15–150: Principles of Functional Programming

Some Notes on Induction

Michael Erdmann*
Spring 2013

These notes provide a brief introduction to induction for proving properties of SML programs. We assume that the reader is already familiar with SML and the notes on evaluation for pure SML programs.

Recall that we write $e \stackrel{k}{\Longrightarrow} e'$ for a computation of k steps, $e \Longrightarrow e'$ for a computation of any number of steps (including 0), $e \hookrightarrow v$ for a complete computation of e to a value v, and $n = m$ or $e = e'$ for mathematical equality.

1 Lemmas

When proving the correctness of SML programs, we usually establish the correctness of the functions we define in sequence. The correctness property of a function corresponds to a lemma we can use in proving the correctness of later functions.

There is another form of lemma, which is a mathematical property which is necessary for a particular step in a program correctness proof. Usually, we take mathematical properties for granted and concentrate on the program property.

As a simple example, consider the function

```
fun square(n:int):int = n * n
```

It is trivial to see that this function correctly implements the squaring function $f(n) = n^2$. In assignments we would take such properties for granted, but if we wanted to establish the correctness formally, we would proceed as follows.

Lemma 1 *For every integer n, `square(n)` $\hookrightarrow n^2$.*

Proof: We prove this directly, relying on the operational semantics of SML.

	`square(`n`)`	
$\stackrel{1}{\Longrightarrow}$	n `*` n	by evaluation rule for function application
$\stackrel{1}{\Longrightarrow}$	$n \times n$	by evaluation rule for `*`
$=$	n^2	by mathematical property

□

*Modified from a draft by Frank Pfenning for 15-212, 1997.

2 Mathematical Induction

The simplest form of induction over natural numbers $0, 1, \ldots$ is *mathematical induction*, also called *standard or weak induction*. Assume we have to prove a property for every natural number n. We first prove that the property holds for 0 (induction basis). Then we assume the property holds for n and establish it for $n+1$ (induction step). Basis and step together guarantee that the property holds for all natural numbers.

There are small variations of this scheme which can be justified easily and which we also call mathematical induction. For example, the induction might start at 1 if we want to prove a property of all positive integers, or there might be two base cases for 0 and 1. A distinguishing characteristic of mathematical induction is the step from n to $n+1$.

As an example, we consider a straightforward, but inefficient function to compute n^k for n an arbitrary integer, and k a natural number. We take $0^0 = 1$.

```
(* power : (int * int) -> int
   REQUIRES: k >= 0
   ENSURES: power(n,k) ==> n^k, with 0^0 = 1.
*)

fun power(n:int, k:int):int =
    if k = 0 then 1 else n * power(n, k-1)
```

Theorem 2 *`power(n, k)` $\hookrightarrow n^k$ for $k \geq 0$ and all integers `n`.*

Proof: By mathematical induction on k.

Base Case: $k = 0$.

$$
\begin{array}{ll}
 & \texttt{power}(n, 0) \\
\stackrel{1}{\Longrightarrow} & \texttt{if 0 = 0 then 1 else } n \texttt{ * power}(n, 0\texttt{-}1) \\
\stackrel{1}{\Longrightarrow} & \texttt{if true then 1 else } n \texttt{ * power}(n, 0\texttt{-}1) \\
\stackrel{1}{\Longrightarrow} & 1 \\
= & n^0
\end{array}
$$

Induction Step: Prove for $k+1$, with $k \geq 0$.

Inductive hypothesis: Assume that for some $k \geq 0$, and all integers n, `power(`n`, `k`)` $\hookrightarrow n^k$.

We need to show that: `power(`n`, `$k+1$`)` $\hookrightarrow n^{k+1}$.

Evaluating code, we see that:

$$
\begin{array}{lll}
 & \texttt{power}(n, k+1) & \\
\stackrel{1}{\Longrightarrow} & \texttt{if } k+1 \texttt{ = 0 then 1 else } n \texttt{ * power}(n, k+1\texttt{-}1) & \\
\stackrel{1}{\Longrightarrow} & \texttt{if false then 1 else } n \texttt{ * power}(n, k+1\texttt{-}1) & \text{since } k \geq 0 \\
\stackrel{1}{\Longrightarrow} & n \texttt{ * power}(n, k+1\texttt{-}1) & \\
\stackrel{1}{\Longrightarrow} & n \texttt{ * power}(n, k) & \\
\Longrightarrow & n \texttt{ * } n^k & \text{by inductive hypothesis} \\
\stackrel{1}{\Longrightarrow} & n \times n^k & \\
= & n^{k+1} &
\end{array}
$$

□

The level of detail in a proof generally depends on the context in which the proof is carried out and the mathematical sophistication of the expected reader. In homework assignments you should feel free to omit the number of computation steps (unless we are investigating computational complexity) and combine obvious steps. Appeals to the inductive hypothesis or other non-obvious steps should be justified as in the example above.

Your primary concern should be the appropriateness of the induction principle you use and the correctness of the individual steps (even if not all details are given).

3 Strong Induction

The principle of *strong induction* (also called *complete induction and course-of-values induction*) formalizes a frequent pattern of reasoning. It can be justified entirely from the principle of mathematical induction, but it is useful enough to be stated as another admissible induction principle.

To prove a property by strong induction on a variable n, we first establish the induction basis for the base case $n = 0$. Then we prove the induction step for $n > 0$ by assuming the property for all $n' < n$ and establishing it for n. One can think of it like mathematical induction, except that we are allowed to appeal to the inductive hypothesis for any $n' < n$ and not just the immediate predecessor.

As an example we write a more efficient implementation of the `power` function which requires fewer recursive calls. It uses the `square` function above, and one new auxiliary function to test if a number is even:

```
fun even(n:int):bool = (n mod 2 = 0)
```

We assume without proof that `even`$(n) \hookrightarrow$ `true` if n is even, and `even`$(n) \hookrightarrow$ `false` if n is odd. We also depart from the first definition of `power` in that we use pattern matching to define the function.

```
(* power : (int * int) -> int
   REQUIRES: k >= 0
   ENSURES: power(n,k) ==> n^k, with 0^0 = 1.
*)

fun power(n, 0) = 1
  | power(n, k) =          (* k > 0 *)
    if even k then square(power(n, k div 2))
              else n * power(n, k-1)
```

When we compute the application of a function defined by pattern matching to an argument, we do not consider the sequential matching of the argument value against the patterns as separate steps. However, to understand a proof it may be important to note why a particular case matched or did not match the arguments.

Theorem 3 *`power(`n`, `k`)`* $\hookrightarrow n^k$ *for* $k \geq 0$ *and all integers* n.

Proof: By strong induction on k.

Base Case: $k = 0$.

$$\begin{array}{lll} & \texttt{power(}n\texttt{, 0)} & \\ \stackrel{1}{\Longrightarrow} & 1 & \text{by evaluation, including pattern matching} \end{array}$$

Induction Step: $k > 0$.

Inductive Hypothesis: Assume that $\texttt{power(}n\texttt{, }k'\texttt{)} \hookrightarrow n^{k'}$ for all $k' < k$ and all integers `n`.

We need to show that: $\texttt{power(}n\texttt{, }k\texttt{)} \hookrightarrow n^k$.

Evaluating code, we see that:

$$\begin{array}{lll} & \texttt{power(}n\texttt{, }k\texttt{)} & \\ \stackrel{1}{\Longrightarrow} & \texttt{if even } k \texttt{ then square(power(}n\texttt{, }k \texttt{ div 2))} & \\ & \qquad \texttt{else } n \texttt{ * power(}n\texttt{, }k\texttt{-1)} & \text{since } k > 0 \end{array}$$

Now we distinguish two subcases, depending on whether k is even or odd.

Case $k = 2k'$ for some $k' < k$. Then continuing the computation above (assuming the correctness of `even`) yields

$$\begin{array}{lll} & \texttt{power(}n\texttt{, }k\texttt{)} & \\ \Longrightarrow & \texttt{square(power(}n\texttt{, }2k' \texttt{ div 2))} & \\ \stackrel{1}{\Longrightarrow} & \texttt{square(power(}n\texttt{, }k'\texttt{))} & \\ \Longrightarrow & \texttt{square(}n^{k'}\texttt{)} & \text{by inductive hypothesis on } k' \\ \Longrightarrow & (n^{k'})^2 & \text{by Lemma 1} \\ = & n^{2k'} = n^k & \end{array}$$

Note that $k' < k$ and $k > 0$, so the inductive hypothesis applies.

Case $k = 2k' + 1$ for some $k' < k$. then continuing the computation above (assuming the correctness of `even`) yields

$$\begin{array}{lll} & \texttt{power(}n\texttt{, }k\texttt{)} & \\ \Longrightarrow & n \texttt{ * power(}n\texttt{, }k\texttt{-1)} & \\ \stackrel{1}{\Longrightarrow} & n \texttt{ * power(}n\texttt{, }k - 1\texttt{)} & \\ \stackrel{1}{\Longrightarrow} & n \texttt{ * } n^{k-1} & \text{by inductive hypothesis on } k - 1 \\ \stackrel{1}{\Longrightarrow} & n \times n^{k-1} = n^k & \end{array}$$

Here we can apply the inductive hypothesis since $k - 1 < k$ and $k > 0$.

□

Before starting a proof, it is generally useful to examine the pattern of recursion of the function one wishes to prove correct. If it calls itself on the immediate predecessor only, this signals a use of mathematical induction. If it calls itself on other, smaller terms, a use of strong induction is more likely.

4 Generalizing the Inductive Hypothesis

From the examples so far it may seem that induction is always completely straightforward. While many induction proofs that arise in program correctness are indeed simple, there is the occasional function whose correctness proof turns out to be difficult. This is often because we have to prove something more general than the final result we are aiming at. This is referred to as *generalizing the inductive hypothesis* and it can be shown that there exists no general recipe for generalization which will always work. However, one can isolate certain common cases.

As an example, consider the following implementation of the factorial function.

```
(* fact' : int * int -> int
   REQUIRES: n >= 0
   ENSURES: fact'(n, a) ==> (n!)*a
*)

fun fact'(0, a) = a
  | fact'(n, a) = fact'(n-1, n*a)


(* fact : int -> int
   REQUIRES: n >= 0
   ENSURES: fact(n) ==> n!
*)

fun fact(n) = fact'(n, 1)
```

We would like to prove that `fact(`n`)` $\hookrightarrow n!$ for every $n \geq 0$. This requires a lemma about `fact'`, which takes one more argument. So we have to determine which specification `fact'` satisfies. Simply checking if

`fact'(`n`, 1)` $\hookrightarrow n!$

will not work, since a proof by induction fails.

Here is the problematic case. Suppose we have assumed the inductive hypothesis

`fact'(`n`, 1)` $\hookrightarrow n!$

and then try to prove

`fact'(`$n+1$`, 1)` $\hookrightarrow (n+1)!$.

We start as before:

$$\begin{array}{ll} & \texttt{fact'}(n+1,\ 1) \\ \stackrel{1}{\Longrightarrow} & \texttt{fact'}(n+1\texttt{-}1,\ n+1\texttt{*}1) \\ \stackrel{1}{\Longrightarrow} & \texttt{fact'}(n,\ n+1\texttt{*}1) \\ \stackrel{1}{\Longrightarrow} & \texttt{fact'}(n,\ n+1) \end{array}$$

but now we cannot apply the inductive hypothesis since the second argument in the call to `fact'` is not `1` but n.

The solution is to generalize the induction property to allow any `a:int` in such a way that the desired result above follows easily. The following theorem generalizes appropriately.

Lemma 4 *`fact'(n, a)` $\hookrightarrow (n!) \times a$ for $n \geq 0$ and every integer `a`.*

Proof: By mathematical induction on n.

Base Case: $n = 0$.

$$\begin{array}{ll} & \texttt{fact'(0, } a\texttt{)} \\ \stackrel{1}{\Longrightarrow} & a = 1 \times a = 0! \times a \end{array}$$

Induction Step: Prove for $n + 1$, with $n \geq 0$.

Inductive hypothesis: Assume that for some $n \geq 0$, and every integer a, `fact'(`n`, `a`)` $\hookrightarrow (n!) \times a$.

We need to show that: `fact'(`$n+1$`, `a`)` $\hookrightarrow ((n+1)!) \times a$.

Evaluating code, we see that:

$$\begin{array}{lll} & \texttt{fact'(}n+1\texttt{, } a\texttt{)} & \\ \stackrel{1}{\Longrightarrow} & \texttt{fact'(}n+1\texttt{-1, } n+1\texttt{*}a\texttt{)} & \text{since } n \geq 0 \\ \stackrel{2}{\Longrightarrow} & \texttt{fact'(}n\texttt{, } (n+1) \times a\texttt{)} & \\ \Longrightarrow & n! \times ((n+1) \times a) & \text{by inductive hypothesis on } n \\ = & (n+1)! \times a & \end{array}$$

Note that the lemma (and therefore the inductive hypothesis) is stated *for every* a. That's why we can apply the inductive hypothesis when the second argument to `fact'` is $(n + 1) \times a$.

□

The main theorem now follows directly:

Theorem 5 *`fact(n)` $\hookrightarrow n!$ for $n \geq 0$.*

Proof: By direct computation and Lemma 4.

$$\begin{array}{lll} & \texttt{fact(}n\texttt{)} & \\ \stackrel{1}{\Longrightarrow} & \texttt{fact'(}n\texttt{, 1)} & \\ \Longrightarrow & (n!) \times 1 & \text{by Lemma 4} \\ = & n! & \end{array}$$

□