CIS 301: Lecture Notes on Induction

Torben Amtoft
Department of Computing and Information Sciences
Kansas State University

April 29, 2004

These notes are written as a supplement to [1, Sect. 16.1&16.3], but can be read independently.

1 Demystifying Induction

Consider a loop of the form `while` B `do` C `od`, and assume that we know[1]

ψ is established by the preamble of the loop (1)

if with B true, ψ holds prior to C, then ψ also holds after C (2)

Then we can infer that

ψ is an invariant of the loop. (3)
That is, each time control reaches B, ψ holds.

[1]Using the framework of *CIS 301: Lecture Notes on Program Verification (Part I)*, available at `http://www.cis.ksu.edu/~tamtoft/CIS301/Spring04/verificationI.pdf`, this amounts to (1) and (2) below being valid annotations.

$\{\psi\}$	(1)
`while` B `do`	
$\{\psi \wedge B\}$	**WhileTrue**
C	
$\{\psi\}$	(2)
`od`	

Now observe that[2]

- (1) amounts to saying that ψ holds after 0 iterations;
- (2) amounts to saying that if ψ holds after k iterations, then after $k+1$ iterations ψ also holds;
- (3) amounts to saying that after any number (≥ 0) of loop iterations, ψ holds.

We thus have the following

Principle 1 (Induction on Iterations). *Assume that for a given loop,*

- *ψ holds after 0 iterations; and*
- *if ψ holds after k iterations then, after $k+1$ iterations, ψ also holds.*

Then, for all $k \geq 0$, after k iterations, ψ will hold.

2 Induction on Natural Numbers

Principle 1 carries over to a general principle:

Principle 2 (Induction on Natural Numbers). *Assume Q is such that*

- *$Q(0)$ holds; and*
- *for all natural numbers k, if $Q(k)$ holds then also $Q(k+1)$ holds.*

Then, for all natural numbers k, $Q(k)$ holds.

This is the rule mentioned in [1, p. 454]; in Fitch format, it can be written

$$\begin{array}{r|l} & Q(0) \\ & \vdots \\ & \forall k((\mathsf{Nat}(k) \wedge Q(k)) \to Q(k+1)) \\ & \vdots \\ \triangleright & \forall k(\mathsf{Nat}(k) \to Q(k)) \end{array}$$

[2]When we say that ψ holds after k iterations, we mean that *if* the loop iterates at least k times then ψ holds after the k'th iteration. If control exits from the loop earlier, or if one of the first k iterations gives rise to infinite computation (due to a subloop), then it is vacuously true that "ψ holds after k iterations".

Here Nat is a predicate that is true on exactly the numbers $0, 1, 2, 3, 4, \ldots$.

It is instructive to note that a sentence $\forall k(\mathsf{Nat}(k) \rightarrow Q(k))$ might also be provable using a "General Conditional Proof":

$$\begin{array}{r|l} & \begin{array}{|l} \boxed{k}\ \mathsf{Nat}(k) \\ \hline \vdots \\ Q(k) \end{array} \\ \rhd & \forall k(\mathsf{Nat}(k) \rightarrow Q(k)) \end{array}$$

But such an approach is less likely to succeed, since when proving $Q(k)$ for an arbitrary k, we now *cannot* assume $Q(k-1)$. On the other hand, a general conditional proof may be the only way to establish $\forall k(P(k) \rightarrow Q(k))$ in the case where the objects satisfying P do not have any "structure".

Example 2.1 ([1, p. 454]). We want to prove that for all natural numbers n we have

$$1 + \cdots + n = \tfrac{n(n+1)}{2}$$

With LHS and RHS given by

$$\begin{array}{rcl} \mathrm{LHS}(n) & = & 1 + \cdots + n \\ \mathrm{RHS}(n) & = & \dfrac{n(n+1)}{2} \end{array}$$

the claim is that for all natural numbers n we have $Q(n)$ where $Q(n)$ is given by $\mathrm{LHS}(n) = \mathrm{RHS}(n)$. We prove that by induction:

<u>Basis step.</u> We must establish $Q(0)$, that is $\mathrm{LHS}(0) = \mathrm{RHS}(0)$. Since $\mathrm{LHS}(0) = 0$ (as the sum of zero numbers is 0), this follows since $\mathrm{RHS}(0) = \frac{0 \cdot 1}{2} = 0$.

<u>Inductive step.</u> We can assume that $Q(n)$ holds, that is $\mathrm{LHS}(n) = \mathrm{RHS}(n)$, and must prove that $Q(n+1)$ holds. But this follows since

$$\begin{array}{rcl} \mathrm{LHS}(n+1) & = & (1 + \cdots + n) + (n+1) = \mathrm{LHS}(n) + (n+1) \\ & = & \mathrm{RHS}(n) + (n+1) = \dfrac{n(n+1)}{2} + (n+1) \\ & = & \dfrac{n(n+1) + 2(n+1)}{2} = \dfrac{(n+1)(n+2)}{2} \\ & = & \mathrm{RHS}(n+1) \end{array}$$

where the third equality follows from the induction hypothesis. □

2.1 Alternative formulations

We could choose another starting point:

$$
\begin{array}{r|l}
 & Q(1) \\
 & \vdots \\
 & \forall k((\mathsf{Nat}(k) \land Q(k)) \to Q(k+1)) \\
 & \vdots \\
\rhd & \forall k(\mathsf{Nat}(k) \to Q(k))
\end{array}
$$

assuming that Nat does hold only on $1, 2, 3, 4, \ldots$ but not on 0. But it is important to be *consistent*, unlike the formulation of Proposition 4 in [1, p.454] which—even though 0 just earlier on the page has been declared the first natural number—implicitly assumes that the first n natural numbers are $1, \ldots, n$.

A somewhat different perspective is offered by the following rule[3], called *course-of-values* induction:

$$
\begin{array}{r|l}
 & \\
 & \forall k(\forall m(m < k \to Q(m)) \to Q(k)) \\
 & \vdots \\
\rhd & \forall k Q(k)
\end{array}
$$

To justify the validity of this rule, assume (in order to arrive at a contradiction) that the conclusion does *not* hold. That is, there exists natural numbers not satisfying Q. Let k be the *least* such number. That is, for all $m < k$ we have $Q(m)$. But then our premise tells us that also $Q(k)$, yielding the desired contradiction. (A variation of this proof, where we do a proof by cases depending on whether $k = 0$ or $k > 0$, can be used to establish the validity of the original induction principle.)

Note that to establish the premise needed for course-of-values induction, a proof of the following form is probably needed:

$$
\begin{array}{|l}
\begin{array}{|l}
\boxed{k}\ \forall m(m < k \to Q(m)) \\
\hline
\vdots \\
Q(k)
\end{array} \\
\forall k(\forall m(m < k \to Q(m)) \to Q(k))
\end{array}
$$

[3]For simplicity, we implicitly assume that all entities are natural numbers.

We now give an example that illustrates the usefulness of course-of-values induction. Consider the Fibonacci numbers given by

$$\begin{array}{rcll} \mathrm{fib}(0) & = & 1 & \\ \mathrm{fib}(1) & = & 1 & \\ \mathrm{fib}(n) & = & \mathrm{fib}(n-1)+\mathrm{fib}(n-2) & \qquad \text{for } n \geq 2 \end{array}$$

Theorem 2.2. *If $n+1$ is divisible by 3, then* $\mathrm{fib}(n)$ *is an even number, otherwise* $\mathrm{fib}(n)$ *is an odd number.* □

Proof. We shall employ course-of-values induction; there is thus no base step but "only" the inductive step where we have to establish that for an arbitrary n, if (with Q the property mentioned in the theorem) $Q(m)$ holds for all $m < n$ then also $Q(n)$ holds. We have to do a case analysis.

Case 1: $n = 0$ or $n = 1$. Then $n + 1$ is not divisible by 3, and accordingly $\mathrm{fib}(n) = 1$ which is odd.

Case 2: $n + 1$ is divisible by 3. Then neither $(n-1)+1$ nor $(n-2)+1$ is divisible by 3. Our induction hypothesis thus tells us that $\mathrm{fib}(n-1)$ and $\mathrm{fib}(n-2)$ are both odd. As $\mathrm{fib}(n) = \mathrm{fib}(n-1)+\mathrm{fib}(n-2)$, this implies that $\mathrm{fib}(n)$ is even, as desired.

Case 3: $n > 2$ and $n + 1$ is not divisible by 3. Then exactly one of $(n-1)+1$ and $(n-2)+1$ is divisible by 3. Our induction hypothesis then tells us that exactly one of $\mathrm{fib}(n-1)$ and $\mathrm{fib}(n-2)$ is even. As $\mathrm{fib}(n) = \mathrm{fib}(n-1) + \mathrm{fib}(n-2)$, this implies that $\mathrm{fib}(n)$ is odd, as desired. □

Note that the last two steps could not have been carried out using the original principle of induction (Principle 2), where we in order to establish $Q(n)$ can assume only $Q(n-1)$ but not $Q(n-2)$.

3 Induction on Lists

Lists, a very important data structure, are inductively defined as follows:

base clause: $\mathsf{List}(\mathsf{nil})$ holds;

inductive clause: if $\mathsf{List}(x)$ and v is a value then also $\mathsf{List}(v \copyright x)$.

That is, a list is either empty (nil), or a value v in front of a list; here values could be natural numbers but also characters etc. (and they could even be lists themselves!)

Example 3.1. Consider a list with the elements 5,7,4 (note that the order matters). This list is in our syntax written as

$$5 \copyright (7 \copyright (4 \copyright \mathsf{nil}))$$

or graphically

```
  C
 / \
5   C
   / \
  7   C
     / \
    4  nil
```

$\square$

The induction principle for lists is (using Fitch format)

$$\begin{array}{r|l} & Q(\mathsf{nil}) \\ & \vdots \\ & \forall x \forall v((\mathsf{List}(x) \land Q(x)) \to Q(v \copyright x)) \\ & \vdots \\ \rhd & \forall x(\mathsf{List}(x) \to Q(x)) \end{array}$$

So in order to show that a property holds for all lists, one must show that it holds for the empty list, and that the property holds for a non-empty list provided it holds for its "tail". We call that induction principle *structural induction.*

To justify the validity of this rule, assume (in order to arrive at a contradiction) that the conclusion does *not* hold. That is, there exists lists not satisfying Q. Let x be among the "shortest" such lists (there might be several choices). That is, for all y such that y is shorter than x we have $Q(y)$.

We shall do a case analysis depending on whether x is nil or not, in both cases arriving at a contradiction. If $x = \mathsf{nil}$ the contradiction comes since our first premise then tells us that $Q(x)$ does hold. If x is of the form $v \copyright y$, then y is shorter than x so $Q(y)$ holds, which by our second premise implies that also $Q(x)$ holds, yielding the desired contradiction.

Definition 3.2. The *append* function, taking two lists x and y as arguments and returning their concatenation $x +\!\!+\, y$ (also a list), is given by the following recursive definition

$$\begin{array}{rcl} \mathsf{nil} +\!\!+\, y &=& y \\ (v \copyright x) +\!\!+\, y &=& v \copyright (x +\!\!+\, y) \end{array} \qquad \Box$$

For example, we have

$$\begin{array}{rl} & (5 \copyright (7 \copyright \mathsf{nil})) +\!\!+\, (8 \copyright (4 \copyright \mathsf{nil})) \\ = & 5 \copyright ((7 \copyright \mathsf{nil}) +\!\!+\, (8 \copyright (4 \copyright \mathsf{nil}))) \\ = & 5 \copyright (7 \copyright (\mathsf{nil} +\!\!+\, (8 \copyright (4 \copyright \mathsf{nil})))) \\ = & 5 \copyright (7 \copyright (8 \copyright (4 \copyright \mathsf{nil}))) \end{array}$$

By definition, nil is a "left neutral element" for the append function. We shall now show that it is also a right neutral element.

Theorem 3.3. *For all lists x, we have $Q(x)$, given by $x +\!\!+\, \mathsf{nil} = x$.* $\Box$

Proof. Structural induction. For the basis step, we must establish $Q(\mathsf{nil})$, that is

$$\mathsf{nil} +\!\!+\, \mathsf{nil} = \mathsf{nil}$$

which is trivial from Definition 3.2.

For the inductive step, we can assume $Q(x)$ and must show $Q(v \copyright x)$, which follows from the calculation

$$(v \copyright x) +\!\!+\, \mathsf{nil} = v \copyright (x +\!\!+\, \mathsf{nil}) = v \copyright x$$

Here we used Definition 3.2 for the first equality, and the induction hypothesis for the second equality. $\Box$

We can also prove that the append function is *associative*:

Theorem 3.4. *For all x,y,z, we have $(x +\!\!+\, y) +\!\!+\, z = x +\!\!+\, (y +\!\!+\, z)$.* $\Box$

Proof. We do structural induction on x: for given y and z, we define $Q(x)$ as the predicate $(x \mathbin{+\!\!+} y) \mathbin{+\!\!+} z = x \mathbin{+\!\!+} (y \mathbin{+\!\!+} z)$.

For the basis step, we must establish Q(nil), that is

$$(\mathsf{nil} \mathbin{+\!\!+} y) \mathbin{+\!\!+} z = \mathsf{nil} \mathbin{+\!\!+} (y \mathbin{+\!\!+} z)$$

which follows as both left hand side and right hand side reduces to $y \mathbin{+\!\!+} z$.

For the inductive step, we can assume $Q(x)$, and must establish $Q(v \text{ © } x)$, which follows from the calculation

$$\begin{array}{rcl}
((v \text{ © } x) \mathbin{+\!\!+} y) \mathbin{+\!\!+} z & = & \text{(Definition 3.2)} \\
(v \text{ © } (x \mathbin{+\!\!+} y)) \mathbin{+\!\!+} z & = & \text{(Definition 3.2)} \\
v \text{ © } ((x \mathbin{+\!\!+} y) \mathbin{+\!\!+} z) & = & \text{(Induction hypothesis)} \\
v \text{ © } (x \mathbin{+\!\!+} (y \mathbin{+\!\!+} z)) & = & \text{(Definition 3.2, backwards)} \\
(v \text{ © } x) \mathbin{+\!\!+} (y \mathbin{+\!\!+} z) & &
\end{array}$$

□

Note that induction in y or in z would not have worked.

3.1 Other kinds of structural induction

In Sect. 2.1, we saw that for natural numbers, "course-of-values induction" is often more applicable than the standard induction principle. Similarly, for other inductively defined data structures (like binary trees), it is often more convenient to apply the following induction principle

$$\begin{array}{l|l}
 & \forall x(\forall y(y \text{ “smaller than” } x \to Q(y)) \to Q(x)) \\
 & \vdots \\
\rhd & \forall x Q(x)
\end{array}$$

The unspecific "smaller than" can be defined in a numerous ways. For binary trees, one could for instance say that "y is smaller than x" iff y has fewer nodes than x.

References

[1] Jon Barwise and John Etchemendy. *Language, Proof and Logic.* CSLI Publications, 1999.