# Experiments with a simplified version of (fast-power x n),
# which computes (Power x n) with $\log n$ multiplications,
# as an example where strong-induction proofs are useful.
# Based on the fast-power-embedded.ath, but nongeneric and
# experimenting with variations on strong-induction.

load "nat-power"
load "nat-half"
load "strong-induction"

extend-module N {
declare fast-power: [N N] -> N [{int->nat int->nat}] 

module fast-power {
assert axioms :=
(fun
[(fast-power x n) =
[one when (n = zero)
(square (fast-power x half n)) when (n /= zero & even n)
((square (fast-power x half n)) * x) when (n /= zero & ~ even n)]])

define [if-zero nonzero-even nonzero-odd] := axioms

(print "\n2 raised to the 3rd with fast-power: " (eval (fast-power 2 3)) "\n")

correctness := (forall n x . (fast-power x n) = x ** n)

define ^ := fast-power

define step :=
method (n)
assume ind-hyp :=
(forall m . m < n ==> forall x . x ^ m = x ** m)
conclude (forall x . x ^ n = x ** n)

pick-any x
(two-cases

assume (n = zero)
(!chain [(x ^ n)
--> one [if-zero]
<-- (x ** n) [Power.if-zero]
<-- (x ** n) [(n = zero)]])

assume (n /= zero)

let {fact1 :=
conclude goal := (forall x . x ^ half n = x ** half n)
(!chain-> [(n /= zero)
==> (half n < n) [half.less]
==> goal [ind-hyp]]));

fact2 :=
(converse (x ^ half n) = x ** (two * half n))
(!chain
[(square (x ^ half n))
--> (square (x ** half n)) [fact1]
--> (x ** (half n)) *
x ** half n) [square.def]
<-- (x ** ((half n) + half n)) [Power.Plus-case]
<-- (x ** (two * half n)) [Times.two-times]])

(two-cases

assume (even n)
(!chain
[(x ^ n)
--> (square (x ^ half n)) [nonzero-even]
--> (x ** (two * half n)) [fact2]
--> (x ** n) [EO.even-definition]])

assume (~ (even n))

let _ := (!chain-> [~ even n]
==> (odd n) [EO.odd-if-not-even]])
(!chain

[(x ^ n)

--> ((square (x ^ half n)) * x) [nonzero-odd]

--> ((x ** (two * half n)) * x) [fact2]

<-- ((x ** (two * half n)) * (x ** one)) [Power.right-one]

<-- (x ** ((two * half n) + one)) [Power.Plus-case]

--> (x ** n) [EO.odd-definition]]]]))

={!strong-induction.principle correctness step})