Speaking the Syntax

Ex 1: <=

Inductive le : nat → nat → Prop :=

| le_n (n : nat) : le n n

constructor #1

le_n says: Provided n, we can say $n \leq n$

| le_S (n m : nat) (H: le n m) : le n (S m)

constructor #2

le_S says:

Provided $n \leq m$, we can say $n \leq S\ m$

Ex 2: Collatz

Inductive Collatz_holds_for : nat → Prop :=

| Chf_done : Collatz_holds_for 2

| Chf_more (m: nat) : Collatz_holds_for (f m) →

Collatz_holds_for m

or, equivalently,

Inductive Collatz_holds_for : nat → Prop :=

| Chf_done : Collatz_holds_for 2

| Chf_more (m: nat) (H: Collatz_holds_for (f m)) :

Collatz_holds_for m

The ability to move H in-n-out suggests that Chf_move is essentially a function.

Let's speak it:

Inductive Collatz_holds_for : nat → Prop :=

| Chf_done : Collatz_holds_for 2

Chf_done says: Collatz holds for 1.

| Chf_more (m: nat) (H: Collatz_holds_for (f m)) :

Collatz_holds_for m

Chf_more says: Given m and that Collatz holds for (f m), we know Collatz holds for m.

Ex 3:

Inductive Perm3 : [X] → [X] → Prop :=

| swap12 (a b c : X) :

Perm3 [a; b; c] [b; a; c]

swap12 says: [a b c] is Perm3 of [b a c]

| swap23 (a b c : X) :
    Perm3 [a; b; c] [a; c; b]

swap23 says: [a b c] is perm3 of [a c b]

| trans (l1 l2 l3 : [X]) :
    Perm3 l1 l2 → Perm3 l2 l3
        → Perm l1 l3

If l1, l2 are perms and l2, l3 perms,
then l1, l3 are perms.

Inverting on evidences

Evidences are inductive propositions in the hypothesis.
Consider ev 2 is in the hypothesis, we
should we be able to infer ?

Of course, we can compute
    ev_SS 2 (ev 2) = ev 4

But more interestingly, we should be able
to infer

ev 0

I.e., ev 2 → ev 0 via inversion.

Let's do more inversions:

- ev 1 → False

  ← inverting on improbable construction turns it into False!

- [n; m] = [0; 0] → n = 0, m = 0

  though inversion is the primary use, inversion also does injection. It actually does quite a lot more.

- ev n → $\overbrace{n = 0}^{\text{corresponds to ev\_0}}$ \/ $\underbrace{\text{ev } (n - 2)}_{\text{corresponds to ev\_SS}}$

- ev (2 + n) → ev n

- 2 <= 1 → False

Internally, coq is trying to find a \`le\` constructor that satisfies 2 <= 1:

Can 2 <= 1 be le_n? No.

Can 2 <= 1 be le_S? Yes but if only 2 <= 0 is le.

Can 2 <= 1 be le_S ? [illegible]

Can 2 <= 0 be le_n? No.

Can 2 <= 0 be le_S ? No.

No because we can't continue to decrement 0.

- S n <= 1 → S n = 1 ✓

  S n <= 0

Inducting on inductive props

The tricky stuff!

As we know, induction is destruct plus assuming previous case, so let's work through destruct first.

Ex 1

n : nat

E : ev n

(1/1) Goal

Even n

after destruct E

n: nat

E : ev n'

(1/2) Goal

Even 0

(2/2) Goal

Even (2 + n')

What's going on?

Recall that inverting ev n gives

$n = 0 \ \backslash/ \ ev\ (n-2)$

So, • in the first case, n=0, hence Even = 0.

• in the second case, ev (n-2) means that there is a n' = n-2 such that ev n'.

added as hypothesis.

The new variable n' is used to rewrite all hypotheses and the goal.

Hence, Even n becomes Even (2 + n')

now, what about induction?

n: nat

E: ev n

(1/1) Goal

Even n

after induction E

n : nat

E : ev n'

H : Even n' → the only difference!

(1/2) Goal

Even 0

(2/2) Goal

Even (2 + n')

okay. Why are we able to infer Even n'?

Ex 2

m, n, o : nat

H : $m \leq n$

Ho : $n \leq 0$

(1/1) ______

$m \leq 0$

After induction [$n \le o$]:

m, n : nat

H : $m \le n$

(1/2) ———

$m \le n$

o got replaced by n.

m, n: nat

H: $m \le n$

o': nat

Hno': $n \le o'$

IHmo': $m \le o'$

(2/2) ———

$m \le S\ o'$

$n \le o$ either means

- le_n, in which case n matches the argument, and we must have $o = n$ to get $n \le n$.

- le_S, in which case we must have $n \le o'$, where $o = o' + 1$

$o' = o - 1$

$H: a > b$

case gt_base: a matches, and

$$S\ a' > a'$$

$$a' = a - 1, \quad b = a'$$

$$H_0: a' > c$$

$$G: S\ a' > c$$

case gt_S: a, b matches with n, m,

$$a > b \iff (n+2) > (m+1)$$

$$a = n+2$$
$$b = m+1$$

H_0: $b > c$ becomes

$$m+1 > c$$

G: $a > c$ becomes

$$n+2 > C$$