InductFun!
Types, Program Synthesis, and Inductive Proof

Peter-Michael Osera
Steve Zdancewic

Penn University of Pennsylvania

EXCAPE
How can programming systems technology not just augment existing pedagogy but also enable new pedagogy?
Proof. We use mathematical induction.

Base case: Setting \( n = 0 \), we get \( 2^0 = 1 = 2^1 - 1 \) as required.

Induction step: Let \( n \) be an arbitrary natural number and suppose that \( 2^0 + 2^1 + \cdots + 2^n = 2^{n+1} - 1 \). Then

\[
2^0 + 2^1 + \cdots + 2^{n+1} = (2^0 + 2^1 + \cdots + 2^n) + 2^{n+1} \\
= (2^{n+1} - 1) + 2^{n+1} \\
= 2 \cdot 2^{n+1} - 1 \\
= 2^{n+2} - 1.
\]

\[\square\]

- Unscalable
- Vaguely evaluated
- Uninspiring
Demo!
(inductfun.org)
InductFun! is (will be):

- A scalable, programming-centric pedagogy for learning proof.
- An intelligent tutor for inductive proof.
- Types and program synthesis in action.
All expressions
nat → bool
\[ \text{eq_refl}_{x, A, x} \]

\[ X =_A X \]
Q: Where can we use synthesis technology in InductFun!?  

A: Assisting students with generalizing induction hypotheses

Fixpoint double (n m:nat) : nat :=
  match n with
  | O    => O
  | S n' => S (S (double n' m))
  end.

Theorem double_injective :
  forall n m, double m = double n -> m = n
Theorem double_injective_bad : 
   \forall n m, double m = double n -> m = n.
Proof.
   intros n m.
   induction n.
   (* ... *)
   assert (n' = m') as H.

\begin{itemize}
\item n' : nat
\item m' : nat
\item IHn' : double n' = double (S m') -> n' = S m'
\item eq : double (S n') = double (S m')
\end{itemize}
\begin{equation*}
\text{n' = m'}
\end{equation*}
Theorem double_injective_good : 
  \forall n m, double m = double n -> m = n.
Proof.
  intros n.
  induction n.
  (* ... *)
  assert (n' = m') as H.

n' : nat
m' : nat
IHn' : \forall m : nat, double n' = double m' 
  -> n' = m'
eq : double (S n') = double (S m')
============================================= 
n' = m'
Definition double_injective_bad :=
...
| S m’ =>
  fun (IHn’0 : ...)
    (eq1 : ...) =>
    (fun H : n’ = m’ => ?70) ?69
...

Definition double_injective_good :=
...
| S m’ =>
  (eq1 : ...) =>
  (fun H : n’ = m’ => ?135) ?134
...
Definition double_injective_bad :=
  (fun n m : nat =>
   nat_ind (fun n0 : nat => double n0 =
               double m -> n0 = m)

Definition double_injective_good :=
  (fun n : nat =>
   nat_ind (fun n0 : nat => forall m : nat,
               double_n0 = double m -> n0 = m) ...)
Try it out!
http://inductfun.org

Questions? Guinea Pig?
posera@cis.upenn.edu

Thanks!