Illustrations of (Co-)Inductive Definitions

Roger Bishop Jones

Date: 2012/08/11 21:01:53

Abstract

This document provides examples of the use of the facilities provided in t007.doc.

http://www.rbjones.com/rbjpub/pp/doc/t008.pdf

Id: t008.doc,v 1.7 2012/08/11 21:01:53 rbj Exp

Contents

References

[1] Roger Bishop Jones. Inductive, Co-Inductive and Psuedo-(Co-)Inductive Definitions in ProofPower. *RBJones.com*, 2010. `http://www.rbjones.com/rbjpub/pp/doc/t007.pdf`.

[2] Roger Bishop Jones. Membership Structures. *RBJones.com*, 2010. `http://www.rbjones.com/rbjpub/pp/doc/t004.pdf`.

1 INTRODUCTION

The document follows, as far as is reasonable, the structure of [1] in providing material illustrating (and testing) the facilities for (co-)inductive definition of sets and types provided in that document.

1.1 Formalities

Create new theory "fixp_egs".

SML

```
open_theory "fixp";
force_new_theory "fixp-egs";
set_merge_pcs["hol", "'savedthm_cs_∃_proof"];
```

2 FIXED POINTS

I probably won't supply any direct examples of this material. It is used indirectly in all that follows.

3 INDUCTIVE DEFINITIONS OF SETS

3.1 Hereditarily Over a Function

The example I had in mind which made me think this kind of definition useful is the inductive definition of the theorems of some deductive system. This belongs here because taking the rules of a deductive system as constructors gives an inductive definition in which the constructor is not one-one. It also raises a question about inductive definition of functions over the type, since a naive interpretation of the 'content' relation would not be well-founded.

Another example would be the definition of recursive function.

These things are easy to do without any special machinery so it might be interesting to try one which has already been done that way, for example my definition of a first order property in [2].

SML

```
new_parent "membership";
```

The inductive definition is that of *fof*. How would that look if formulated as the fixed point of a function?

HOL Constant

$FofCSet: ((STRING, 'a)\ PPMS\ SET \times (STRING, 'a)PPMS)\ SET$

$$
\begin{aligned}
&FofCSet = \{(ppmss, ppms1) \mid \\
&\quad ppmss = \{\} \land (\exists s1\ s2 \bullet ppms1 = s1 =_p s2 \lor ppms1 = s1 \in_p s2) \\
&\quad \lor (\exists p\ s \bullet ppmss = \{p\} \land ppms1 = (\exists_p\ s\ p) \lor ppms1 = \neg_p\ p) \\
&\quad \lor (\exists p1\ p2 \bullet ppmss = \{p1;\ p2\} \land ppms1 = p1 \land_p p2) \\
&\quad \}
\end{aligned}
$$

This is turned into a set of formulae thus:

HOL Constant

$Fof: (STRING, 'a)PPMS\ SET$

$$Fof = HeredFun\ (Rules2Fun\ FofCSet)$$

From this we should be able to get an induction principle by instantiating and expanding *HeredFun_induction_thm2*.

However, this looks quite hard to massage into a nicely formulated induction property.

By comparison coding up the required closure property directly gives:

HOL Constant

$FofCProp: (STRING, 'a)PPMS\ SET \to BOOL$

$$
\begin{aligned}
&\forall\ ppmss \bullet FofCProp\ ppmss \Leftrightarrow \\
&\quad (\forall s1\ s2 \bullet s1 =_p s2 \in ppmss \land s1 \in_p s2 \in ppmss) \\
&\quad \land (\forall p \bullet p \in ppmss \Rightarrow \neg_p\ p \in ppmss \land \forall s \bullet (\exists_p\ s\ p) \in ppmss) \\
&\quad \land (\forall p1\ p2 \bullet p1 \in ppmss \land p2 \in ppmss \Rightarrow p1 \land_p p2 \in ppmss)
\end{aligned}
$$

HOL Constant

$Fof_2: (STRING, 'a)PPMS\ SET$

$$Fof_2 = \bigcap \{s \mid FofCProp\ s\}$$

$fof\_induction\_thm \vdash \forall s \bullet FofCProp\ s \Rightarrow Fof_2 \subseteq s$

$$
\begin{aligned}
&fof\_induction\_thm2 \vdash \forall\ s \\
&\quad \bullet (\forall\ s1\ s2 \bullet s1 =_p s2 \in s \land s1 \in_p s2 \in s) \\
&\qquad \land (\forall\ p \bullet p \in s \Rightarrow \neg_p\ p \in s \land (\forall\ s' \bullet \exists_p\ s'\ p \in s)) \\
&\qquad \land (\forall\ p1\ p2 \bullet p1 \in s \land p2 \in s \Rightarrow p1 \land_p p2 \in s) \\
&\quad \Rightarrow Fof_2 \subseteq s
\end{aligned}
$$

3.2 Hereditarily Over a Relation

3.3 Hereditarily Over a Property

4 INDUCTIVE DEFINITIONS OF SETS

The examples here are examples which do not use the machinery for generating constructor functions.

4.1 Sets Defined Using CCP's

4.1.1 Hereditarily Pure Functions

4.1.2 Hereditarily Pure Functors and Categories

5 CODING CONSTRUCTIONS

5.1 HOL Types and Terms

Here is a simple type description to work from, the types and terms of HOL: Note that for each constructor is supplied the type of the constructor and a predicate over the domain of the constructor. In this example the predicates enforce constraints on the names of constants and variables requiring that names beginning with ' are variable names.

The type variables $'TYPE$ and $'TERM$ give the names of the sets or types which are being defined.

SML

```
val constructor_types = [
    (⌜MkVarType: STRING → 'TYPE⌝,
                ⌜λx:STRING• Hd x = "'"⌝),
    (⌜MkCType: STRING → 'TYPE LIST →'TYPE⌝,
                ⌜λ(x:STRING, y:'TYPE LIST)• ¬ Hd x = "'"⌝),
    (⌜MkVarTerm: STRING → 'TYPE →'TERM⌝,
                ⌜λ(x:STRING, y:'TYPE)• Hd x = "'"⌝),
    (⌜MkCTerm: STRING → 'TYPE →'TERM⌝,
                ⌜λ(x:STRING, y:'TYPE)• ¬ Hd x = "'"⌝),
    (⌜MkLamTerm: STRING → 'TYPE → 'TERM → 'TERM⌝,
                ⌜λ(x:STRING, y:'TYPE, z:'TERM)• Hd x = "'"⌝),
    (⌜MkAppTerm: 'TERM → 'TERM → 'TERM⌝,
                ⌜λ(x:'TERM, y:'TERM)• T⌝)
];
```

This system of type specifications can be translated into an expression for a fixedpoint as follows:

SML

```
set_flag ("pp_use_alias", true);
declare_ctk_aliases ntree_ctk ctk_aliases;
val fixp_expression = translate_sig ntree_ctk constructor_types;
undeclare_ctk_aliases ntree_ctk ctk_aliases;
fixp_expression;
```

Which yields:

```
val fixp_expression =
  ⌜Θ
   [[Ξ (φ γ) (φ ρ) (λ x• Head x = "");
        Ξ (φ γ × φ I) (φ ρ × φ (ω 1)) (λ (x, y)• ¬ Head x = "")];
      [Ξ (φ γ × I) (φ ρ × ω 1) (λ (x, y)• Head x = "");
         Ξ (φ γ × I) (φ ρ × ω 1) (λ (x, y)• ¬ Head x = "");
         Ξ (φ γ × I × I) (φ ρ × ω 1 × ω 2) (λ (x, y, z)• Head x = "");
         Ξ (I × I) (ω 2 × ω 2) (λ (x, y)• T)]]⌝ : TERM
```

Without the aliases that is:

```
val fixp_expression =
  ⌜FR
   [[CR (NTreeeMkList MkLeafTreee) (NTrListC NTrLeafC) (λ x• Head x = "");
        CR
           (NTreeeMkProd (NTreeeMkList MkLeafTreee) (NTreeeMkList I))
           (NTrProdC (NTrListC NTrLeafC) (NTrListC (NTreeeTagC 1)))
           (λ (x, y)• ¬ Head x = "")];
      [CR
           (NTreeeMkProd (NTreeeMkList MkLeafTreee) I)
           (NTrProdC (NTrListC NTrLeafC) (NTreeeTagC 1))
           (λ (x, y)• Head x = "");
         CR
            (NTreeeMkProd (NTreeeMkList MkLeafTreee) I)
            (NTrProdC (NTrListC NTrLeafC) (NTreeeTagC 1))
            (λ (x, y)• ¬ Head x = "");
         CR
            (NTreeeMkProd (NTreeeMkList MkLeafTreee) (NTreeeMkProd I I))
            (NTrProdC
                (NTrListC NTrLeafC)
                (NTrProdC (NTreeeTagC 1) (NTreeeTagC 2)))
            (λ (x, y, z)• Head x = "");
         CR
            (NTreeeMkProd I I)
            (NTrProdC (NTreeeTagC 2) (NTreeeTagC 2))
            (λ (x, y)• T)]]⌝
: TERM
```

which has type:

SML

```
type_of fixp_expression;
```

```
val it = ⌜:(ℕ, CHAR) TREEE SET⌝ : TYPE
```

This set contains the representatives of each type discriminated by tags, so to recover the two sets we need to filter the one as follows:

SML

```
declare_ctk_aliases ntree_ctk ctk_aliases;
val type_rep = ⌜{x | ∃y• y ∈ ML⌜fixp_expression⌝ ∧ IsTag 1 y ∧ x = UnTag y}⌝;
```

```
val term_rep =
   ⌜{x
   |∃ y
     • y
           ∈ Θ
             [[Ξ (φ ρ) (φ ρ) (λ x• Head x = "");
                    Ξ (φ ρ × φ I) (φ ρ × φ (ω 1)) (λ (x, y)• ¬ Head x = "")];
                 [Ξ (φ ρ × I) (φ ρ × ω 1) (λ (x, y)• Head x = "");
                   Ξ (φ ρ × I) (φ ρ × ω 1) (λ (x, y)• ¬ Head x = "");
                   Ξ
                       (φ ρ × I × I)
                       (φ ρ × ω 1 × ω 2)
                       (λ (x, y, z)• Head x = "");
                   Ξ (I × I) (ω 2 × ω 2) (λ (x, y)• T)]]
          ∧ IsTag 2 y
          ∧ x = UnTag y}⌝ : TERM
```

SML

```
val term_rep = ⌜{x | ∃y• y ∈ ML⌜fixp_expression⌝ ∧ IsTag 2 y ∧ x = UnTag y}⌝;
```


```
val term_rep =
   ⌜{x
   |∃ y
     • y
           ∈ Θ
             [[Ξ (φ ρ) (φ ρ) (λ x• Head x = "");
                    Ξ (φ ρ × φ I) (φ ρ × φ (ω 1)) (λ (x, y)• ¬ Head x = "")];
                 [Ξ (φ ρ × I) (φ ρ × ω 1) (λ (x, y)• Head x = "");
                   Ξ (φ ρ × I) (φ ρ × ω 1) (λ (x, y)• ¬ Head x = "");
                   Ξ
                       (φ ρ × I × I)
                       (φ ρ × ω 1 × ω 2)
                       (λ (x, y, z)• Head x = "");
                   Ξ (I × I) (ω 2 × ω 2) (λ (x, y)• T)]]
          ∧ IsTag 2 y
          ∧ x = UnTag y}⌝ : TERM
```


Let's check this out by trying a proof that the set of types is non-empty.

SML

```
set_goal ([], ⌜∃t• t ∈ ML⌜type_rep⌝⌝);
a (rewrite_tac (map get_spec [⌜Θ⌝, ⌜Ξ⌝, ⌜NTrProdC⌝, ⌜NTreeeMkProd⌝]));
```

6 MAKING NEW TYPES

Here are some examples I would like to be able to handle.

algebras e.g. a type of groups, these are essentially subtypes of labelled product types in which the projections yield objects conforming to the signature of the relevant algebra. This is therefore a special case of the general inductive datatypes problem (if an implementation of that allowed predicates).

types from algebras e.g. given a type of groups, make a type out of a presentation of a particular group.

from hereditatily sets e.g. a type of hereditarily pure functions, or a pair of types for the pure concrete categories and functors.

7 The Theory fixp-egs

7.1 Parents

membership *fixp*

7.2 Constants

$\boldsymbol{FofCSet}$	$(STRING, 'a)\ PPMS\ \mathbb{P} \leftrightarrow (STRING, 'a)\ PPMS$
$\boldsymbol{Fof}$	$(STRING, 'a)\ PPMS\ \mathbb{P}$
$\boldsymbol{FofCProp}$	$(BOOL, (STRING, 'a)\ PPMS\ \mathbb{P})\ VA$
$\boldsymbol{Fof_2}$	$(STRING, 'a)\ PPMS\ \mathbb{P}$

7.3 Aliases

$\boldsymbol{\upsilon}$	$NTreeeTag : ((((\mathbb{N}, 'b)\ TREEE, 'a)\ VA, ((\mathbb{N}, 'b)\ TREEE, 'a)\ VA)\ VA, \mathbb{N})\ VA$
$\boldsymbol{\omega}$	$NTreeeTagC : (((\mathbb{N}, 'a)\ TREEE\ \mathbb{P}, (\mathbb{N}, 'a)\ TREEE)\ VA, \mathbb{N})\ VA$
$\boldsymbol{\gamma}$	$MkLeafTreee : (('a, 'b)\ TREEE, 'b)\ VA$
$\times$	$NTreeeMkProd : ((((\mathbb{N}, 'c)\ TREEE, 'a \times 'b)\ VA, ((\mathbb{N}, 'c)\ TREEE, 'b)\ VA)\ VA, ((\mathbb{N}, 'c)\ TREEE, 'a)\ VA)\ VA$
$+$	$NTreeeMkSum : ((((\mathbb{N}, 'c)\ TREEE, 'a + 'b)\ VA, ((\mathbb{N}, 'c)\ TREEE, 'b)\ VA)\ VA, ((\mathbb{N}, 'c)\ TREEE, 'a)\ VA)\ VA$
$\boldsymbol{\phi}$	$NTreeeMkList : (((\mathbb{N}, 'b)\ TREEE, 'a\ LIST)\ VA, ((\mathbb{N}, 'b)\ TREEE, 'a)\ VA)\ VA$
$\boldsymbol{\rho}$	$MkLeafTreee : ((\mathbb{N}, CHAR)\ TREEE, CHAR)\ VA$
$\times$	$NTrProdC : ((((\mathbb{N}, 'c)\ TREEE\ \mathbb{P}, 'a \times 'b)\ VA, ((\mathbb{N}, 'c)\ TREEE\ \mathbb{P}, 'b)\ VA)\ VA, ((\mathbb{N}, 'c)\ TREEE\ \mathbb{P}, 'a)\ VA)\ VA$
$+$	$NTrSumC : ((((\mathbb{N}, 'c)\ TREEE\ \mathbb{P}, 'a + 'b)\ VA, ((\mathbb{N}, 'c)\ TREEE\ \mathbb{P}, 'b)\ VA)\ VA, ((\mathbb{N}, 'c)\ TREEE\ \mathbb{P}, 'a)\ VA)\ VA$
$\boldsymbol{\phi}$	$NTrListC : (((\mathbb{N}, 'b)\ TREEE\ \mathbb{P}, 'a\ LIST)\ VA, ((\mathbb{N}, 'b)\ TREEE\ \mathbb{P}, 'a)\ VA)\ VA$
$\boldsymbol{\rho}$	$NTrLeafC : ((\mathbb{N}, 'b)\ TREEE\ \mathbb{P}, 'a)\ VA$
$\boldsymbol{\Xi}$	$CR : (((((BOOL, 'b)\ VA, 'b)\ VA, (BOOL, 'a)\ VA)\ VA, ('b\ \mathbb{P}, 'a)\ VA)\ VA, ('b, 'a)\ VA)\ VA$
$\boldsymbol{\Theta}$	FR

$: ((\mathbb{N}, 'a)\ TREEE\ \mathbb{P},$
$((BOOL, (\mathbb{N}, 'a)\ TREEE)\ VA, (\mathbb{N}, 'a)\ TREEE)\ VA\ LIST$
$LIST)\ VA$

7.4 Definitions

FofCSet $\vdash FofCSet$
$= \{(ppmss,\ ppms1)$
$| ppmss = \{\}$
$\wedge (\exists\ s1\ s2$
$\bullet\ ppms1 = s1 =_p s2 \vee ppms1 = s1 \in_p s2)$
$\vee (\exists\ p\ s$
$\bullet\ ppmss = \{p\} \wedge ppms1 = \exists_p\ s\ p$
$\vee ppms1 = \neg_p\ p)$
$\vee (\exists\ p1\ p2$
$\bullet\ ppmss = \{p1;\ p2\} \wedge ppms1 = p1 \wedge_p p2)\}$

Fof $\vdash Fof = HeredFun\ (Rules2Fun\ FofCSet)$

FofCProp $\vdash \forall\ ppmss$
$\bullet\ FofCProp\ ppmss$
$\Leftrightarrow (\forall\ s1\ s2$
$\bullet\ s1 =_p s2 \in ppmss \wedge s1 \in_p s2 \in ppmss)$
$\wedge (\forall\ p$
$\bullet\ p \in ppmss$
$\Rightarrow \neg_p\ p \in ppmss \wedge (\forall\ s \bullet \exists_p\ s\ p \in ppmss))$
$\wedge (\forall\ p1\ p2$
$\bullet\ p1 \in ppmss \wedge p2 \in ppmss \Rightarrow p1 \wedge_p p2 \in ppmss)$

Fof₂ $\vdash Fof_2 = \bigcap\ \{s | FofCProp\ s\}$

7.5 Theorems

fof_induction_thm
$\vdash \forall\ s \bullet\ FofCProp\ s \Rightarrow Fof_2 \subseteq s$

fof_induction_thm2
$\vdash \forall\ s$
$\bullet\ (\forall\ s1\ s2 \bullet\ s1 =_p s2 \in s \wedge s1 \in_p s2 \in s)$
$\wedge (\forall\ p$
$\bullet\ p \in s \Rightarrow \neg_p\ p \in s \wedge (\forall\ s' \bullet \exists_p\ s'\ p \in s))$
$\wedge (\forall\ p1\ p2 \bullet\ p1 \in s \wedge p2 \in s \Rightarrow p1 \wedge_p p2 \in s)$
$\Rightarrow Fof_2 \subseteq s$

8 INDEX