15–150: Principles of Functional Programming

Some Notes on Induction

Michael Erdmann*
Spring 2020

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 $e \stackrel{k}{\Longrightarrow} e'$ (or $e \Longrightarrow^k e'$) for a computation of k steps, $e \Longrightarrow e'$ (or $e \Longrightarrow^* e'$) 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 value 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 `SML`'s evaluation rule for `*`, with $\times$ meaning mathematical multiplication]
$=$	n^2	[math, assuming implemented correctly]

□

*Modified from a draft by Frank Pfenning.

2 Mathematical Induction

The simplest form of induction over natural numbers $0, 1, \ldots$ is *mathematical induction*, also called *standard induction* or *simple induction* or *weak induction*. Assume we have to prove a property for every natural number n. We first prove that the property holds for 0 (base case). Then we assume the property holds for n and establish it for $n + 1$ (induction step). Base case and induction 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, 0:int):int = 1
  | power(n:int, k:int):int = 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 2 *`power(`n`, k)` $\hookrightarrow n^k$ for all integers $k \geq 0$ and all integers `n`.*

Comment: To be very precise we should formulate the theorem as: *`power(`n`, k)` $\hookrightarrow n^k$ for all integer values $k \geq 0$ and all integer values `n`.* After all, if n is simply an expression of type `int`, then it itself might not reduce to a value, in which case we cannot assert that passing it to a function will produce a value. This subtlety can matter in certain contexts. However, generally it is understood that one is talking about arguments being values when one states a theorem or lemma as above.

Proof: By mathematical induction on k.

Base Case: $k = 0$.

We need to show that: `power(`n`, 0)` $\hookrightarrow n^0$, for all `n`. Observe that $n^0 = 1$.

	`power(`n`, 0)`	
$\stackrel{1}{\Longrightarrow}$	1	[the first clause of `power` is relevant since $k = 0$]

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

Induction 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:

$$
\begin{array}{lll}
 & \texttt{power}(n,\ k+1) & \\
\overset{1}{\Longrightarrow} & n\ \texttt{*}\ \texttt{power}(n,\ k+1\ \texttt{-}\ 1) & \text{[second clause of } \texttt{power} \text{ since } k+1>0] \\
\overset{1}{\Longrightarrow} & n\ \texttt{*}\ \texttt{power}(n,\ k) & \text{[math]} \\
\Longrightarrow & n\ \texttt{*}\ n^k & \text{[by induction hypothesis]} \\
\overset{1}{\Longrightarrow} & n \times n^k & \text{[evaluation rule for } \texttt{*}] \\
= & n^{k+1} & \text{[math]}
\end{array}
$$

By the induction principle for natural numbers, this completes the proof. □

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 induction 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* or *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 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 induction 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(k:int):bool = (k mod 2 = 0)
```

We assume without proof that for all integer values k, `even(`k`)` $\hookrightarrow$ `true` if k is even and `even(`k`)` $\hookrightarrow$ `false` if k is odd. This also says that the function `even` is total. On a homework assignment you need to prove such facts for any helper functions unless we give you those facts as lemmas.

```
(* 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)
```

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

Proof: By strong induction on k.

Base Case: $k = 0$.

We need to show that: `power(n, 0)` $\hookrightarrow n^0$, for all `n`. Observe that $n^0 = 1$.

	`power(n, 0)`	
$\stackrel{1}{\Longrightarrow}$	1	[the first clause of `power` is relevant since $k = 0$]

Induction Step: $k > 0$.

Induction Hypothesis: Assume that, for some $k > 0$, `power(n, k')` $\hookrightarrow n^{k'}$ for all k' such that $0 \leq k' < k$ and for all integers `n`.

We need to show that: `power(n, k)` $\hookrightarrow n^k$, for all integers `n`.

Evaluating code, we see that:

	`power(n, k)`	
$\stackrel{1}{\Longrightarrow}$	`if even(k)`	[second clause of `power` since $k > 0$]
	`then square(power(n, k div 2))`	
	`else n * power(n, k-1)`	

We now distinguish two subcases (as does the code), depending on whether k is even or odd:

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

	`power(n, k)`	
$\Longrightarrow$	`square(power(n, k div 2))`	[by assumptions about `even`]
$\Longrightarrow$	`square(power(n, k'))`	[since $k = 2k'$, and assuming `div` correct]
$\Longrightarrow$	`square(`$n^{k'}$`)`	[by induction hypothesis on k']
$\Longrightarrow$	$(n^{k'})^2$	[by Lemma 1]
$=$	$n^{2k'} = n^k$	[math]

Note that $0 < k' < k$, so the induction hypothesis applies.

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

	`power(n, k)`	
$\Longrightarrow$	`n * power(n, k - 1)`	[by assumptions about `even`]
$\Longrightarrow$	`n *` n^{k-1}	[by induction hypothesis on $k - 1$]
$\Longrightarrow$	n^k	[math]

(Here we could apply the induction hypothesis because $0 \leq k - 1 < k$.)

By strong induction over the natural numbers, this completes the proof. □

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 Induction 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 induction 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 induction 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 induction hypothesis since the second argument in the call to `fact'` is not `1` but $n+1$.

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: $n = 0$.

We need to show that `fact'(0, a)` $\hookrightarrow (0!) \times a$, for all integers a.

	`fact'(0, a)`	
$\stackrel{1}{\Longrightarrow}$	a	[first clause of `fact'`]
$=$	$1 \times a$	[math]
$=$	$0! \times a$	[math]

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

Induction 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)` $\hookrightarrow ((n+1)!) \times a$, for all integers a.

Evaluating code, we see that:

	`fact'(`$n+1$`, a)`	
$\stackrel{1}{\Longrightarrow}$	`fact'(`$n+1$`-1, (`$n+1$`)*a)`	[2nd clause of `fact'`, since $n+1 > 0$]
$\stackrel{2}{\Longrightarrow}$	`fact'(`n`, (`$n+1$`) × a)`	[math]
$\Longrightarrow$	$n! \times ((n+1) \times a)$	[by induction hypothesis on n, with $a' = (n+1) \times a$]
$=$	$(n+1)! \times a$	[math]

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

By the induction principle for natural numbers, this completes the proof. □

The main theorem now follows directly:

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

Proof: By direct computation and Lemma 4:

	`fact(`n`)`	
$\stackrel{1}{\Longrightarrow}$	`fact'(`n`, 1)`	[code for `fact`]
$\Longrightarrow$	$(n!) \times 1$	[by Lemma 4]
$=$	$n!$	[math]

□