Interactive Computer Theorem Proving

Lecture 4: Inductively-Defined Predicates

CS294-9
September 14, 2006
Adam Chlipala
UC Berkeley

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." :-)

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.
```

le_f 1 3 $\longrightarrow$ le_f 0 2 $\longrightarrow$ True

le_f 3 1 $\longrightarrow$ le_f 2 0 $\longrightarrow$ False

Proving Transitivity

- **Theorem**: $\mathrm{le\_f}\; n1\; n2$ -> $\mathrm{le\_f}\; n2\; n3$ -> $\mathrm{le\_f}\; n1\; n3$
- **Proof**: By induction on $n1$.
- **Case**: $n1 = 0$.
  - $\mathrm{le\_f}\; n1\; n3$ follows by computation.
- **Case**: $n1 = \mathrm{S}\; n1'$.
  - By computation, $n2 = \mathrm{S}\; n2'$ with $\mathrm{le\_f}\; n1'\; n2'$.
  - From $\mathrm{le\_f}\; (\mathrm{S}\; n2')\; n3$, we have $n3 = \mathrm{S}\; n3'$ with $\mathrm{le\_f}\; n2'\; n3'$.
  - By the IH, $\mathrm{le\_f}\; n1'\; n3'$.
  - By computation, $\mathrm{le\_f}\; (\mathrm{S}\; n1')\; (\mathrm{S}\; n3')$.

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”

$$\frac{}{n \leq n}\,\text{le\_n} \qquad \frac{n \leq m}{n \leq \mathbf{S}\; m}\,\text{le\_S}$$

$$\frac{}{4 \leq 4}\,\text{le\_n} \qquad \cfrac{\cfrac{\cfrac{}{1 \leq 1}\,\text{le\_n}}{1 \leq 2}\,\text{le\_S}}{1 \leq 3}\,\text{le\_S} \qquad \frac{?}{3 \leq 1}\,?$$

Transitivity Again

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

Proof: Induction **on the structure of** $\mathcal{Q}$.

Case: $\mathcal{P} : n1 \leq n2;\ \mathcal{Q} = \dfrac{}{n2 \leq n2}$ le_n $\quad (n3 = n2)$

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

Case: $\mathcal{P} : n1 \leq n2;\ \mathcal{Q} = \dfrac{\mathcal{Q}' : n2 \leq n3'}{n2 \leq \mathrm{S}\ n3'}$ le_S $\quad (n3 = \mathrm{S}\ n3')$

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

Answer: $\mathcal{R} = \dfrac{\mathcal{R}' \text{ (with } \mathcal{P}' := \mathcal{P})}{n1 \leq \mathrm{S}\ n3'}$ le_S

A Comparison

Fixpoint

- Computation proves many theorems *atomically*.
- Proofs follow the structure of the data type.
- The natural formulation only works for a limited subset of the *computable* functions.

Inductive

- Explicit "logic programming" needed in all cases.
- Proofs follow the structure of *your custom proof rules*.
- Natural formulations work for expressing very *uncomputable* notions.

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.

$$\frac{}{C \Rightarrow C} \qquad \frac{C \Rightarrow^1 C' \quad C' \Rightarrow C''}{C \Rightarrow C''}$$

How do we express "a machine starting in configuration C terminates"?

$$\exists\, C', C' \in \mathcal{F} \wedge C \Rightarrow C'$$

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

Fixpoint run (n : nat) (C : config) {**struct** n} : **Prop** :=
match n **with**
| O => F C
| S n' => F C $\vee$ run n' (s1 C)
end.

$$\exists\, n, \text{run } n\ C$$

$$\text{run } 2\ C \longrightarrow \text{F } C \vee \text{F } (\text{s1 } C) \vee \text{F } (\text{s1 } (\text{s1 } C))$$

Example: Sorted Lists

We want to define what it means for a $\mathrm{list\ nat}$ to be sorted.

$$\frac{}{\mathrm{sorted}\ []}$$

$$\frac{}{\mathrm{sorted}\ [n]}$$

$$\frac{n \le m \qquad \mathrm{sorted}\ (m :: ls)}{\mathrm{sorted}\ (n :: m :: ls)}$$

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.

$$\frac{}{\text{sublist } [] \; []}$$

$$\frac{\text{sublist } ls \; ls'}{\text{sublist } (n :: ls) \; (n :: ls')}$$

$$\frac{\text{sublist } ls \; ls'}{\text{sublist } (n :: ls) \; ls'}$$

A Theorem

$$\frac{}{\text{sorted } []}$$

$$\frac{}{\text{sorted } [n]}$$

$$\frac{n \le m \qquad \text{sorted } (m :: ls)}{\text{sorted } (n :: m :: ls)}$$

$$\frac{}{\text{sublist } [] \, []}$$

$$\frac{\text{sublist } ls \; ls'}{\text{sublist } (n :: ls) \; (n :: ls')}$$

$$\frac{\text{sublist } ls \; ls'}{\text{sublist } (n :: ls) \; ls'}$$

If sorted ls

and sublist ls ls'

then sorted ls'.

Example: A Programming Language Interpreter

$\underline{n}$ ::= O | S $\underline{n}$

$\underline{b}$ ::= true | false

x ::= [variable]

$\underline{e}$ ::= $\underline{n}$ | $\underline{b}$ | x | $\underline{e}$ + $\underline{e}$ | $\underline{e}$ = $\underline{e}$

$\underline{c}$ ::= **skip** | x := $\underline{e}$ | $\underline{c}$; $\underline{c}$ | **if** $\underline{e}$ **then** $\underline{c}$ **else** $\underline{c}$
| **while** $\underline{e}$ **do** $\underline{c}$

Conclusion

- Sample HW2 solution will be on the web site.
- 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/`

[1] That means using computers to write proofs about programming languages.