Class 20

Using Saoithín (I)

- A demo of more of the features and infrastructure of Saoithín.

Proof Strategy: Law Reduction

- If we start with an existing law (\textit{LAW}), and can transform it into our goal (\textit{GOAL}), then we have proven our goal.
- Law \textit{LAW} is equivalent to \textit{LAW} \equiv \text{true}, so

  \[
  \begin{align*}
  \text{GOAL} & \equiv \text{"our proof" } \\
  \text{LAW} & \equiv \text{"LAW is a theorem" } \\
  \text{true} & 
  \end{align*}
  \]

- If a law has free variables then we can substitute them by anything to get a statement that is still true.

Induction as Law-Reduction

- Consider Natural Number Induction

  \[
  \begin{align*}
  \langle \text{\textsc{N-induction}} \rangle & \quad P[0/\mathit{x}] \land (\forall n \bullet P[n/\mathit{x}] \Rightarrow P[\text{succ } n/\mathit{x}]) \\
  & \Rightarrow (\forall \mathit{x} \mid \mathit{x} \bullet P)
  \end{align*}
  \]

- If our goal is \(k + 0 = 0\) (say), then we can substitute it in for \(P\) in the induction axiom, also replacing \(x\) by \(k\), to get what is still a law (still true for values of its variables)

  \[
  \begin{align*}
  ( (k + 0 = 0)[0/k] \\
  & \land (\forall n \bullet (k + 0 = k))[n/k] \Rightarrow (k + 0 = k)[\text{succ } n/k]) \\
  & \Rightarrow (\forall k \mid k : \mathbb{N} \bullet (k + 0 = k))
  \end{align*}
  \]

- If we can reduce this to \text{true}, we are done.
Natural Induction using Saoithín

We now show the proof of this conjecture in Saoithín. (Live proof — screenshots and commentary will appear in “Using-Saoithin.pdf”)

Saoithín ‘How-to’ (1)

Match Against the Laws: Right-click in Proof-Goal window
Apply a substitution: Focus in on expressions and substitution, and press ‘a’ or ‘s’ key.
Theorem Summary: Right-click on Theory Name in top window and select Proof Summary.

CS3001 Project 2011

Goal: build a portfolio of theories in Saoithín
Method:
1. Rolling deadlines, one per week
2. Each Week develops a new Theory
3. Subsequent Theories build on earlier ones
   - A correct version of each such theory will be made available after each deadline.

Theory Portfolio

- The initial project release with have the following Theories: Logic, Equality
- Due start Week 10: Naturals
- Due start Week 11: Semantics
- Due start Week 12: Programs
- Submissions:
  - Electronic: email, subject CS3001-TheoryName All .teoric files.
  - Hardcopy: “Theorem Summary”, to CS Office, When: 4pm on Monday
Project Grading

- Marks will be awarded based on the number of conjectures proven (some worth more than others).
- Marks will also be awarded for pointing out genuine errors in theories, along with suggested corrections.
- Also marks will be awarded for identifying bugs (features, even!) in the prover.
- Fixes to theories/software will be issued ASAP.

A Theory of Lists (Syntax)

We use $\sigma$, $\tau$ for lists (a.k.a. sequences)

\[
\begin{align*}
\sigma, \tau & \in T^* \\
\{\} & \text{empty list} \\
X : \sigma & \text{list construction} \\
(x_1, \ldots, x_n) & \text{list enumeration} \\
\text{head}(\sigma) & \text{list destruction (1)} \\
\text{tail}(\sigma) & \text{list destruction (2)} \\
\sigma \triangleright \tau & \text{list concatenation} \\
\# \sigma & \text{list length} \\
\sigma \preceq \tau & \text{list prefix relation}
\end{align*}
\]

Some axioms for lists

Axioms cover the basic building blocks

- $\{\}\text{-is-}T^*$
- $\{(\cdot)\text{-is-}T^*$
- $X : T \land \sigma : T^* \Rightarrow (x \triangleright \sigma) : T^*$
- $\{\neq\}\text{-not-}\sigma$
- $x = y \land (\sigma = \tau) \equiv (x \triangleright \sigma) = (y \triangleright \tau)$
- $T^*$-Induction
- $P[\{\}/\sigma] \land (\forall v, \tau \Rightarrow P[\sigma]) \Rightarrow (\forall \sigma \Rightarrow T^* \bullet P)$

Some definitions for lists (!) (??)

- $\text{head-def}$
- $\text{tail-def}$
- $\text{enumeration-def}$
- $\text{concat-def}$
- $\text{length-def}$
- $\text{prefix-def}$
- $\text{front-def}$
- $\text{last-def}$

- Note that $\text{head}$, $\text{tail}$, $\text{front}$ and $\text{last}$ are partial, defined only for non-empty lists.
Some theorems for lists

Proof of \(\langle \langle \text{r-unit} \rangle \rangle \) (I)

- Goal: \( \sigma \langle \rangle = \sigma \)
- Strategy: induction over \( \sigma \)
  - Property \( P(\sigma) \equiv \sigma \langle \rangle = \sigma \)
  - Base Case: Show \( P(\langle \rangle) \)
  - Inductive Step: Show \( P(\tau) \Rightarrow P(x \circ \tau) \)

Proof of \(\langle \langle \text{r-unit} \rangle \rangle \) (II)

Base Case \( P(\langle \rangle) \equiv (\langle \rangle \langle \rangle = \langle \rangle) \)
Strategy: reduce LHS to RHS

\[
\begin{align*}
\langle \rangle \langle \rangle &= \langle \rangle \\
\text{"\langle \langle \text{-def} \rangle \rangle"} &
\end{align*}
\]

Proof of \(\langle \langle \text{r-unit} \rangle \rangle \) (III)

Inductive Step \( P(\tau) \Rightarrow P(x \circ \tau) \)
Goal: \( (\tau \langle \rangle = \tau) \Rightarrow ((x \circ \tau) \langle \rangle = x \circ \tau) \)
Strategy: assume LHS to show RHS

\[
\begin{align*}
\text{ind-hyp} &
\end{align*}
\]

\[
\begin{align*}
(x \circ \tau) \langle \rangle &= x \circ \tau \\
\text{"\langle \langle \text{-def} \rangle \rangle"} &
\end{align*}
\]

\[
\begin{align*}
x \circ \tau &= x \circ \tau \\
\text{"\langle \langle \text{-refl} \rangle \rangle"} &
\end{align*}
\]

true
Some definitions for lists (!) (??)

front (x) = ()
front (x : σ) = x :: front (σ)
last (x) = x
last (x : σ) = last (σ)

front and last are problematical, but not because they are partial.

Definitions for lists (Corrected, Safe)

head (x : σ) = x
tail (x : σ) = σ
<enumeration-def> (x₁, x₂, ..., xₙ) = x₁ :: (x₂ :: (... (xₙ :: ()))
<emptiness-def> () ⊢ τ = τ
<elimination-def> (x : σ) ⊢ τ = x :: (σ ⊢ τ)
<arity-def> #() = 0
<arity-def> #(x : σ) = 1 + #σ
front (x) = x
front (x : y : σ) = x :: front (y : σ)
last (x) = x
last (x : y : σ) = last (y : σ)

Note that head, tail, front and last are partial, defined only for non-empty lists.

Lists Theory: summary

- Induction using {T*}-Induction is the main proof technique for the laws of lists.
- Most if not all of the laws shown are done this way
- Care needs to be taken when defining functions ‘by cases’
  - Need to ensure that overlapping cases do not conflict.