Chapter 1

Library LNMItImp

LNMItImp.v Version 2.5 March 2009 needs impredicative Set, runs under V8.2, later tested with version 8.2pl1

this is the implementation part where impredicative methods justify LNMIt, based on ideas of Venanzio Capretta to represent simultaneous inductive-recursive definitions

this is code that no longer conforms to the description in the article "An induction principle for nested datatypes in intensional type theory" by the author, that appeared in the Journal of Functional Programming, since it now uses type classes instead of the record **EFct** and the type **pEFct**, as well as for **mon** and **NAT**

forms part of the code that comes with a submission to the journal Science of Computer Programming

```
Require Import Utf8.

Require Import LNMItPred.
```

a device to parameterize the implementation

```
Module Type LNMItParam.

Parameter F: k2.
```

the only general requirement: F preserves extensional functors

```
Instance FpEFct : pEFct F.
```

the only axiom we want to use: proof irrelevance

```
Axiom pirr : ∀ (A: Prop) (a1 a2: A), a1 = a2.

End LNMItParam.
```

the implementation of LNMIt for a given F and FpEFct

```
Module LNMIt(LNMP: LNMItParam) <: LNMIt_Type with Definition F:=LNMP.F
with Definition FpEFct:=LNMP.FpEFct.

Import LNMP.
```

Definition F:= *F*.
Definition FpEFct:= *FpEFct*.
Definition pirr:= *pirr*.

the type of the iterator, parameterized over the source constructor

Definition MItPretype (S: k1) : Type :=
∀ G : k1, (∀ X : k1, $X \subseteq G$ → F $X \subseteq G$) → $S \subseteq G$.

the following inductive definition is only a record

Inductive **mu2E**: Set → Set :=
inE : ∀ (G: k1)(ef: **EFct** G)(G': k1)(m': **mon** G')(it: MItPretype G')(j: $G \subseteq G'$),
NAT j → F $G \subseteq$ **mu2E**.

the rough intention is that we only want to use inE with G':= mu2, m':= mapmu2 and it:= MIt.

We do not want to have j as implicit argument due to eta-problems.

Implicit Arguments inE [G G' m' A].

the preliminary map term

Instance mapmu2E : **mon mu2E**.

the preliminary iterator with source **mu2E** does not iterate at all

Definition MItE : MItPretype **mu2E**.

Lemma MItERed : ∀ (G: k1)(s: ∀ X : k1, $X \subseteq G$ → F $X \subseteq G$)(A: Set)
(X: k1)(ef: **EFct** X)(G': k1)(m': **mon** G')(it: MItPretype G')
(j: $X \subseteq G'$) n (t: F X A),
MItE s (inE ef it j n t) = s X (fun A ⇒ (it G s A) ∘ (j A)) A t.

single out the good elements of **mu2E** A

Inductive **mu2Echeck** : ∀ (A: Set), **mu2E** A → Prop :=
inEcheck : ∀ (G: k1)(ef: **EFct** G)(j: $G \subseteq$ **mu2E**)(n: **NAT** j),
(∀ (A: Set)(t: G A), **mu2Echeck** (j A t)) →
∀ (A: Set)(t: F G A),
mu2Echeck (inE ef MItE (fun A t ⇒ j A t) n t).

this expansion of j will later be needed

Implicit Arguments inEcheck [G A].

Definition μ_0 (A: Set) := {r: **mu2E** A | **mu2Echeck** r}.
this is a convenient form to write *sig* (**mu2Echeck**(A:= A)).

Definition μ : k1 := fun A ⇒ μ_0 A.

Definition mu2cons (A: Set)(r: **mu2E** A)(p: **mu2Echeck** r) : μ A :=
exist (fun r : **mu2E** A ⇒ **mu2Echeck** r) r p.

Implicit Arguments mu2cons [A].

a non-iterative definition of the monotonicity witness:

Instance mapmu2 : **mon** μ.

the usual projections from a *sig* are proj1_sig and proj2_sig

Definition pi1: $\mu \subseteq$ **mu2E**.

Definition MItType: Type := MItPretype μ.

Definition MIt0 : MItType.

This has been very easy since μ is only the source type of the transformation. Therefore, we did not even need destruct r. Had we used it nevertheless, we would have encountered problems with eta.

the specification dictates this second eta-expansion

Definition MIt : MItPretype μ := fun $G\ s\ A\ r \Rightarrow$ MIt0 $s\ r$.

Lemma pi2 : $\forall(A\colon$ Set$)(r\colon\ \mu\ A)$, **mu2Echeck** (pi1 r).

first projection commutes with the maps

Lemma pi1mapmu2 : $\forall\ (A\ B\colon$ Set$)(f\colon\ A \to B)(r\colon\ \mu\ A)$, pi1 (map $f\ r$) = map f (pi1 r).

the type of the future datatype constructor In

Definition InType : Type :=
$\forall\ (X\colon$ k1$)(ef\colon$ **EFct** $X)(j\colon\ X \subseteq \mu)$, **NAT** $j \to$ F $X \subseteq \mu$.

Definition pi1' $(X\colon$ k1$)(j\colon\ X \subseteq \mu)\colon\ X \subseteq$ **mu2E**.

Lemma pi1'pNAT : $\forall\ (X\colon$ k1$)(m\colon$ **mon** $X)(j\colon\ X \subseteq \mu)$, **NAT** $j \to$ **NAT** (pi1' j).

Lemma pi2' : $\forall(X\colon$ k1$)(j\colon\ X \subseteq \mu)(A\colon$ Set$)(t\colon\ X\ A)$, **mu2Echeck** (pi1' $j\ A\ t$).

in is reserved for Coq, so the datatype constructor will be called In

Definition In : InType.

Lemma mapmu2Red : $\forall\ (A\colon$ Set$)(G\colon$ k1$)(ef\colon$ **EFct** $G)(j\colon\ G \subseteq \mu)$
$(n\colon$ **NAT** $j)(t\colon$ F $G\ A)(B\colon$ Set$)(f\colon\ A \to B)$,
map f (In $ef\ n\ t$) = In $ef\ n$ (m $f\ t$).

Lemma MItRed : $\forall\ (G$: k1$)$
$(s$: $\forall\ X$: k1, $X \subseteq G \to$ F $X \subseteq G)(X$: k1$)(ef\colon$ **EFct** $X)(j\colon\ X \subseteq \mu)$
$(n\colon$ **NAT** $j)(A\colon$ Set$)(t\colon$ F $X\ A)$,
MIt s (In $ef\ n\ t$) = $s\ X$ (fun $A \Rightarrow$ (MIt s $(A{:=}\mathrm{A})) \circ (j\ A))\ A\ t$.

our desired induction principle, first just as a proposition

Definition mu2IndType : Prop :=
$\forall\ P$: $(\forall\ A$: Set, $\mu\ A \to$ Prop$)$,
$(\forall\ (X$: k1$)(ef\colon$ **EFct** $X)(j$: $X \subseteq \mu)(n\colon$ **NAT** $j)$,
$(\forall\ (A$: Set$)\ (x$: $X\ A)$, $P\ A\ (j\ A\ x)) \to$
$\forall\ (A\colon$ Set$)(t$: F $X\ A)$, $P\ A$ (In $ef\ n\ t$)$) \to$
$\forall\ (A$: Set$)\ (r$: $\mu\ A)$, $P\ A\ r$.

Scheme *mu2EcheckInd* := Induction for *mu2Echeck* Sort Prop.

the first consequence of proof irrelevance we will use is injectivity of pi1

Lemma mu2pirr : $\forall$ (A: Set)(r_1 r_2: μ A), pi1 $r_1 =$ pi1 $r_2 \rightarrow r_1 = r_2$.

the second consequence of proof irrelevance we will use: uniqueness of naturality proofs

Lemma UNP : $\forall$(X Y: k1)(j: $X \subseteq Y$)(mX: **mon** X)(mY: **mon** Y)
(n_1 n_2: **NAT** j), $n_1 = n_2$.

the main theorem of the whole approach

Lemma mu2Ind : mu2IndType.
equates n and pi1'pNAT n_1

End LNMIt.

Chapter 2

Library LNGMItImp

LNGMItImp.v Version 1.3a March 2009 needs impredicative Set, runs under V8.2, later tested with version 8.2pl1

this is the implementation part where impredicative methods justify *LNGMIt* by reduction to LNMIt

forms part of the code that comes with a submission to the journal Science of Computer Programming

```
Require Import Utf8.

Require Import LNMItPred.
Require Import LNGMItPred.
```

right Kan extension along H

```
Definition GRan (H G: k1) : k1 := fun A ⇒ ∀ B: Set, (A → H B) → G B.

Definition LeqRan (X H G: k1) : X <_{H} G → X ⊆ GRan H G.

Definition RanLeq(X H G: k1) : X ⊆ GRan H G → X <_{H} G .
```

end of preparations for the following module that represents the proof of Proposition 1 of the paper

```
Module LNGMItBaseImp(M: LNMIt_Type) <: LNGMIt_Type.

Import M.

Module LNM:= M.
Definition F:= F.
Definition FpEFct:= FpEFct.
Definition μ0 := μ0.
Definition μ := μ.
Definition mapmu2 := mapmu2.
Definition MItType:= MItType.
Definition MIt0 := MIt0.
```

```
Definition MIt := MIt.
Definition InType := InType.
Definition In := In.
Definition mapmu2Red:= mapmu2Red.
Definition MItRed:= MItRed.
Definition mu2IndType:= mu2IndType.
Definition mu2Ind:= mu2Ind.
```

```
Section GMIt0.
```

```
Variables H G: k1.
Variable s: ∀ X: k1, X <_{H} G → F X <_{H} G.
```

```
Definition sMItGMIt: ∀ X : k1, X ⊆ GRan H G → F X ⊆ GRan H G.
```

The following definition corresponds to the definition in the proof of Proposition 1 in the paper.

```
Definition GMIt0 : μ <_{H} G := RanLeq (MIt sMItGMIt).
```

```
Lemma GMIt0Red : ∀(A B: Set)(f: A → H B)(X: k1)(ef: EFct X)(j: X ⊆ μ)
        (n: NAT j)(t: F X A),
                GMIt0 f (In ef (j:= j) n t) =
          s (fun (A B: Set) (f: A → H B) ⇒ (GMIt0 f) ∘ (j A)) f t.
```

```
End GMIt0.
```

```
Definition GMIt: ∀ (H G: k1), (∀ X: k1, X <_{H} G → F X <_{H} G) → μ <_{H} G
  := fun (H G: k1) s (A: Set) B f t ⇒ GMIt0(H:= H)(G:= G)(A:= A) s B f t.
```

```
Lemma GMItRed : ∀ (H G: k1)(s: ∀ X: k1, X <_{H} G → F X <_{H} G)
  (A B: Set)(f: A → H B)(X: k1)(ef: EFct X)(j: X ⊆ μ)(n: NAT j)(t: F X A),
                GMIt s B f (In ef (j:= j) n t) =
          s X (fun (A B: Set) (f: A → H B) ⇒ (GMIt s B f) ∘ (j A)) A B f t.
```

```
End LNGMItBaseImp.
```

```
Require Import LNMItImp.
```

```
Module LNGMItDefImpImp(LNMP: LNMItParam).
```

```
Module LNMItBaseImp := LNMIt LNMP.
Module LNGMItBaseImpImp := LNGMItBaseImp LNMItBaseImp.
```

```
Import LNGMItBaseImpImp.
```

```
Lemma GMItRed : ∀ (H G: k1)(s: ∀ X: k1, X <_{H} G →
    F X <_{H} G)(A B: Set)(f: A → H B)(X: k1)(ef: EFct X)(j: X ⊆ μ)(n: NAT j)(t:
F X A),
      GMIt s _ f (In ef (j:= j) n t) =
          s X (fun (A B: Set) (f: A → H B) ⇒ (GMIt s _ f) ∘ (j A)) A B f t.
```

```
Module LNGMItDefImpImp := LNGMItDef LNGMItBaseImpImp.
```

`Module` LNMItDefImp := LNGMItDefImpImp.LNMItDef.

`Import` *LNGMItDefImpImp*.
`Import` *LNMItDefImp*.

`Lemma` GMItRedCan : $\forall (H\ G\colon \mathsf{k1})(s\colon \forall\ X\colon \mathsf{k1},\ X <\_\{\mathrm{H}\}\ G \to \mathsf{F}\ X <\_\{\mathrm{H}\}\ G)$
$(A\ B\colon \mathtt{Set})(f\colon A \to H\ B)(t\colon \mathsf{F}\ \mu\ A),$
$\mathsf{GMIt}\ s\ \_\ f\ (\mathsf{InCan}\ t) = s\ \_\ (\mathsf{GMIt}\ s)\ \_\ \_\ f\ t.$

Hence, in the impredicative encoding, the additional reduction rules are part of the convertibility relation.

`End` LNGMItDefImpImp.