CS3110 Fall 2013 Lecture 3: Induction and Recursion (9/5)

Robert Constable

1 Lecture Plan

- Recursion and Induction
  - termination
  - Gauss formula $\sum_{i=0}^{n} i = \frac{n*(n+1)}{2}$
  - evidence
- Tail recursion
- Sets and Types
  - disjoint union (variants)
  - match operator
  - see Hickey chapter 6, Kozen 2009 Lecture 4

2 Recursion and Induction

Recall that $sigma(n) =$ **if** $n = 0$ **then** 0
else $sigma(n-1) + n$

In the Lecture 2 Notes we saw that

Fact: $\forall n : \mathbb{N}. \exists m : \mathbb{N}.(sigma(n) = m)$

Proof by induction on n

Base: $n = 0$, $sigma(n) = 0$
Induction hypothesis: $\exists m : \mathbb{N}.(\mathbf{sigma(n-1)} = m)$
Let m_0 be this number, so $sigma(n-1) = m_0$
We must show $\exists m : \mathbb{N}.(sigma(n) = m)$
* By definition $sigma(n) = sigma(n-1) + n$
By the induction hypothesis we know the value of $sigma(n-1) = m_0$
Thus by substituting in line * we known $\mathbf{sigma(n) = m_0 + n}$
Thus the value m we need is $m_0 + n$

Qed

This proof seems trivial, easy, elementary and obvious. So what does it demonstrate to us?

1. It shows that $sigma(n)$ as a recursive function **terminates**. We can't say this in OCaml as a general fact about $\mathbb{N}$.

2. In OCaml we can only know
   $sigma(0) \downarrow 0$
   $sigma(1) \downarrow 1$
   $sigma(2) \downarrow 3$
   $sigma(3) \downarrow 6$
   etc.

3. The inductive proof of termination is essentially building evidence recursively. The very method of induction on $\mathbb{N}$ is a claim about a whole family of recursive programs that follow this pattern.

- Base: build something (object, evidence, ...) for $n = 0$
- Show how to take something built for value $n - 1$ and build it for n

  $object(n-1) \hookrightarrow object(n)$ using some function f

4. Induction is this recursive function

$$\begin{aligned} induction(n) = \ & \textbf{if } n = 0 \ \textbf{ then } evidence(0) \\ & \textbf{else } f(induction(n-1)) \end{aligned}$$

In PS1 you saw how to view this as the iteration of function f, namely

$$f^{(n)}(evidence(0))$$

$$\text{where} \quad \begin{cases} f^{(0)}(a) = a \\ f^{(n)}(a) = f(f^{(n-1)}(a)) \end{cases}$$

Now let's prove something much more interesting about $sigma(n)$ as $\sum_{i=0}^{n} i$.

Gauss noticed this

$$\begin{array}{ccccccccccccc} 1 & + & 2 & + & 3 & + & \cdots & + & 99 & + & 100 & & \\ 100 & + & 99 & + & 98 & + & \cdots & + & 2 & + & 1 & & \\ \hline 101 & + & 101 & + & 101 & + & \cdots & + & 101 & + & 101 & = & \frac{100*(101)}{2} \end{array}$$

In general $sigma(n) = \frac{n*(n+1)}{2}$.

We can prove this by induction and compute the sum at the same time.

Theorem: $\forall n : \mathbb{N}.(sigma(n) * 2 = n * (n+1)) = true$

Proof by induction

Base: $(sigma(0) * 2 = 0 * (n+1)) = true$ since $sigma(0) = 0$.
Induction: **assume sigma$(\mathbf{n-1}) * \mathbf{2} = (\mathbf{n-1}) * \mathbf{n}$** to show $sigma(n) * 2 = n * (n+1)$

By definition

$$\begin{aligned} sigma(n) * 2 &= (n + sigma(n-1)) * 2 \\ \text{By the induction hyp} \quad &= (n * 2 + sigma(n-1) * 2) \\ &= (n * 2 + (n-1) * n) \\ &= (n^2 + n) \\ &= n * (n+1) \\ &= true \end{aligned}$$

Qed

So the inductive proof is actually a recursive procedure we can execute.

The same proof we just gave can establish this more informative claim.

Theorem: $\forall n : \mathbb{N}. \exists m : \mathbb{N}. \exists b : Bool.$
$m = sigma(n)$ &
$b = (2 * m = n * (n+1))$ &
$b = true$

The "program" we get from the proof is

$$\begin{aligned} induction(n) = \textbf{if } n = 0 \textbf{ then } &((0, 2 * 0 = 0 * 1), true) \\ \textbf{else } &((sigma(n), 2 * sigma(n) = n * n + 1), true) \end{aligned}$$

Although recursive procedures in functions like *sigma* are easy to reason about and understand, there are other good methods of doing these computations such as loops. Here is a simple generic while loop. It is slightly more efficient than *sigma* as we see by performing the computations step by step.

$sum := 0$
while $n > 0$ **do**
  $sum := sum + n$
  $n := n - 1$
od
$\{n = 0, sum = n + n - 1 + \cdots + 1\}$

We can write this as a recursive function

$$
\begin{aligned}
\mathit{while}(s, n) = \ & \textbf{if } n = 0 \textbf{ then } s \\
& \textbf{else } \mathit{while}(s + n, n - 1)
\end{aligned}
$$

We can show that

$$\mathit{while}(0, n) = \mathit{sigma}(n)$$

We will see in due course how to write while loops generally as recursive functions. These are **tail recursive functions**, a topic we study later. (OCaml won't accept while as the name.)

3 Addendum

3.1 Sets and Types

In mathematics books, data is usually represented by sets, even for natural numbers $\mathbb{N}$.

$$\begin{array}{ll} \text{as a type:} & 0, 1, 2, 3, \ldots \\ \text{as a set:} & \emptyset \quad \text{empty set} \\ & \{\emptyset\} \\ & \{\emptyset, \{\emptyset\}\} \\ & \vdots \end{array}$$

In mathematics, the union of sets is a primitive,

$$\begin{array}{lcl} A \cup B & = & \{a\text{'s}, b\text{'s}\} \\ \mathbb{N} \cup Bool & = & \{true, false, 0, 1, 2, \ldots\} \end{array}$$

In OCaml we use **disjoint unions** also called **variant types** or **sum types**.

```
type int_bool = Int of int | Bool of bool
```

3.2 Disjoint Union aka Variant types

How do we use elements of `int_bool`?

There is a new kind of expression

$$\textbf{match } e \textbf{ with } p_1 \to e_1 \mid \cdots \mid p_n \to e_n$$

Here is a function to convert `int_bool` to `bool`

```
let convert(x:int_bool):bool =
  match x with
  | Int(x) -> true
  | Bool(x) -> x
```