Interactive Computer Theorem Proving

Lecture 4: Inductively-Defined Predicates
Administrivia

• The course registration database has been updated so that you can change your units for this class to 1.
  – Stay at 3 units if you plan to do a class project for credit.
  – Otherwise, reduce to 1 unit.
• Resource links added to web site.
• Created Majordomo mailing list for the class (details to follow by e-mail)
• You can check HW1 yourself, so not being “handed back.” :-)

2
What Is “less than or equal to”?

Fixpoint le_f (n m : nat) {struct n} : Prop :=
    match n with
    | O => True
    | S n' =>
        match m with
        | O => False
        | S m' => le_f n' m'
        end
    end
end.

le_f 1 3 \rightarrow le_f 0 2 \rightarrow True
le_f 3 1 \rightarrow le_f 2 0 \rightarrow False
Proving Transitivity

- **Theorem:** \( \text{le}_f n_1 n_2 \to \text{le}_f n_2 n_3 \to \text{le}_f n_1 n_3 \)

- **Proof:** By induction on \( n_1 \).

- **Case:** \( n_1 = 0 \).
  - \( \text{le}_f n_1 n_3 \) follows by computation.

- **Case:** \( n_1 = S n_1' \).
  - By computation, \( n_2 = S n_2' \) with \( \text{le}_f n_1' n_2' \).
  - From \( \text{le}_f (S n_2') n_3 \), we have \( n_3 = S n_3' \) with \( \text{le}_f n_2' n_3' \).
  - By the IH, \( \text{le}_f n_1' n_3' \).
  - By computation, \( \text{le}_f (S n_1') (S n_3') \).
Pros and Cons

• Pros
  – We used the same recursive function mechanism that's worked before.

• Cons
  – That was a fairly acrobatic proof for such a simple theorem!
  – How could we use this same approach to define what it means for a Turing machine to halt with a particular configuration?
Another “less than or equal to”

\[ n \leq n \quad \text{le}_n \]

\[ n \leq S \cdot m \quad \text{le}_S \]

\[ 4 \leq 4 \quad \text{le}_n \]

\[ 1 \leq 1 \quad \text{le}_n \]

\[ 1 \leq 2 \quad \text{le}_S \]

\[ 1 \leq 3 \quad \text{le}_S \]

\[ 3 \leq 1 \quad ? \]
Transitivity Again

If $\mathcal{P} : n1 \leq n2$ and $Q : n2 \leq n3$, then there exists $\mathcal{R} : n1 \leq n3$.

**Proof:** Induction *on the structure of $Q$.*

**Case:** $\mathcal{P} : n1 \leq n2$; $Q = \frac{n2 \leq n2}{\text{le}_n}$ ($n3 = n2$)

**Answer:** $\mathcal{R} = \mathcal{P}$

**Case:** $\mathcal{P} : n1 \leq n2$; $Q = \frac{Q' : n2 \leq n3'}{\text{le}_S}$ ($n3 = S \ n3'$)

**IH:** For any $\mathcal{P}' : n1 \leq n2$, there exists $\mathcal{R}' : n1 \leq n3'$.

**Answer:** $\mathcal{R} = \frac{\mathcal{R}' \ (\text{with } \mathcal{P}' := \mathcal{P})}{n1 \leq S \ n3'} \ \text{le}_S$
## A Comparison

<table>
<thead>
<tr>
<th>Fixpoint</th>
<th>Inductive</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Computation proves many theorems <em>atomically</em>.</td>
<td>• Explicit “logic programming” needed in all cases.</td>
</tr>
<tr>
<td>• Proofs follow the structure of the data type.</td>
<td>• Proofs follow the structure of <em>your custom proof rules</em>.</td>
</tr>
<tr>
<td>• The natural formulation only works for a limited subset of the <em>computable</em> functions.</td>
<td>• Natural formulations work for expressing very <em>uncomputable</em> notions.</td>
</tr>
</tbody>
</table>

**Fixpoint**

**Inductive**
Example: Turing Machines

Let $\mathcal{F}$ be the set of final states.

Let $\Rightarrow^1$ be a binary relation expressing a single step.

We want to define $\Rightarrow$, the state reachability relation.

\[
\begin{array}{c}
C \Rightarrow C \\
C \Rightarrow^1 C' \\
C' \Rightarrow C''
\end{array}
\]

How do we express “a machine starting in configuration $C$ terminates”?

\[
\exists C', C' \in \mathcal{F} \land C \Rightarrow C'
\]

How is it that we got around the undecidability of this problem?

$$\begin{array}{c}
\text{Fixpoint } \text{run} \ (n : \text{nat}) \ (C : \text{config}) \ {\text{struct } n} : \\
\ \text{Prop} := \\
\ \text{match } n \ \text{with} \\
\ | \ 0 => F \ C \\
\ | \ S \ n' => F \ C \lor \text{run} \ n' \ (s1 \ C) \\
\ \text{end.} \\
\exists \ n, \ \text{run} \ n \ C
\end{array}$$

$$\text{run} \ 2 \ C \Rightarrow F \ C \lor F \ (s1 \ C) \lor F \ (s1 \ (s1 \ C))$$
Example: Sorted Lists

We want to define what it means for a list \( \text{nat} \) to be sorted.

\[
\begin{align*}
\text{sorted } [] &= \text{true} \\
\text{sorted } [n] &= \mathbf{if} \quad n \leq m \\
&\quad \text{sorted } (m :: ls) \\
&\quad \text{then } \text{sorted } (n :: m :: ls) \quad \mathbf{else} \quad \text{false}
\end{align*}
\]
Sublists

We want to define what it means for one list to be a sublist of another.

Example: [1], [1, 4], and [2, 3] are some sublists of [1, 2, 3, 4]. [3, 2] is not a sublist of [1, 2, 3, 4], because order matters.

\[
\text{sublist } \emptyset \emptyset \\
\text{sublist } \text{} \text{} \\
\text{sublist } \text{ls} \text{ls}' \\
\text{sublist } (n :: \text{ls}) (n :: \text{ls}') \\
\text{sublist } \text{ls} \text{ls}' \\
\text{sublist } (n :: \text{ls}) \text{ls}'
\]
A Theorem

If sorted \( ls \)
and sublist \( ls \) \( ls' \)
then sorted \( ls' \).
Example: A Programming Language Interpreter

\[\begin{align*}
n &::= O \mid S \ n \\
b &::= \text{true} \mid \text{false} \\
x &::= [\text{variable}] \\
e &::= n \mid b \mid x \mid e + e \mid e = e \\
c &::= \text{skip} \mid x ::= e \mid c; c \mid \text{if } e \text{ then } c \text{ else } c \\
        \mid \text{while } e \text{ do } c
\end{align*}\]
Conclusion

- Sample HW2 solution will be on the website.
- HW3 is posted
  - Inductively-defined predicates involving lists
- Next lecture: Proofs as programs
  - Given by George, since I'll be out of town at the Workshop on Mechanizing Metatheory\(^1\)
    - [http://www.cis.upenn.edu/~sweirich/wmm/](http://www.cis.upenn.edu/~sweirich/wmm/)

\(^1\) That means using computers to write proofs about programming languages.