The Theory of PolySets and its Applications

Roger Bishop Jones

Date: 2006/12/11 13:56:10

Abstract

A model for a set theory with a universal set and its use in the construction of other interesting and possibly useful foundational ontologies.

http://www.rbjones.com/rbjpub/isar/HOL/PolySets.pdf
http://www.rbjones.com/rbjpub/isar/HOL/PolySets.tgz

Id: root.tex,v 1.4 2006/12/11 13:56:10 rbj01 Exp

Contents

1 Introduction

The purpose of thie document is to construct a non-well-founded domain of sets with a universal set. It is hoped that the particular construction employed will yield a collection of sets suitable for use in further constructions yielding domains with elements with slightly more common structure than sets (for example functions or functors). These domains will then be used as the underlying ontology for a foundation system for formalised mathematics, and more generally for the formal demonstration of analytic truths. Some of these avenues of further development will be explored.

The special feature of the proposed ontologies by contrast with other already established systems is their anticipated suitability for foundational systems intended for formalisation "in the large" i.e. which are designed for modular specification and proof, maximising re-usability. This is the terminology of software engineering, and is used because of the relevant similarities which can be seen, between large scale formalisation of mathematics and (other analytic domains), and software development.

These non-well-founded foundation systems are not intended to make a difference to the way in which ordinary mathematics is done. By 'ordinary' mathematics, I mean arithmetic, analysis, geometry and any other mathematical study in which the domain of discourse is some definite concrete mathematical structure (often a system of numbers). Nor is it intended to provide an alternative treatment of cardinal arithmetic, which would continue to be undertaken via the Von Neumann ordinals.

The areas it would impact more substantially are those parts of mathematics which are in some way generic, where a theory is developed without any definite conception of its domain of discourse, expecting the theory to be applicable in many different domains. The most obvious examples of this kind of mathematics are abstract algebra, universal algebra and category theory.

1.1 Some History

The problem addressed is a long standing problem going back to the beginnings of mathematical logic. It may be helpful to describe briefly some of the historical connections,

Our story begins with the discovery by Russell of the incoherence of Frege's logical system. This discovery made necessary Russell's work leading to his "Theory of Types" in which draconian restrictions were placed through the type theory on what entities could be obtained by abstraction. It was immediately evident that the restrictions placed were prohibitive and would render the formal logicisation of mathematics unrealisable using the Theory of Types. Two separate problems were perceived and resolved in Principia

Mathematica. The first which comes from the "ramifications" in the Type Theory which made it predicative, was resolved by Russell's adoption of an axiom which made it as if the ramifications had never been included, and lead to the ramifications being omitted from most subsequent type theories. This does not here concern is.

The second problem was that many widely used mathematical entities appeared no longer as single objects but as complete families of objects with a variety of different types. Strict formalisation of mathematics would require that theorems about these objects be proved separately for each type with which they are used, even thought the proofs would all, apart from type annotations, be identical. To deal with this Russell adopted the device of typical ambiguity, an informal understanding that type annotations could be omitted in a proof and the proof then suffice for all results for which it is type-correct.

This essential logical scheme travelled down through the 20th century arriving in about 1985 in the logic HOL devised and implemented in Computer Software for use in the formal verification of digital hardware. This variant of what was by then known as "Higher Order Logic" remained logically similar to Russell's theory of types (without the ramifications). The two main innovations, of which the impact was pragmatic rather than fundamental, were the translation into a form based on the lambda-calculus and the transfer of the type variables with which Russell's pragmatic notion of "typical ambiguity" were made explicit, from the metalanguage into the object language. The lambda-calclulus, and the variety of Higher Order Logic based on it (known as the Simple Theory of Types), are both due to Church [2, 3].

1.2 Notation and Terminology

It is in the nature of the subject matter that many different variants of familiar mathematical concepts are used. In particular, there are a variety of different kinds of set membership, and also various kinds of predication and type assignment. Not only do we have a confusing profusion of similar but not identical concpts, but we are constrained by the fact that this work is mathematics formalised for processing by computer, and must therefore fit in with the constraints imposed by the software (in this case Isabelle).

Overloading the membership sign, would I fear lead to an unintelligible document, but chosing another symbol each time a new relationship is introduced would not be much better. Subscripts and superscripts are not much needed for their cusomary purposes and are therefore used exclusively as decorations for symbols which distinguish the variants employed.

I have tried to make their use as systematic as possible, a different letter is associated with each domain and is used as a subscript on operators over that domain.

The letters are as follows:

letter	type	domain	description
g	Set	UNIV	a domain of pure well-founded sets
r	Set	polysets	intermediate representatives for the polysets
q	Set set	ps_eqc	final representatives for polysets
p	pset	UNIV	the polysets themselves (as a new type)

2 Miscellanea

theory *PsMisc*
imports *Main*
begin

The theory *PsMisc* [1] is the home for material required for the development of the theory of PolySets and its applications which perhaps should be elsewhere but as yet is not.

When a better home is found this material will be moved out.

Your best plan is to skip this section and refer back to it as needed,

2.1 FixedPoint Theory

Both inductive definitions of sets and recursive definitions of functions are used in defining the concept of PolySet and it is then necessary to prove properties of the defined sets.

Though Isabelle-HOL provides support for both these kinds of definition, it does not fully meet the needs of this application. For example the Isabelle-HOL inductive set definition facility does prove an induction principle, but this principle is oriented toward proving common properties of the elements of the set, rather than properties of the set as a whole. Also, the support for recursive functions is oriented toward the definition of total functions, which many of the functions we consider are not.

The following material was begun when I was having difficulty with an inductive set definition, and thought I needed a stronger induction principle. Before solving that problem I changed the definition so that I no longer required the stronger principle and so this is at present on hold. However, there are other areas where I will probably have to do more here. The new definition may prove awkward to pull the necessary properties over when I get further along, since I made the inductive set definition more complex in order to easy the proof that it gives and equivalence relation (see definition of *ps_equiv* in theory PolySetsC).

There are also at present in the definition of PolySets places where a definition of a function which most naturally would be by recursion has been done as an inductive definition of the set of ordered pairs which are its graph. It may be better to prove a better version of the recursion theorem so that I do it as a recursive function definition (or perhaps just figure out how to do it with the available recursion theorem).

Anyway, what I have here is at present completely useless, but I expect eventually to have to come back and do something more useful.

[1] Id: PsMisc.thy,v 1.1 2006/11/28 16:50:49 rbj01 Exp

2.1.1 Chain Induction

There is in the theory *FixedPoint* on which inductive definitions of sets is based, one induction principle which proves properties of the defined set as a whole.

lfp_ordinal_induct:

$$[\![mono\ f;\ \bigwedge S.\ P\ S \Longrightarrow P\ (f\ S);\ \bigwedge M.\ \forall S{\in}M.\ P\ S \Longrightarrow P\ (\bigcup M)]\!] \Longrightarrow P\ (lfp\ f)$$

This principle requires that the desired property is continuous across arbitrary unions. This not true of the property of being an equivalence relation, though that property is continuous across unions of upward chains of sets, i.e. sets linearly ordered by the subset relation. Strictly speaking one ought not to need any more than continuity over upward "f-chains" where f is the functional whose least fixed point is the inductively defined set and an f chain is obtained by repeated application of f.

Here we have only a skirmish in obtaining such an induction principle. At the moment I seem to be going round in circles.

constdefs
  chain :: *'a set set ⇒ bool*
    chain s == $\forall t\ u.\ t \in s \wedge u \in s \longrightarrow t \subseteq u \vee u \subseteq t$
  p-chain :: (*'a set ⇒ bool*) ⇒ *'a set set ⇒ bool*
    p-chain p s == *chain s* ∧ ($\forall t.\ t \in s \longrightarrow p\ t$)

lemma *f-chain-mono* : $A \subseteq B \Longrightarrow S \subseteq A \longrightarrow S \subseteq B$
⟨*proof*⟩

consts *f-chain* :: (*'a set ⇒ 'a set*) ⇒ *'a set set*
inductive
  f-chain f
intros
  fczI: {}:*f-chain f*
  fcsI: *S*:*f-chain f* ⟹ *f S* : *f-chain f*
  fclI: $S \subseteq$ (*f-chain f*) ⟹ $\bigcup S$: *f-chain f*
monos *f-chain-mono*

lemma *chain-f-chain* : *chain* (*f-chain f*)
⟨*proof*⟩
thm *f-chain.induct*
⟨*proof*⟩

On the face of it it looks like you need the induction principle I am trying to prove to get this result.

Here is the statement of the induction principle sought.

constdefs
  pf-chain :: (*'a set ⇒ bool*) ⇒ (*'a set ⇒ 'a set*) ⇒ *bool*

$$pf\text{-}chain\ P\ f == (\forall S.\ S \subseteq (f\text{-}chain\ f) \wedge (\forall s.\ s \in S \longrightarrow P\ s) \longrightarrow P(\bigcup S))$$

lemma *lfp-ordinal-induct2*:
assumes *mono*: *mono f*
shows [| !!S. $P\ S$ ==> $P(f\ S)$;
pf-chain P f |]
==> $P(lfp\ f)$
⟨*proof*⟩

2.2 Set Theory

lemma *inter-mono*:
$A \subseteq B \Longrightarrow A \cap C \subseteq B \cap C$
⟨*proof*⟩

2.3 Cartesian Product

lemma *rsubdomcran*:
$r \subseteq (Domain\ r \times Range\ r)$
⟨*proof*⟩

lemma *domsub*:
$r \subseteq (A \times B) \Longrightarrow Domain\ r \subseteq A \wedge Range\ r \subseteq B$
⟨*proof*⟩

lemma *cp-mono*:
$A \subseteq B \Longrightarrow (A \times C) \subseteq (B \times C) \wedge (C \times A) \subseteq (C \times B)$
⟨*proof*⟩

2.4 Transitive Closure

lemma *trancl-ss*:
$A \subseteq (trancl\ A)$
⟨*proof*⟩

lemma *tranclsub*:
$trancl\ r \subseteq (Domain\ r \times Range\ r)$
⟨*proof*⟩

lemma *rsubtranclr*:
$r \subseteq (trancl\ r)$
⟨*proof*⟩

lemma *trancl-ss*:
$r \subseteq (A \times A) \Longrightarrow (trancl\ r) \subseteq (A \times A)$
⟨*proof*⟩

lemma *refl-trancl*:
$refl\ C\ r \Longrightarrow refl\ C\ (trancl\ r)$
⟨*proof*⟩

lemma *trancl-mono*:
  $!!p.\ p \in r\hat{}+ \Longrightarrow r \subseteq\ \ s \Longrightarrow p \in s\hat{}+$
 ⟨*proof*⟩

lemma *sym-trancl*:
  $sym\ e \Longrightarrow sym\ (trancl\ e)$
⟨*proof*⟩
thm *trancl.induct*
⟨*proof*⟩

end

3 Pure Well-Founded Sets

theory *Sets*
imports *Main*
begin

The theory *Sets*[2] is a bare-boned set theory in Isabelle-HOL intended only to permit a transparent presentation of the Poly-Sets. I did at first attempt the construction without axiomatising a particular set theory, so that the poly-sets could be built from an arbitrary membership structure with suitable properties, but the advantages of this are overwhelmed by the extra complexity it causes, and I have concluded that maximising the intelligibility of the poly-sets (and of further constructions based on them) is incompatible with the innovation of treating set theory as a theory *about* membership structures, rather than a theory about the sets in *in* one such structure,

The reader should beware that what follows is an axiomatic set theory presented in the context of a Higher Order Logic which has its own set theoretic vocabulary. It is convenient to make use of that vocabulary, and so two notions of set and membership should be distinguished. The extant set theory is that of elements of type *set* and uses the usual set membership predicate $\in$. All of the standard set theoretic symbols belong to this theory. The new set theory is axiomatised in type *Set* and uses the (not very nice) operator `\in_g` for membership.

For the sake of keeping down culture shock the axioms are theorems rather than rules. Working in Isabelle higher-order rules (which provide something like a sequent calculus) are more convenient and will be derived as needed.

3.1 Extensionality and Well-Foundedness

Extensionality and Well-Foundedness express constraints on the notion of set. They constrain the sets to be the pure well-founded sets.

Well foundedness is expressed as an induction principle. This is stronger than the first order axiom of regularity, though in the context of higher order replacement (which appears later) the difference disappears.

As it happens, we may not need it anyway, since the poly-set construction which follows naturally extracts a subset of the well-founded sets, leaving behind any non-well-founded sets.

Nor is it certain that we need an extensional set theory as our starting point. The construction yields in the first instance a non-extensional model, which is made extensional by taking a quotient, so it might well work if it is non-extensional all the way up to that final oeration. Since the question of how weak our starting point might possibly have been is not of much wieght, it

[2] Id: Sets.thy,v 1.3 2006/12/11 12:14:52 rbj01 Exp

is preferred to keep things simple by starting from a cery standard model of the pure-well-founded sets.

typedecl *Set*

consts *mem* :: *Set* ⇒ *Set* ⇒ *bool* (**infix** $\in_g$ *80*)

axioms

Ext: $\forall\ x\ (y{::}Set).\ (x = y) = (\forall\, z{::}Set.\ z \in_g x = z \in_g y)$

Wf: $\forall\, P.\ (\forall\, s.\ (\forall\, t.\ t \in_g s \longrightarrow P\ t) \longrightarrow P\ s) \longrightarrow (\forall\, u.\ P\ u)$

The axiom *Wf* is related to the definition of the predicate *wf* in theory *WellFounded_recursion*. The latter is a property of relations while *Wf* is simply a sentence (which distinguishes the set of relations which satisfy it), The following lemma connects the property with the axiom, and will be used to justify recursive definitions.

lemma *wf*: *wf* $\{(x,y).\ x \in_g y\}$
⟨*proof*⟩

3.2 Closure

Having tied down what kind of thing a set is, the final axiom tells us that there are many of them. It says that every set is a member of a "galaxy" (*Gy*) which is the kind of thing which most people call a universe. The empty set and the hereditarily finite sets are galaxies, and the next one is a model of ZFC. (Note that we inherit choice from our context) Galaxies are closed under full power-set, sumset and replacement. The universe is also closed under replacement (you don't have to show that the image of a set is a subset of a galaxy for it to be a set, though it won't otherwise be a member of the galaxy).

I hope that it makes things easier to read if the available set theoretic vocabulary is used, and to that end define a constant X_g which gives the extension of a *Set* as a *set* of Sets.

constdefs

X_g :: *Set* ⇒ *Set set*
$X_g\ x == \{y\ .\ y \in_g x\}$

Gy :: *Set* ⇒ *bool*
$Gy\ g == (\forall x.\ x \in X_g\ g \longrightarrow$
$(\exists y.\ y \in X_g\ g \wedge X_g\ y = \bigcup\{z.\ \exists v.\ v \in_g x \wedge z = X_g\ v\})$
$\wedge\ (\forall x.\ x \in X_g\ g \longrightarrow (\exists y.\ y \in X_g\ g \wedge X_g\ y = \{z\ .\ X_g\ z \subseteq X_g\ x\}))$
$\wedge\ (\forall\, r{::}(Set * Set)set.\ single\text{-}valued\ r$
$\longrightarrow (\exists\, y.\ X_g\ y = r\ ``\ (X_g\ x) \wedge ((X_g\ y) \subseteq (X_g\ g) \longrightarrow y \in X_g\ g))))$

lemma *ext-xt* [*simp*]: $X_g\ x = X_g\ y \Longrightarrow x = y$
⟨*proof*⟩

Having thus defined a galaxy as a set closed under distributed union, power set and replacement, the final axiom states that every set is a member of a galaxy.

The existence of glaxies will I hope be helpful in overcoming awkwarndesses which might arise in working with the well-founded poly-sets from the presence of non-well-founded poly-sets. Though the domain as a whole is not much like a model of ZFC, we have pleanty of things around which are like that.

axioms

G: $\forall\, s.\ \exists\, t.\ s \in_g t \wedge Gy\ t$

These two definitions may be useful in defining operators or in expressing closure properties. The first expresses the existence of a set with some extension, the second designates the set with some extension.

constdefs

$E_g :: Set\ set \Rightarrow bool$
$E_g\ ss == \exists\, s.\ X_g\ s = ss$

$C_g :: Set\ set \Rightarrow Set$
$C_g\ s == (THE\ x.\ X_g\ x = s)$

$XX_g :: Set \Rightarrow Set\ set\ set$
$XX_g\ s == \{x.\ \exists\, y.\ y \in_g s \wedge x = X_g\ y\}$

lemma C_gX_g [*simp*]:
$\forall\, s.\ C_g(X_g\ s) = s$
⟨*proof*⟩

lemma E_gC_g [*simp*]:
$E_g\ s \Longrightarrow X_g\ (C_g\ s) = s$
⟨*proof*⟩

3.3 Pairs

The constructor for Wiener-Kuratowski ordered pairs is defined here, It remains to be established whether pairs always exist.

constdefs

$Wkp :: (Set * Set) \Rightarrow Set$
$Wkp == \lambda(x,y).\ C_g\{C_g\{x\},C_g\{x,y\}\}$

$Fst :: Set \Rightarrow Set$
$Fst\ s == THE\ x.\ \exists\, y.\ s = Wkp(x,y)$

$Snd :: Set \Rightarrow Set$
$Snd\ s == THE\ y.\ \exists\, x.\ s = Wkp(x,y)$

3.4 Ordinals

The ordinals are defined in Jech [5] as those sets which are transitive and well-ordered by membership. However, I think I might get more milage from a recursive definition of ordinals so that's what I'll do.

First I define zero, the sucessor relationship, the limit of a set of ordinals and the predecessor of a successor ordinal. The last two are just set union (notionally restricted to appropriate sets of ordinals).

constdefs
$zero :: Set$
$zero == C_g\ \{\}$

$succ :: Set \Rightarrow Set$
$succ\ s == C_g\ (X_g\ s \cup \{s\})$

$SetUnion :: Set \Rightarrow Set$
$SetUnion\ s == C_g\ (\bigcup \{x.\ \exists\, y.\ y \in_g s \wedge x = X_g\ y\})$

$limit :: Set \Rightarrow Set$
$limit == SetUnion$

$pred :: Set \Rightarrow Set$
$pred == SetUnion$

The ordinals are now defined inductively. This probably isn't the simplest definition of ordinal, but it might be the one most directly related to our intuition about what a Von Neumann ordinal is.

consts
$ordinals :: Set\ set$
inductive *ordinals*
intros
$zoI[intro!]: zero \in ordinals$
$soI[intro!]: x \in ordinals \Longrightarrow succ\ x \in ordinals$
$loI[intro!]: \forall\, t.\ t \in_g s \longrightarrow t \in ordinals \Longrightarrow limit\ s \in ordinals$

constdefs
$Ordinal :: Set \Rightarrow bool$
$Ordinal\ s == s \in ordinals$

We will need to subtract ordinals on the left. Addition is defined first. Since the ordinals are not a type this is more awkward that it would otherwise be, I couldn't get the definition by recursion through isabelle so I ended up using an inductive set definition.

lemma *subset-mono*:
$A \subseteq B \Longrightarrow\ x \subseteq A \longrightarrow x \subseteq B$
⟨*proof*⟩

consts
oplusx :: $Set \Rightarrow (Set * Set)\ set$
inductive
oplusx x
intros
azI: $y = zero \Longrightarrow (y,x) \in (oplusx\ x)$
asI: $y = succ\ v \Longrightarrow (v,z) \in oplusx\ x \Longrightarrow (y,\ succ\ z) \in oplusx\ x$
alI: $y = limit\ s$
$\Longrightarrow (\exists\, r.\ r \subseteq oplusx\ x$
$\wedge\ single\text{-}valued\ r$
$\wedge\ Domain\ r = X_g\ s$
$\wedge\ z = limit\ (C_g\ (Range\ r)))$
$\Longrightarrow (y,z) \in oplusx\ x$
monos *subset-mono*

constdefs
oplus :: $Set \Rightarrow Set \Rightarrow Set$ (**infixl** $\{+\}$ *60*)
$oplus\ x\ y == THE\ z.\ (y,z) \in (oplusx\ x)$

olminus :: $Set \Rightarrow Set \Rightarrow Set$ (**infixr** $\{|-\}$ *60*)
$olminus\ x\ y == THE\ z.\ y = x\ \{+\}\ z$

end

4 The Construction of PolySets

theory *PolySetsC*
imports *Sets PsMisc Transitive-Closure*
begin

The theory *PolySetsC* [3] first provides a construction of a model of a set theory with a universal set (a membership relation). This is constructed in the context of the higher order axiomatisation of set theory in *Sets.thy* and uses (well-founded) sets in the domain of that set theory to represent the non-well-founded sets in the domain of the new model. A membership relation over these new sets is then defined and we then proceed to establish properties of the model suitable for use in a higher order axiomatisation of a set theory with a universal set. We call these sets *PolySet*s because the kind of non-well-founded set concerned is inpired by the polymorphism in functional programming languages such as ML, and is intended to factilitate the importation into foundations systems suitable for use in the formalisation of mathematics and science improved polymorphism, and local and global structuring features.

4.1 Well-Founded Poly-Sets and Poly-Set Ordinals

We need to define a representation of ordinals as PolySets first, because these ordinals will be used in defining the non-well-founded PolySets. It turns out easiest to define the well-founded sets as a whole and then pick out the Von Neumann ordinals.

There is an injection from the well-founded sets in *Sets* into the PolySets. The set we are about to define is the image under that injection of the Von Neumann ordinals. The injection is defined first.

consts *wf-polyset* :: *Set* ⇒ *Set*
recdef *wf-polyset* {(*x*,*y*). *x* ∈$_g$ *y*}
  wf-polyset s =
    (*if* (*s* = *zero*)
     then zero
     else (*Wkp*(*zero*, *C*$_g$ (*image wf-polyset* {*x* . *x* ∈$_g$ *s*}))))
(**hints** *recdef-wf*: *wf*)

The definitions of the well-founded and ordinal polysets are then:

constdefs
  wf-polysets :: *Set set*
    wf-polysets == *image wf-polyset UNIV*

  polyset-ords :: *Set set*
    polyset-ords == *image wf-polyset ordinals*

[3] Id: PolySetsC.thy,v 1.3 2007/05/22 09:32:28 rbj01 Exp

Not sure whether these are of any value.

constdefs
mem-restr :: *Set set* ⇒ (*Set* ∗ *Set*)*set*
mem-restr s == {(*x*,*y*) . *x* ∈ *s* ∧ *y* ∈ *s* ∧ *x* ∈$_g$ *y*}

The operation of subtraction on the left for ordinals (from *Sets.thy*) must now be transferred to operate on PolySet ordinals.

constdefs
ps-olminus :: *Set* ⇒ *Set* ⇒ *Set* (**infixr** {‖−} *60*)
ps-olminus x y == (*THE z*. ∃ *v w*. *x* = *wf-polyset v*
∧ *y* = *wf-polyset w*
∧ *z* = *wf-polyset* (*v* {|−} *w*))

4.2 PolySets

The PolySets are now definable inductively.

A PolySet is an ordered pair of which the left hand side is a PolySet ordinal and the right is a set of PolySets. This is a recursive definition, to which we add the clause "and nothing else" to get the least fixed point. This is expressed in Isabelle-HOL as a inductive set definition.

consts
polysets :: *Set set*
inductive *polysets*
intros
zPsI[*intro*!]: *zero* : *polysets*
PsI[*intro*!]: (∀ *ord s*. *ord* ∈ *polyset-ords*
∧ (∀ *w*. *w* ∈$_g$ *s* ⟶ *s* : *polysets*))
⟹ *Wkp*(*ords*, *s*) ∈ *polysets*

4.3 Pattern Instantiation

The semantics of PolySets (i.e. the definition of the membership relationship over polysets, which determines which set of PolySets each PolySet denotes), depend on making precise the idea that a PolySet is a collection of patterns.

This is made precise by defining the collection of PolySets which is represented by such a pattern. The elements of this collection are the *instances* of the pattern, and the particular instance is determined by a valuation for the pattern variables. Instead of using named variables, we use the ordinals for pattern variables, and these behave in the same manner as De Bruijn indices [1].

The PolySets are patterns, in which the ordinals are instantiable pattern variables. The key element in defining the membership relation over the PolySets is the specification of when a PolySet conforms to a pattern. It does do of course, when the pattern can be instantiated according to some assignment of values to the pattern variables, to give the required PolySet.

I therefore begin with the definition of this process of instantiation, after which the definition of membership will be straightforward,

Pattern instantiation is defined by transfinite recursion over the PolySets. The well-founded relation with respect to which the recursion is well-founded is not the intended membership relation over the PolySets, but the membership relation from the theory *Sets*.

The function we require here is total over the intended domains, which are subsets of unspecified types. Our logic is a logic of total functions, and so we have to define instantiation not as a function but as a relation (which will be many one over the target domain).

A PolySet valuation for an ordinal in the PolySets is a (HOL, total) function which maps the PolySet ordinals less than the given ordinal to PolySets. It doesn't matter what it does elsewhere.

constdefs
  $wfps\text{-}mem :: Set \Rightarrow Set \Rightarrow bool$ (**infix** $\in_r$ 70)
    $wfps\text{-}mem\ o1\ o2 == o1 \in_g Snd\ o2$

  $is\text{-}ps\text{-}val :: Set * (Set \Rightarrow Set) \Rightarrow bool$
    $is\text{-}ps\text{-}val == \lambda\ (ord, va).\ ord \in polyset\text{-}ords$
                $\wedge\ (\forall\ o2.\ o2 \in_r ord \longrightarrow (va\ o2) \in polysets)$

consts
  $ps\text{-}instantiate :: Set \Rightarrow (Set \Rightarrow Set) \Rightarrow (Set * Set)set$
inductive
  $ps\text{-}instantiate\ ord\ va$
intros

  $vPsInI : o2 \in_r ord$
        $\Longrightarrow (o2, va\ o2) : ps\text{-}instantiate\ ord\ va$

  $oPsInI : o1 \in polyset\text{-}ords$
        $\Longrightarrow \neg\ (wfps\text{-}mem\ o1\ ord)$
        $\Longrightarrow (o1, ord\ \{\|-\}\ o1) : ps\text{-}instantiate\ ord\ va$

  $sPsInI : \neg\ s \in polyset\text{-}ords \wedge t \in polysets$
         $\wedge\ ord2 = Fst\ s \wedge ord2 = Fst\ t$
         $\wedge\ (\exists\, r.\ r \subseteq ps\text{-}instantiate\ ord\ va$
           $\wedge\ single\text{-}valued\ r$
           $\wedge\ Domain\ r = X_g(Snd\ s)$
           $\wedge\ t = C_g\ (Range\ r))$
         $\Longrightarrow (s,t)$: $ps\text{-}instantiate\ ord\ va$

monos *subset-mono*

constdefs
  $ps\text{-}inst :: Set * (Set \Rightarrow Set) \Rightarrow Set \Rightarrow Set$

ps-inst == λ(*ord*, *va*) *ps*. *THE x*. (*ps*,*x*) ∈ *ps-instantiate ord va*

4.4 Membership

Membership is now defined as follows. Nothing is a member of the empty set. If the left element of a PolySet is the empty set, then there are no free variables in the right hand side, and the members of the PolySet are the members of the set on the right. If the left element is not the empty set then it will be a PolySet ordinal which is the set of free variables which may occur in patterns on the right. Each member of the set on the right is a pattern and a set is a member of the PolySet if there is a valuation for the free variables and an element (pattern) of the set on the right such that the set results when the pattern is instantiated using the valuation.

constdefs

```
ps-mem :: Set ⇒ Set ⇒ bool
  ps-mem s t == s ∈ polysets ∧ t ∈ polysets
                ∧ (if t = zero
                   then False
                   else (if Fst t = zero
                         then s ∈g Snd t
                         else (∃ va u. is-ps-val (Fst t, va)
                              ∧ (u,s) ∈ ps-instantiate (Fst t) va )))
```

4.5 Extensionality

The membership structure consisting of the relation *ps_mem* over the set *polysets* is not extensional. For example, no polyset with rhs the empty set has any members, but there are as many such polysets as the Von Neumann ordinals in type *Set*. Also, for every polyset with a non-empty rhs, there are extensionally equivalent polysets obtained by permuting the names (numbers really) of the free variables, or by adding extra unused variables.

We can recover extensionality by taking a quotient with resepect to the smallest equivalence relattionship relative to which the induced membership relationship is extensional. There must be one such quotient, since taking all polysets as equivalent yields a trivially extensional relationship. That there is a smallest such relationship needs to be proven.

There are two obvious ways of defining the relationship formally. The first is to take the intersection of the equivalence classes which yield an extensional quotient. This form of definition probably makes it easy to show that the result is extensionsal, but not so easy for other kinds of proof, where the property required depends on things not being identified unless they really are extensionally equal.

The other method is an inductive definition. Isabelle provide support for inductive definitions of sets based on collections introduction rules for indi-

vidual members of the relation. This yeilds an induction principle, but this is convenient primarily for estblishing properties of members of the recursively defined set. We are more interested in establishing properties of the set as a whole, in the first instance simply that it is an equivalence relationship. This is not so easy to prove in this way, because the individual rules do not preserve this property. An alternative way of defining the relationship inductively is to ignore the facility for defining inductive sets, and use the underlying fixpoint theory with an operator which preserves key properties, for example that of being an equivalence relationship.

It is convenient to define first the extension (as a *Set set*) of a polyset.

constdefs
$X_r :: Set \Rightarrow Set\ set$
$X_r\ t == \{s.\ ps\text{-}mem\ s\ t\}$

The following definitions give an operator on relations whose least fixed point will be the least equivalence relation for which the lifted membership relation is extensional. Care must be taken to ensure not only that this operator is monotonic, but also that it can be seen (proven indeed) to be monotonic without any assumptions being made about the relation on which it operates.

To make it obvious that the following constructor is monotonic, the reference to the inductive set is packaged up as follows. The relationship defined is that the extension of a set is a subset of the extension of some other set, taking into account the equivalence relationahip, i.e. take the image of the second set under the equivalence relationship and see if that contains the first set. This is clearly monotonic in the equivalence relationship.

constdefs
$extsube :: (Set \times Set)set \Rightarrow (Set \times Set)set$
$extsube\ e == \{(s,t).\ s \in polysets \wedge t \in polysets$
$\wedge\ (\forall u.\ ps\text{-}mem\ u\ s \longrightarrow ps\text{-}mem\ u\ t \vee (\exists v.\ (u, v) \in e \wedge ps\text{-}mem\ v\ t))\}$

$exteqe :: (Set \times Set)set \Rightarrow (Set \times Set)set$
$exteqe\ e == ((extsube\ e \cap converse\ (extsube\ e)) \cup (e \cap converse\ e))$
$\cap\ polysets \times polysets$

$tcexteq :: (Set \times Set)set \Rightarrow (Set \times Set)set$
$tcexteq\ e == trancl(exteqe\ e)$

lemma *extsube-mono*:
$A \subseteq B \Longrightarrow extsube\ A \subseteq extsube\ B$
⟨*proof*⟩

lemma *exteqe-mono*:
$A \subseteq B \Longrightarrow exteqe\ A \subseteq exteqe\ B$
⟨*proof*⟩

lemma *extsube-polysets*:
$(x,y) \in extsube\ e \Longrightarrow x \in polysets \wedge y \in polysets$
⟨*proof*⟩

lemma *extsube-refl*:
$refl\ polysets\ (extsube\ e)$
⟨*proof*⟩

lemma *extsube-trans*:
$trans\ e \Longrightarrow trans\ (extsube\ e)$
⟨*proof*⟩

lemma *exteqe-polysets*:
$exteqe\ e \subseteq polysets \times polysets$
⟨*proof*⟩

lemma *exteqe-refl*:
$refl\ polysets\ (exteqe\ e)$
⟨*proof*⟩

lemma *exteqe-sym*:
$sym\ (exteqe\ e)$
⟨*proof*⟩

lemma *tcexteq-mono*:
$A \subseteq B \Longrightarrow tcexteq\ A \subseteq tcexteq\ B$
⟨*proof*⟩

lemma *tcexteq-refl*:
$refl\ polysets\ (tcexteq\ e)$
⟨*proof*⟩

lemma *tcexteq-sym*:
$sym\ (tcexteq\ e)$
⟨*proof*⟩

lemma *tcexteq-trans*:
$trans\ (tcexteq\ e)$
⟨*proof*⟩

Now the equivalence relation.

consts
$ps\text{-}equiv :: (Set * Set)set$
inductive
ps-equiv
intros
pseI: $(s,t) \in tcexteq\ ps\text{-}equiv \Longrightarrow (s,t) \in ps\text{-}equiv$
monos *tcexteq-mono*

lemma *ps-equiv-l1*:
$(t,s) \in tcexteq\ ps\text{-}equiv \Longrightarrow (t,s) \in ps\text{-}equiv$
⟨*proof*⟩

lemma *ps-equiv-l2*:
$(t,s) \in ps\text{-}equiv \Longrightarrow (t,s) \in tcexteq\ ps\text{-}equiv$
⟨*proof*⟩

lemma *ps-equiv-l3*:
$tcexteq\ ps\text{-}equiv = ps\text{-}equiv$
⟨*proof*⟩

lemma *ps-equiv-l3b*:
$ps\text{-}equiv = tcexteq\ ps\text{-}equiv$
⟨*proof*⟩

lemma *ps-equiv-l4*:
$(s,t) \in ps\text{-}equiv \Longrightarrow s \in polysets \wedge t \in polysets$
⟨*proof*⟩

lemma *ps-equiv-refl*:
$refl\ polysets\ ps\text{-}equiv$
⟨*proof*⟩

lemma *ps-equiv-sym*:
$sym\ ps\text{-}equiv$
⟨*proof*⟩

lemma *ps-equiv-trans*:
$trans\ ps\text{-}equiv$
⟨*proof*⟩

lemma *ps-equiv-equiv*:
$equiv\ polysets\ ps\text{-}equiv$
⟨*proof*⟩

Now that we have the equivalence relation we can take a quotient and lift the membership relationship up to the quotient classes. The membership structure $(ps\_eqc, pseqc\_mem)$ will provide the representation for the new type of PolySets.

constdefs
$ps\text{-}eqc :: Set\ set\ set$
$ps\text{-}eqc == polysets\ //\ ps\text{-}equiv$

$pseqc\text{-}mem :: Set\ set \Rightarrow Set\ set \Rightarrow bool$
$pseqc\text{-}mem\ s\ t == \exists v\ w.\ v \in s \wedge w \in t \wedge ps\text{-}mem\ v\ w$

$pseqc :: Set \Rightarrow Set\ set$

$pseqc\ s == ps\text{-}equiv\ ``\ \{s\}$

lemma *zero-ps-eqc*:
$pseqc\ zero \in ps\text{-}eqc$
⟨*proof*⟩

To prove that we now have an extensional membership relation it is convenient to define the properrt of being extensionally equal relative to an arbitrary membership relation. Then we can show that if in one relation two sets are extensionally equal, then they will be equal under the equivalence re

lemma *ps-mem-extensional*:
$\forall x\ y.\ (\forall z.\ ps\text{-}mem\ z\ x = psmem\ z\ y) \longrightarrow x = y$
⟨*proof*⟩

end

5 The Theory of PolySets

theory *PolySets*
imports *PolySetsC*
begin

The theory *PolySets* [4] introduces the new type *pset* (of *PolySets*) and develops the theory of PolySets.

5.1 The Type of PolySets

typedef *pset* = *ps-eqc*
⟨*proof*⟩

We need one binary relation defined in terms of the representation type, membership. In addition I think we probably need a predicate to separate the "mon"(-morphic) polysets from the rest. This is essentially the same as "Low" in *Church-Oswald* construction terminology.

constdefs
$psmem :: pset \Rightarrow pset \Rightarrow bool$ (**infix** $\in_p$ *80*)
$psmem\ e\ s == pseqc\text{-}mem\ (Rep\text{-}pset\ e)\ (Rep\text{-}pset\ s)$

$mon :: pset \Rightarrow bool$
$mon\ s == \exists r.\ Fst\ r = zero \wedge r \in (Rep\text{-}pset\ s)$

5.2 Axioms for PolySets

In this section theorems for PolySets are proven which might constitute a reasonable higher order axiomatisation. This is not the target application for PolySet theory, so if the axioms seem not very nice, that might not be a problem.

5.2.1 Axioms yet to be Demonstrated

The axioms are given as axioms, later, hopefully, to be replaced by theorems.

axioms
$Ext_p : \forall x\ y.\ (x = y) = (\forall z.\ z \in_p x = z \in_p y)$

constdefs
$X_p :: pset \Rightarrow pset\ set$
$X_p\ ps == \{s.\ s \in_p ps\}$

$pGy :: pset \Rightarrow bool$
$pGy\ g ==$
$(\forall x.\ x \in_p g \wedge mon\ x \longrightarrow$

[4] Id: PolySets.thy,v 1.2 2006/11/28 16:50:49 rbj01 Exp

$$
\begin{array}{l}
\quad (\exists\, y.\ y \in_p g \wedge X_p\ y = \bigcup \{z.\ \exists\, v.\ v \in_p x \wedge z = X_p\ v\}) \\
\wedge\ (\exists\, y.\ y \in_p g \wedge X_p\ y = \{z\ .\ X_p\ z \subseteq X_p\ x\}) \\
\wedge\ (\forall\, r{::}(pset * pset)set.\ single\text{-}valued\ r \\
\qquad \longrightarrow (\exists\, y.\ y \in_p g \wedge X_p\ y = r\ ``\ (X_p\ x))))
\end{array}
$$

axioms

G: $\forall\, ps.\ \exists\, g.\ pGy\ g \wedge ps \in_p g$

The smallest galaxy is the empty set, the existence of which is only an indirect consequence of this axiom.

The empty set is a member of any galaxy which contains a mon PolySet (by the powerset clause or the replacement clause in the definition of a galaxy). Since every galaxy is mon, and something exists (all types are non empty) there must be a mon set, and the galaxy containing that mon set will contain the empty set.

The galaxy containing the empty set is the set of hereditarily finite mon polysets. The galaxy containing that galaxy is a model of ZFC (noting that choice is inherited from HOL via the higher order formulation of replacement in a galaxy).

The smallest galaxy containing a poly PolySet is its unit set. The smallest galaxy containing a poly PolySet unit set is the hereditarily finite sets with that one PolySet as if it were a urelement.

lemma *inter-g-g*: $\forall\, ps\ sg.\ \forall\, g.\ g \in sg \wedge pGy\ g \wedge ps \in_p g$
$\qquad \longrightarrow pGy\ (C_p(\bigcap \{sgx.\ \exists\, s.\ sgx = X_p\ s \wedge s \in sg\}))$
⟨*proof*⟩

constdefs

$E_p :: pset\ set \Rightarrow bool$
$\quad E_p\ ss == \exists\, s.\ X_p\ s = ss$

$C_p{::}\ pset\ set \Rightarrow pset$
$\quad C_p\ s == (THE\ x.\ X_p\ x = s)$

$XX_p :: pset \Rightarrow pset\ set\ set$
$\quad XX_p\ s == \{x.\ \exists\, y.\ y \in_p s \wedge x = X_p\ y\}$

lemma C_pX_p [*simp*]: $\forall\, s.\ C_p(X_p\ s) = s$
⟨*proof*⟩

lemma E_pC_p [*simp*]: $E_p\ s \Longrightarrow X_p\ (C_p\ s) = s$
⟨*proof*⟩

5.3 Pairs

The constructor for Weiner-Kuratovski ordered pairs is defined here, It remains to be established whether pairs always exist.

constdefs

$Wkp_p :: (pset * pset) \Rightarrow pset$
$Wkp_p == \lambda(x,y).\ C_p\{C_p\{x\},C_p\{x,y\}\}$

$Fst_p :: pset \Rightarrow pset$
$Fst_p\ s == THE\ x.\ \exists\, y.\ s = Wkp_p(x,y)$

$Snd_p :: pset \Rightarrow pset$
$Snd_p\ s == THE\ y.\ \exists\, x.\ s = Wkp_p(x,y)$

end

6 An Illative Lambda Calculus

theory *Ilambda*
imports *PolySets*
begin

The theory *Ilambda* [5] is intended to provide an "illative lambda-calculus" based on the PolySets, i,e, in which the values of expressions are PolySets.

This is an exploratory theory, by which I mean that I don't really know how its going to work out.

6.1 Desiderata

The single most important thing I want to explore here is the conjecture that the functions definable as patterns include the polymorphic functions definable in a language similar to ML.

This question will be explored as a part of conisideration of how one might devise a logical system based on PolySets which is a compromise between the flexibility of the ML type system and the power of the HOL logic.

The main feature common to both these systems is that they are essentially functional rather than set theoretic, and therefore that the primary primitive constructor is the operation of function application rather than the predicate of set membership. A second feature which distinguishes these systems from PolySet *theory* is that they are neither of them first order theories. HOL is the transformation of the typed lambda-calculus into a logical system rather than the development of the theory of the lambda-calculus in some other logic, ML is the adoption of a congenial syntax for something which under the surface might be thought of as a lambda-calculus.

In a similar manner it is intended here to pave the way for leaving behind HOL and using the PolySet dressed in more functional syntax as a logical system in its own right.

Calling this an *illative* lambda-calculus refers back to the use of that term in combinatory logic. In that context, the *pure* combinatory logic is the calculus of functions based usually on the combinatory S,K and I but absent constants corresponding to logical operators. To turn this into a logical system suitable for use as a foundation for mathematics it is necessary to add one or more operators such as equality, universal quantification, or restricted quantification. When that is done it is called an *illative* combinatory logic.

One feature which illative combinatory logics and ML share but in which they differ from HOL is in the role of types. In the former two system types are properties of terms, a term may have many types, but the types are not constituents of or decorations on terms or their constituents. In HOL types

[5] Id: Ilambda.thy,v 1.1 2006/11/28 16:50:49 rbj01 Exp

are part of the syntax of terms and each term has a single type.

This distinction is important in understanding what happens in local definitions (which we will be concerned with) and of some ways of providing structure in the large (which are also of concern but will not here consider). In HOL a polymorphic object is semantically a family of monotypical objects. No function takes a polymorphic value, a polymorphic function is a family of functions each taking an argument whose type is some monotypical instance of the polymorphic type of its argument. This means that you cannot instantiate the types of the formal parameters in the body of a functional abstraction. If a let clause is explained in terms of a lambda-abstraction, then a polymorphic function defined by the let clause cannot be instantiated in the body of the clause,

6.2 Primitive Operators

Rather than looking at syntax, I begin by looking at the semantics of plausible primitive operations.

6.2.1 Application and Abstraction

The first things we need in a lambda-calculus are of course application and abstraction.

The only issue for abstraction is what value it should have if the operator is not a function with the operand in its domain. I would like, in the first instance to adopt a notion of application shich is defined provided only that the operator is single valued at the particular value denoted by the operand and whose value is otherwise undetermined (but is nevertheless some PolySet). In Isabelle-HOL I think I can only do this using the choice function:

Abstraction is defined similarity, via C_p, so that an abstraction denotes the appropriate function only if that function exists, which needs to be proven. Note that abstraction takes place always over some set declared to be the domain of the abstraction. This is a half way house between the typed abstract in HOL and the untyped abstraction in ML and combinatory logics. You can declare the domain as the universal set V (not yet defined) but we will see that there are severe constraints on the kind of functions over V which exist.

constdefs

$$PsApply :: pset \Rightarrow pset \Rightarrow pset \ (\textbf{infixl}\ ._p\ 70)$$

$$PsApply == SOME\ ap{::}pset \Rightarrow pset \Rightarrow pset.\ \forall\ f\ a\ r.\ ap\ f\ a = r \longrightarrow Wkp_p(a,r) \in_p f \vee \neg\ (\exists\ v.\ \forall\ w.\ Wkp_p(a,w) \in_p f = (w = v))$$

$$PsAbs :: pset \Rightarrow (pset \Rightarrow pset) \Rightarrow pset$$

$$PsAbs\ d\ f == C_p\ \{Wkp_p(a,r) \mid a\ r.\ r = f\ a \wedge a \in_p d\}$$

end

References

[1] N.G. De Bruijn, *Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation with Application to the Church Rosser Theorem*, Indag Math 34,5 381-392 (1972)

[2] Alonzo Church, *An Unsolvable Problem in Elementary Number Theory*, American Journal of Mathematics Vol.58 345-363 (1936)

[3] Alonzo Church, *A formulation of the simple theory of types*, Journal of Symbolic Logic Vol.5 56-68 (1940)

[4] Thierry Coquand, *An Analysis of Girard's Paradox*, Proc. Symposium on Logic in Computer Science, Cambridge Mass, (1986)

[5] Thomas Jech, *Set Theory*, The Third Millenium Edition, Springer 2002

[6] Robin Milner, *A Theory of Type Polymorphism in Programming*, Journal of Computer and System Sciences 17, 348-375 (1978)