Programming language theory

3 Structural induction

The induction axiom for the natural numbers can be used to establish induction principles that are more convenient to prove properties of values defined by an abstract grammar. A *property* will be a function from some type `T` defined by the grammar to the set of truth values $\mathbb{B}$. We always require that a property p is a *total* function, i.e. it is defined for all values in `T`. We say that $\mathtt{t} \in \mathtt{T}$ *has property* p if and only if `p(t)`.

3.1 Induction over `N`

Addition for the data type `N` is defined by

```
add Zero m = m
add (Suc n) m = Suc (add n m)
```

We are going to show that `add n m = add m n` for all `n` and `m` of type `N`.

We will use an induction principle for the Haskell type `N` defined in the previous section.

> **Sats.**Induction principle Let $p \in \mathtt{N} \to \mathbb{B}$ be a property. If
>
> a) $p(\mathtt{Zero})$ is true and
> b) $p(\mathtt{k})$ implies $p(\mathtt{Suc\ k})$ for every $\mathtt{k} \in \mathtt{N}$
>
> then $p(\mathtt{n})$ is true for all $\mathtt{n} \in \mathtt{N}$. ■

We will establish the principle in a subsequent section.

We start to prove that $n + 0 = n$ for all n, i.e. `add n Zero = n`. Since this result is a special case of the main theorem we state it as a lemma.

> **Lemma.** `add n Zero = n` for all `n` in `N`. ■

Proof. We use the induction principle with $p(\mathtt{n}) = (\mathtt{add\ n\ Zero{=}n})$.

a) First we prove that $p(\mathtt{Zero})$ is true, i.e `add Zero Zero = Zero`. This follows from the first clause for `add`.

b) Next we assume that $p(\mathtt{k})$ is true for an arbitrary but fixed $\mathtt{k} \in \mathtt{N}$, i.e. `add k Zero = k`. We shall prove that $p(\mathtt{Suc\ k})$, i.e. `add (Suc k) Zero = Suc k.` :
`add (Suc k) Zero = Suc(add k Zero) = Suc k`.

Thus booth the conditions hold and the proof is complete. □

We leave the next lemma as an exercise.

> **Lemma.** Let `m` be any element in `N`. Then `add n (Suc m) = add (Suc n) m` for all `n` in `N`. ■

The main theorem:

Sats. `add n m = add m n` for all `n, m` in `N`. ■

Proof. Let `m` $\in$ `N` be arbitrary but fixed throughout this proof. Let p(`n`) $=$ (`add n m = add m n`)

a) p(`Zero`) $=$ (`add Zero m = add m Zero`). This is true because of the first lemma.

b) Assume that `add k m = add m k` for an arbitrary but fixed `k`. We have to show that `add (Suc k) m = add m (Suc k)`.
`add (Suc k) m = Suc (add k m) = Suc (add m k) = add (Suc m) k`.
`add m (Suc k) = add (Suc m) k` by the previous lemma.

□

3.2 Expressions

Next we state an induction principle for the arithmetic expressions from the first lecture.

```
data Expr = Num Integer | Add Expr Expr | Mul Expr Expr
  deriving Eq
```

Sats. Induction principle Let $p \in$ `Expr` $\to \mathbb{B}$ be a property. If

a) p(`Num n`) is true for all `n`.
b) p(`e1`) and p(`e2`) implies p(`Add e1 e2`) for every `e1, e2` $\in$ `Expr`
c) p(`e1`) and p(`e2`) implies p(`Mul e1 e2`) for every `e1, e2` $\in$ `Expr`

then p(`e`) is true for all `e` $\in$ `Expr`. ■

We define two functions:

```
value :: Expr -> Integer
value (Num n) = n
value (Add expr1 expr2) = value expr1 + value expr2
value (Mul expr1 expr2) = value expr1 * value expr2

mirror :: Expr -> Expr
mirror (Num i) = Num i
mirror (Add e1 e2) = Add (mirror e2) (mirror e1)
mirror (Mul e1 e2) = Mul (mirror e2) (mirror e1)
```

Then the following is true.

Sats. `value e = value (mirror e)` is true for all `e` $\in$ `Expr`. ■

Proof. a) If `e = Num i` then

```
value (mirror e) =
value (mirror (Num i)) =
value (Num i) =
value e
```

b) Let `e = (Add e1 e2)` and assume that `value e1 = value (mirror e1)` and `value e2 = value (mirror e2)` then

```
value (mirror e) = value (mirror (Add e1 e2)) =
value (Add (mirror e2) (mirror e1)) =
value (mirror e2) + value (mirror e1)=
value e2 + value e1 =
value e1 + value e2 =
value (Add e1 e2) =
value e
```

c) When `e = Mul e1 e2` we reason in the same way.

$\square$

3.3 Induction principles

We shall prove the induction principle for `Expr`.

Sats. Let $p \in$ `Expr` $\to \mathbb{B}$ be a property. If

a) p(`Num n`) is true for all `n`.
b) p(`e1`) and p(`e2`) implies p(`Add e1 e2`) for every `e1, e2` $\in$ `Expr`
c) p(`e1`) and p(`e2`) implies p(`Mul e1 e2`) for every `e1, e2` $\in$ `Expr`

then p(`e`) is true for all `e` $\in$ `Expr`. $\blacksquare$

Proof. Define the depth `d` of an `Expr`:

```
d :: Expr -> Integer
d (Num _) = 0
d (Add e1 e2) = 1 + max (d e1) (d e2)
d (Mul e1 e2) = 1 + max (d e1) (d e2)
```

Let $q(n)$ be the property that all `Expr e` with `d e` $\leq n$ have the property p(`e`). Let $A = \{n \mid q(n)\}$ be the set of all n such that $q(n)$ is true.

1. Because of a) we have that $0 \in A$.
2. Now assume that $k \in A$ where k is a fixed arbitrary number. This means that all `e` with `d e` $\leq k$ has property p. We have to show that $q(k+1)$ is true, i.e. $p(e)$ for all `e` with `d e` $\leq k+1$. It remains to prove that all `e` with `d e` $= k+1$ has property p. Since e must be either of the form `Add e1 e2` or `Mul e1 e2` where `d e1` $\leq k$ `d e2` $\leq k$ it follows from b) and c) that e has property p.

Using the induction axiom for natural numbers we conclude that $A = \mathbb{N}$ and the proof is complete. $\square$

It is easy to extend this theorem to any recursive data type that does not depend on a mutually recursive data type.