Teaching Induction with Functional Programming and a Proof Assistant

Peter-Michael Osera
Steve Zdancewic
### PL Programming Languages (8 Core-Tier1 hours, 20 Core-Tier2 hours)

<table>
<thead>
<tr>
<th>PL/ Programming Language</th>
<th>Core-Tier1 hours</th>
<th>Core-Tier2 hours</th>
<th>Includes Electives</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/Object-Oriented Programming</td>
<td>4</td>
<td>6</td>
<td>N</td>
</tr>
<tr>
<td>PL/Functional Programming</td>
<td>3</td>
<td>4</td>
<td>N</td>
</tr>
<tr>
<td>PL/Event-Driven and Reactive Programming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL/Basic Type Systems</td>
<td>1</td>
<td>4</td>
<td>N</td>
</tr>
<tr>
<td>PL/Program Representation</td>
<td>1</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>PL/Language Translation and Execution</td>
<td>3</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>PL/Syntax Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL/Compiler Semantic Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL/Code Generation</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Runtime Systems</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Static Analysis</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Advanced Programming Constructs</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Concurrency and Parallelism</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Type Systems</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Formal Semantics</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Language Pragmatics</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>PL/Logic Programming</td>
<td></td>
<td></td>
<td>Y</td>
</tr>
</tbody>
</table>

**Note:**
- Some topics from one or more of the first three Knowledge Units (Object-Oriented Programming, Functional Programming, Event-Driven and Reactive Programming) are likely to be integrated with topics in the Software Development Fundamentals Knowledge Area in a curriculum’s introductory courses. Curricula will differ on which topics are integrated in this fashion and which are delayed until later courses on software development and programming languages.
- Some of the most important core learning outcomes are relevant to object-oriented programming, functional programming, and, in fact, all programming. These learning outcomes are repeated in the Object-Oriented Programming and Functional Programming Knowledge Units, with a note to this effect. We do not intend that a
Is functional programming...

An ends towards better software?

A means to better teaching CS?
“What’s induction good for?”

1. What’s induction?
2. Absolutely nothing.
3. It makes me a “better programmer”.
Inductive Datatypes + Recursive Functions

Inductive Reasoning
InductFun!
 Proof assistant + fun. prog.

Let's compare the proof script for `plus n 0 = n` to a paper proof of the same proposition:

<table>
<thead>
<tr>
<th>Decs</th>
<th>Eval</th>
<th>Proofs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Lemma plusO : forall (n:nat), plus n 0 = n</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Proof</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>introduce</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>induction n</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>simplify</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>reflexivity</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>rewrite IHn</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>reflexivity</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Qed</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After line 5...
2 subgoals

---------------------------
O = 0

subgoal 2 is:
`plus (S n) 0 = S n`
Example!
What is it?
What’s next?
Proof Assistants and Education

The Coq Proof Assistant

ProofWeb

ACL2
Our goals:

1. Teach the mechanics of inductive proofs.

Small functional programs

Small inductive proofs
Our goals:

2. Bridge formal and informal reasoning.

“\( n + 0 = n \) is true by induction on \( n \) (and I believe it!)”
Programming

- High-level concepts
- Brevity

Paper

- Verification
- Attn. to details

Compiler
Proof → Pf. Assistant → Paper → Head

- High-level concepts
- Brevity
- Verification
- Attn. to details

Informal → Formal
Our Non-goals

1. Full-on course replacement.
2. Complete treatment of fn. prog.
4. Complete programming/proof system
5. Type theory/Curry-Howard

Minimize complexity ➔ Maximize Adoptability
InductFun! outline

1. Functional Programming
2. Basic proofs
3. Induction
4. Hypothesis strengthening
5. Formal/informal reasoning
6. Extended examples
What’s next?

• Better proof language
• User feedback ➔ UI tweaks
• “Domain-specific” examples
• UPenn course integration
• Automatic feedback and guidance
reflexivity simplify introduce rewritef h rewriteb h

true by reflexivity simplify to e assume e : T rewritef h to e rewriteb h to e

• More verbose ➔ better reflect paper pfs.
• Explicit assumptions and steps.
\textbf{destruct e}
\textbf{induction e}

by case analysis on e:
\quad | \text{case p } \rightarrow \text{ pf}

by induction on e:
\quad | \text{case p with } \text{H:P } \rightarrow \text{ pf}

Explicit:
\begin{itemize}
  \item Exhaustive enumeration of cases.
  \item Binding of names.
  \item Checks on inductive hypotheses.
\end{itemize}
Try it out!
http://inductfun.org

Questions? Comments?
posera@cis.upenn.edu

*(Poster session, too!)

Thanks!