Propaganda for the Dark Side

Jannis Limperg

University of Freiburg

DeepSpec Summer School 2017

```
Inductive Vec (A : Type) : nat → Type :=
| nil : Vec A 0
| cons : ∀ {n}, A → Vec A n → Vec A (S n).

Inductive Fin : nat → Type :=
| fzero {n} : Fin (S n)
| fsuc {n} : Fin n → Fin (S n).
```

```
Fixpoint index {A n} (xs : Vec A n) (f : Fin n) : A.
  refine (
    match xs in (Vec _ n') return n = n' → _ with
    | nil ⇒ fun Eqn ⇒ _
    | cons x ys ⇒ fun Eqn ⇒
        match f in (Fin n'') return n = n'' → _ with
        | fzero ⇒ fun _ ⇒ x
        | fsuc f' ⇒ fun Eqn'' ⇒ _
        end
        eq_refl
    end
    eq_refl
  ).
  - rewrite Eqn in f. inversion f.
  - subst. injection Eqn''. intro H. rewrite H in ys.
    exact (index _ _ ys f').
Defined.
```

$$
\begin{aligned}
&\mathsf{index} : \forall\ \{A\ n\} \to \mathsf{Vec}\ A\ n \to \mathsf{Fin}\ n \to A \\
&\mathsf{index}\ []\ \quad\quad\quad\ () \\
&\mathsf{index}\ (x :: xs)\ \ \mathsf{fzero}\ \quad\quad = x \\
&\mathsf{index}\ (x :: xs)\ \ (\mathsf{fsuc}\ fin)\ \ = \mathsf{index}\ xs\ fin
\end{aligned}
$$

kthxbye

```
Fixpoint index1 {A n} (xs : Vec A n) (f : Fin n) : A.
  inversion xs as [Eqn|? x ys Eqn].
  - rewrite ← Eqn in f. inversion f.
  - destruct f.
    × exact x.
    × injection Eqn. intros. subst.
      exact (index1 _ _ ys f).
Defined.
```

```
module _ (A : Set) where

  data _∈_ (a : A) : List A → Set where
    here  : ∀ {as} → a ∈ (a :: as)
    there : ∀ {a' as} → a ∈ as → a ∈ (a' :: as)

  del : ∀ {a} as → a ∈ as → List A
  del (a :: as) here      = as
  del (a :: as) (there p) = a :: del as p

  ∈-heq : ∀ {a a' as} → (p : a ∈ as) → a' ∈ as → (a' ∈ del as p) ⊎ (a ≡ a')
  ∈-heq here  here      = inj₂ refl
  ∈-heq here  (there q) = inj₁ q
  ∈-heq (there p) here  = inj₁ here
  ∈-heq (there p) (there q) with ∈-heq p q
  ... | inj₁ q' = inj₁ (there q')
  ... | inj₂ eq = inj₂ eq

  ∈-heq-eq :
    ∀ {a a' as eq}(p : a ∈ as)(q : a' ∈ as) → ∈-heq p q ≡ inj₂ eq → subst (_∈ as) eq p ≡ q
  ∈-heq-eq here  here      refl = refl
  ∈-heq-eq here  (there q) ()
  ∈-heq-eq (there p) here  ()
  ∈-heq-eq (there p) (there q) r with ∈-heq p q | inspect (∈-heq p) q
  ∈-heq-eq (there p) (there q) () | inj₁ _ | _

  ∈-heq-eq (there p) (there q) refl | inj₂ refl | [ eq ] = cong there (∈-heq-eq p q eq)
```

```
Section del_val.
  Context {T : Type}.
  Variable ku : T.

  Fixpoint del_member (ls : list T) (m : member ku ls) : list T :=
    match m with
    | @MZ _ _ l ⇒ l
    | @MN _ _ ku' _ m ⇒ ku' :: del_member m
    end.
End del_val.

Section member_heq.
  Context {T : Type}.

  Fixpoint member_heq (l r : T) (ls : list T) (m : member l ls)
  : member r ls → member r (del_member m) + (l = r) :=
    match m as m in member _ ls
          return member r ls → member r (del_member m) + (l = r)
    with
    | @MZ _ _ ls ⇒ fun b : member r (l :: ls) ⇒
                    match b in member _ (z :: ls)
                          return l = z → member r (del_member (@MZ _ _ ls)) + (l = r)
                    with
                    | MZ ⇒ @inr _ _
                    | MN m' ⇒ fun pf ⇒ inl m'
                    end eq_refl
    | @MN _ _ l' ls' mx ⇒ fun b : member r (l' :: ls') ⇒
                           match b in member _ (z :: ls)
                                 return (member _ ls → member _ (del_member mx) + (_ = r)) →
                                        member r (del_member (@MN _ _ _ _ mx)) + (_ = r)
                           with
                           | MZ ⇒ fun _ ⇒ inl MZ
                           | MN m' ⇒ fun f ⇒ match f m' with
                                             | inl m ⇒ inl (MN m)
                                             | inr pf ⇒ inr pf
                                             end
                           end (fun x ⇒ @member_heq _ _ _ mx x)
    end.
```

```
Definition member_match {t t' : T} {ts}
           (P : ∀ t t' ts, member t (t' :: ts) → Type)
           (Hz : P t' t' ts (MZ))
           (Hn : ∀ m : member t ts, P t t' ts (MN m))
: ∀ m, P t t' ts m :=
  fun m ⇒
    match m in member _ (x :: y)
          return P x x y (@MZ _ x y) → (∀ m0 : member t y, P t x y (MN _)) → P t x y m
    with
    | MZ ⇒ fun X _ ⇒ X
    | MN m ⇒ fun _ X ⇒ X m
    end Hz Hn.

Definition inj_inr {T U a b} (pf : @inr T U a = inr b)
: a = b :=
  match pf in _ = X return match X with
                           | inl _ ⇒ True
                           | inr X ⇒ a = X
                           end
  with
  | eq_refl ⇒ eq_refl
  end.

Lemma member_heq_eq : ∀ {l l' ls} (m1 : member l ls) (m2 : member l' ls) pf,
    member_heq m1 m2 = inr pf →
    match pf in _ = X return member X ls with
    | eq_refl ⇒ m1
    end = m2.
Proof.
  induction m1.
  { refine (@member_match _ _ _ (fun l' a ls m2 ⇒ ∀ (pf : a = l'),
                                   member_heq (@MZ _ a ls) m2 = inr pf →
                                   match pf in (_ = X) return (member X (a :: ls)) with
                                   | eq_refl ⇒ MZ
                                   end = m2) _ _).
    { simpl. intros.
      refine
```

```
      match H in _ = X
            return match X with
                   | inl _ ⇒ True
                   | inr X ⇒ match X in (_ = X)
                                    return (member _ (l :: ls)) with
                             | eq_refl ⇒ MZ
                             end = MZ
                   end
      with
      | eq_refl ⇒ eq_refl
      end. }
    { simpl. inversion 1. } }
  { intro.
    revert IHm1. revert m1. revert m2.
    refine (@member_match _ _ _ (fun l' l0 ls m2 ⇒ ∀ m1 : member l ls,
                                   (∀ (m3 : member l' ls) (pf : l = l'),
                                      member_heq m1 m3 = inr pf →
                                      match pf in (_ = X) return (member X ls) with
                                      | eq_refl ⇒ m1
                                      end = m3) →
                                   ∀ pf : l = l',
                                     member_heq (MN m1) m2 = inr pf →
                                     match pf in (_ = X) return (member X (l0 :: ls)) with
                                     | eq_refl ⇒ MN m1
                                     end = m2) _ _).
    { inversion 2. }
    { intros.
      specialize (H m pf). simpl in *.
      destruct (member_heq m1 m).
      { inversion H0. }
      { specialize (H (f_equal _ (inj_inr H0))).
        subst. reflexivity. } } }
Defined.

End member_heq.
```