypes.

Inductive Datatypes are freely generated by some constructors.

\[
\text{datatype } \text{nat} = \text{Zero} \mid \text{Succ } \text{nat}
\]

\[
\text{datatype } 'a \text{ list} = \text{Nil} \mid \text{Cons } 'a \ text{ 'a list}
\]

Some values: \(\{\text{Succ (Succ Zero)}\text{ i.e. “2”}, \text{Cons Zero (Cons Zero Nil)}\text{ i.e. “[0, 0]”}\}\)

In general:

\[
\text{datatype } ('a \to ty = C_1 \ t_{11} \ \ldots \ t_{1n_1} | \ldots | C_m \ t_{m1} \ \ldots \ t_{mn_m}
\]

where \(ty\) must appear strictly positively (never to the left of a \(\Rightarrow\)).
Functions can be defined by recursion on “structurally smaller” data.

```plaintext
primrec length :: "'a list ⇒ nat"
where
"length Nil = Zero" | 
"length (Cons x xs) = Succ (length xs)"

primrec append :: "'a list ⇒ 'a list ⇒ 'a list"
where
"append Nil ys = ys" | 
"append (Cons x xs) ys = Cons x (append xs ys)"

primrec reverse :: "'a list ⇒ 'a list"
where
"reverse Nil = Nil" | 
"reverse (Cons x xs) = append (reverse xs) (Cons x Nil)"
```
**Proof by Structural Induction**

Properties of structurally recursive functions can be proved by structural induction.

To show \( \forall xs. P xs \):

\[
\begin{align*}
\text{prove } & P \text{ Nil} \\
\text{for all } & x, xs, \text{ assume } P \text{ xs to prove } P \text{ (Cons } x \text{ xs)}
\end{align*}
\]
Proof by Structural Induction

Properties of structurally recursive functions can be proved by structural induction.

To show $\forall xs. \ P \ xs$: 

\[
\begin{cases}
\text{prove } P \ Nil \\
\text{for all } x, \ xs, \ assume \ P \ xs \ to \ prove \ P \ (\text{Cons } x \ xs)
\end{cases}
\]

To prove: $\text{append } xs \ (\text{append } ys \ zs) = \text{append } (\text{append } xs \ ys) \ zs$:
Proof by Structural Induction

Properties of structurally recursive functions can be proved by structural induction.

To show \( \forall xs. \ P \ xs \): \[
\begin{cases}
\text{prove } P \text{ Nil} \\
\text{for all } x, xs, \text{ assume } P \ xs \text{ to prove } P \ (\text{Cons } x \ xs)
\end{cases}
\]

To prove: append \( xs \) (append \( ys \) \( zs \)) = append (append \( xs \) \( ys \)) \( zs \):

(base) append \( \text{Nil} \) (append \( \text{xs} \) \( \text{ys} \)) = append \( \text{xs} \) \( \text{ys} \) = append (append \( \text{Nil} \) \( \text{xs} \)) \( \text{ys} \)
Proof by Structural Induction

Properties of structurally recursive functions can be proved by structural induction.

To show \( \forall xs. P xs \): \( \begin{cases} 
\text{prove } P \text{ Nil} \\
\text{for all } x, xs, \text{ assume } P xs \text{ to prove } P (\text{Cons } x xs) 
\end{cases} \)

To prove: \( \text{append } xs (\text{append } ys zs) = \text{append } (\text{append } xs ys) zs \):

(base) \( \text{append } \text{Nil} (\text{append } xs ys) = \text{append } xs ys \)

(\( = \) \text{append } (\text{append } \text{Nil } xs) ys \)

(step) \( \text{append } (\text{Cons } x xs) (\text{append } xs ys) \)

(\( = \) \text{Cons } x (\text{append } xs (\text{append } xs ys))) \)

(\( = \) \text{Cons } x (\text{append } (\text{append } xs ys) zs) \text{ by IH} \)

(\( = \) \text{append } (\text{Cons } x (\text{append } xs ys)) zs \)

(\( = \) \text{append } (\text{append } (\text{Cons } x xs) ys) zs \)
Proof by Structural Induction

Properties of structurally recursive functions can be proved by structural induction.

To show $\forall xs. P xs$: 
- prove $P \text{Nil}$
- for all $x, xs$, assume $P xs$ to prove $P (\text{Cons} \ x \ xs)$

To prove: $\text{append} \ xs \ (\text{append} \ ys \ zs) = \text{append} \ (\text{append} \ xs \ ys) \ zs$:

(base) $\text{append} \ \text{Nil} \ (\text{append} \ xs \ ys) = \text{append} \ xs \ ys$

(step) $\text{append} \ (\text{Cons} \ x \ xs) \ (\text{append} \ xs \ ys)$

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

$= \text{Cons} \ x \ (\text{append} \ (\text{append} \ xs \ ys) \ zs)$ by IH

$= \text{append} \ (\text{Cons} \ x \ (\text{append} \ xs \ ys)) \ zs$

$= \text{append} \ (\text{append} \ (\text{Cons} \ x \ xs) \ ys) \ zs$

In practice: start with the equation to be proved as the goal, and rewrite both sides to be equal.
Well-Founded Induction

Let $<$ be an ordering on a set such that, for all $x$, there are no infinite downward chains:

Not allowed: $\ldots < \ldots < x_3 < x_2 < x_1 < x$

Such an ordering is called well-founded (or noetherian)
Well-Founded Induction

Let $<$ be an ordering on a set such that, for all $x$, there are no infinite downward chains:

Not allowed: $\ldots < \ldots < x_3 < x_2 < x_1 < x$

Such an ordering is called well-founded (or noetherian)

Then, to prove $\forall x. \ P \ x$, it suffices to prove:

$$\forall y. \ (\forall z. \ z < y \rightarrow P \ z) \rightarrow P \ y$$
Well-Founded Induction

Let $<$ be an ordering on a set such that, for all $x$, there are no infinite downward chains:

Not allowed: $\ldots < \ldots < x_3 < x_2 < x_1 < x$

Such an ordering is called well-founded (or noetherian)

Then, to prove $\forall x. P x$, it suffices to prove:

$$\forall y. (\forall z. z < y \rightarrow P z) \rightarrow P y$$

Specialised to the natural numbers, with the usual less-than ordering, this is usually called Complete Induction.
Theoretical Limitations of Automated Inductive Proof

Recall $L$-systems, with left- and right-introduction rules:

\[
\frac{\Gamma, P, Q \vdash R}{\Gamma, P \land Q \vdash R} \quad \text{(e conjE)} \quad \frac{\Gamma \vdash P}{\Gamma \vdash P \lor Q} \quad \text{(disjI1)} \quad \frac{\Gamma \vdash P \quad \Gamma, P \vdash Q}{\Gamma \vdash Q} \quad \text{(cut)}
\]

This system has two nice properties:

1. **Cut elimination**: the cut rule is unnecessary

2. **Sub-formula property**: every cut-free proof only contains formulas which are sub-formulas of the original goal

   \(Q(t)\) is a sub-formula of \(\forall x. Q(x)\) and \(\exists x. Q(x)\), for any \(t\)

So can do complete (but possibly non-terminating) proof search.
Theoretical Limitations of Automated Inductive Proof

Recall $L$-systems, with left- and right-introduction rules:

\[
\begin{align*}
\Gamma, P, Q &\vdash R \quad (e \text{ conjE}) \\
\Gamma, P \land Q &\vdash R \\
\Gamma &\vdash P \quad (\text{disjI1}) \\
\Gamma &\vdash P \lor Q \\
\Gamma &\vdash Q \quad (\text{cut})
\end{align*}
\]

This system has two nice properties:

1. Cut elimination: the cut rule is unnecessary
2. Sub-formula property: every cut-free proof only contains formulas which are sub-formulas of the original goal

$(Q(t)$ is a sub-formula of $\forall x. Q(x)$ and $\exists x. Q(x)$, for any $t$)

So can do complete (but possibly non-terminating) proof search.

If we add an induction rule:

\[
\begin{align*}
\Gamma &\vdash P(0) \\
\Gamma, P(n) &\vdash P(n + 1) \\
\Gamma &\vdash \forall n. P(n)
\end{align*}
\]

Then Cut elimination fails!

There are variant rules that bring it back, but sub-formula property still fails
The Need for Intermediate Lemmas

Practically, the lack of a guarantee of a proof with the sub-formula property means that we need *creative generalisation* during proofs, or we need to *speculate new lemmas.*
The Need for Intermediate Lemmas

Practically, the lack of a guarantee of a proof with the sub-formula property means that we need *creative generalisation* during proofs, or we need to *speculate new lemmas*.

To prove: $\text{reverse} \ (\text{reverse} \ xs) = xs$
The Need for Intermediate Lemmas

Practically, the lack of a guarantee of a proof with the sub-formula property means that we need *creative generalisation* during proofs, or we need to *speculate new lemmas*.

To prove: $\text{reverse} \ (\text{reverse} \ xs) = xs$

(base) $\text{reverse} \ (\text{reverse} \ \text{Nil}) = \text{reverse} \ \text{Nil} = \text{Nil}$
The Need for Intermediate Lemmas

Practically, the lack of a guarantee of a proof with the sub-formula property means that we need *creative generalisation* during proofs, or we need to *speculate new lemmas*.

To prove: \( \text{reverse (reverse } \text{xs}) = \text{xs} \)

**(base)** reverse (reverse Nil) = reverse Nil = Nil

**(step)** IH: reverse (reverse xs) = xs

Attempt: reverse (reverse (Cons x xs))
\[
= \text{reverse (append (reverse xs) (Cons x Nil))}
\]
\[
= \text{Cons x xs}
\]
The Need for Intermediate Lemmas

Practically, the lack of a guarantee of a proof with the sub-formula property means that we need *creative generalisation* during proofs, or we need to *speculate new lemmas*.

To prove: reverse (reverse $xs$) = $xs$

(base) reverse (reverse Nil) = reverse Nil = Nil

(step) IH: reverse (reverse $xs$) = $xs$

Attempt: reverse (reverse (Cons $x$ $xs$))

= reverse (append (reverse $xs$) (Cons $x$ Nil))

????

= Cons $x$ $xs$

We need to *speculate* a new lemma.
A New Lemma

In this case, it turns out that we need:

\[
\text{reverse } (\text{append } xs \ ys) = \text{append } (\text{reverse } ys) (\text{reverse } xs)
\]

(which is proved by induction, and needs \textit{another} lemma)
A New Lemma

In this case, it turns out that we need:

\[ \text{reverse} \left( \text{append} \ x s \ y s \right) = \text{append} \left( \text{reverse} \ y s \right) \left( \text{reverse} \ x s \right) \]

(which is proved by induction, and needs *another* lemma)

Now we can proceed:

**(step) IH:** \( \text{reverse} \left( \text{reverse} \ x s \right) = x s \)

*Attempt:*

\[ \text{reverse} \left( \text{reverse} \left( \text{Cons} \ x \ x s \right) \right) \]
\[ = \text{reverse} \left( \text{append} \left( \text{reverse} \ x s \right) \left( \text{Cons} \ x \ Nil \right) \right) \]
\[ = \text{append} \left( \text{Cons} \ x \ Nil \right) \left( \text{reverse} \left( \text{reverse} \ x s \right) \right) \text{ by lemma} \]
\[ = \text{Cons} \ x \left( \text{append} \ Nil \left( \text{reverse} \left( \text{reverse} \ x s \right) \right) \right) \]
\[ = \text{Cons} \ x \left( \text{reverse} \left( \text{reverse} \ x s \right) \right) \]
\[ = \text{Cons} \ x \ x s \text{ by IH} \]
Another approach

We got stuck trying to prove:

\[
\text{reverse } (\text{append } (\text{reverse } x) \ (\text{Cons } x \ \text{Nil})) = \text{Cons } x \ x
\]

under the assumption that \(\text{reverse } (\text{reverse } x) = x\)
Another approach

We got stuck trying to prove:

\[
\text{reverse} \ (\text{append} \ (\text{reverse} \ xs) \ (\text{Cons} \ x \ \text{Nil})) = \text{Cons} \ x \ xs
\]

under the assumption that \(\text{reverse} \ (\text{reverse} \ xs) = xs\)

What if we rewrite the RHS \textit{backwards} by the IH, to get the new goal:

\[
\text{reverse} \ (\text{append} \ (\text{reverse} \ xs) \ (\text{Cons} \ x \ \text{Nil})) = \text{Cons} \ x \ (\text{reverse} \ (\text{reverse} \ xs))
\]

Maybe this can be proved by induction?
Another approach

We got stuck trying to prove:

\[ \text{reverse} \left( \text{append} \left( \text{reverse} \ x s \right) \left( \text{Cons} \ x \ \text{Nil} \right) \right) = \text{Cons} \ x \ x s \]

under the assumption that \( \text{reverse} \left( \text{reverse} \ x s \right) = x s \)

What if we rewrite the RHS \textit{backwards} by the IH, to get the new goal:

\[ \text{reverse} \left( \text{append} \left( \text{reverse} \ x s \right) \left( \text{Cons} \ x \ \text{Nil} \right) \right) = \text{Cons} \ x \left( \text{reverse} \left( \text{reverse} \ x s \right) \right) \]

Maybe this can be proved by induction?

Not quite (try it and see!); need to \textit{generalise} and prove:

\[ \text{reverse} \left( \text{append} \ x s \left( \text{Cons} \ x \ \text{Nil} \right) \right) = \text{Cons} \ x \left( \text{reverse} \ x s \right) \]

(A special case of the lemma speculated earlier)
Challenges in Automating Inductive Proofs

Theoretically, and practically, to do inductive proofs, we need:

▶ Lemma speculation
▶ Generalisation

Techniques (other than “Get the user to do it”):

▶ Boyer-Moore approach
   roughly the approach described here (implemented in ACL2)
▶ Rippling, “Productive Use of Failure” (Bundy and Ireland, 1996)
▶ Up-front speculation:
   e.g. “maybe this binary function is associative?”
▶ Cyclic proofs
   (search for a circular proof, and afterwards prove it is well-founded)
▶ Only doing a few cases (0, 1, ..., 6)
▶ Special purpose techniques (e.g., generating functions)
Summary

- Proof by Induction (in Isabelle)
  - Natural number induction
  - Inductive Datatypes and Structural Induction (H&R 1.4.2)
  - The need for generalisation and lemma speculation
- No Lectures next week! (ILW)
- Tuesday 24th February 2015: Introduction to Model Checking:
  - So far the question has been: Is it the case that $M \models P$, for all $M$?
  - Model checking: Does $M \models P$, for fixed $M$ and $P$.
  - Temporal Logic.