A Comparison of HOL-ST and Isabelle/ZF

Sten Agerholm
University of Cambridge Computer Laboratory
New Museums Site, Cambridge CB2 3QG, UK

Abstract

The use of higher order logic (simple type theory) is often limited by its restrictive type system. Set theory allows many constructions on sets that are not possible on types in higher order logic. This paper presents a comparison of two theorem provers supporting set theory, namely HOL-ST and Isabelle/ZF, based on a formalization of the inverse limit construction of domain theory; this construction cannot be formalized in higher order logic directly. We argue that whilst the combination of higher order logic and set theory in HOL-ST has advantages over the first order set theory in Isabelle/ZF, the proof infrastructure of Isabelle/ZF has better support for set theory proofs than HOL-ST. Proofs in Isabelle/ZF are both considerably shorter and easier to write.

1 Introduction

Though higher order logic (simple type theory) is a useful framework for doing mathematics, there are situations where it is too weak, due to its simple type system. One can then use a stronger type theory, or observe that many constructions that are not possible on types in higher order logic would be possible on sets in set theory (which is completely untyped). Set theory might therefore provide a simple alternative to the increasing interest in applying stronger type theories in theorem proving.

Paulson has done a lot of pioneering work on mechanizing set theory. He has developed a very large amount of set theory in his Isabelle/ZF system [7, 8], which is an extension of a first order logic instantiation of the generic

theorem prover Isabelle [9] with axioms of Zermelo-Fraenkel (ZF) set theory. Gordon has also been experimenting with mechanizing set theory in an attempt to combine the usefulness of higher order logic with the expressive power of set theory in a single system [5]. A prototype system, called HOL-ST, has been implemented by extending the existing HOL system [4] with axioms of ZF set theory (this is not a conservative extension).

A larger case study on HOL-ST was presented in [1]. By formalizing the inverse limit construction of domain theory, which would not be possible in HOL directly [2], the case study demonstrated how one can make essential use of the additional expressive power of set theory. The inverse limit construction is a method to give solutions to recursive domain equations that may involve non-trivial constructions such as the (continuous) function space. In [1], it was used to obtain a non-trivial model D_∞ of the untyped λ-calculus, i.e. D_∞ was proved to be isomorphic to its own (continuous) function space:

$$D_\infty \cong [D_\infty \to D_\infty].$$

This paper presents a comparison of HOL-ST and Isabelle/ZF based on a formalization of the inverse limit construction in both systems. We concentrate on their different support for the formalization, i.e. for definitions, theorems and proofs, but do not give a detailed presentation of the formalization (see [1]). The version of the inverse limit construction employed here is based on categorical methods using embedding project pairs, see e.g. [6, 11, 10].

Comparing systems is difficult. The lack of some feature supported by one system does not mean that it could not be supported by another. In this paper, we have chosen to freeze time in the sense that the systems are compared in their state when this project started (Autumn 1994). The size and scope of the paper could easily grow out of hand if we had to argue about the alternatives for implementing all differences (and better proof support in general).

The rest of the paper is organized as follows. In Section 2, we introduce the HOL system briefly. Then the set theories of HOL-ST and Isabelle/ZF are introduced in Section 3. The comparison starts in Section 4 where we consider the definitions of the formalizations. It continues in Section 5 which discusses how the systems support the proofs of theorems of the formalization. Section 6 summarizes the conclusions. This paper assumes that the reader is familiar with Isabelle.

2 The HOL System

The HOL system is a mechanized proof-assistant for proving theorems in higher order logic. It implements on an expressive higher order logic (the HOL logic) and is built on top of a functional programming environment ML (which stands for Meta Language). The HOL logic is a typed logic with terms, types, and theorems represented in ML. The logic, which is described further below, is organized in theories each of which contains a set of types, constants, definitions, axioms and theorems. The purpose of the HOL system is to provide tools for constructing such theories.

Theories can be extended with new constants and types by giving definitions and axioms. Definitional extension is safe, which means that it preserves consistency of the HOL logic, because new constants and types are defined in terms of existing ones. Axiomatic extension is not safe and usually not accepted in the HOL community.

The representation of theorems in ML as an abstract type guarantees that theorems can only be created by formal proof. A proof is a derivation using a number of inference rules, pre-proved theorems and axioms. An inference rule is a function in ML which takes a number of theorems (premises) as arguments and produces a theorem as a result. All inference rules are derived from eight primitive rules, possibly using other so-called derived inference rules. Conversions are special cases of inference rules which take no theorem arguments but instead a term argument.

Inference rules support forward proofs of theorems. However, a more natural goal-directed (or backwards) proof style is also supported—by the subgoal package. This allows proofs of theorems to be constructed by applying tactics interactively, in order to reduce goal terms to truth. A tactic is an ML function which typically implements the backwards use of one or more inference rules.

2.1 Logic

While Isabelle [9] is a generic theorem prover, the HOL system [4] is not. It has only one hardwired logic. This is a higher order logic which is very similar to the higher order logic instantiation of Isabelle, which in turn shares some notions with the Isabelle meta logic.

The HOL logic is a polymorphic (classical) higher order logic (simply typed λ-calculus). As in Isabelle's meta logic, a term can be a constant, a variable, an abstraction, or an application. A term must be well-typed in the

usual sense. The syntax of types and terms is a bit different than in Isabelle. In HOL, the function type is written as "$:\alpha$ `->` β" and functions can be curried "$f:\alpha_1$ `->` (α_2 `->` ... `->` (α_n `->` β)...)" or uncurried "$g:\alpha_1$ `#` α_2 `#` ... `#` α_n `->` β)...)", where `#` is the operator for the product type. Function application is then written as $f\,x_1 \ldots\ x_n$ and $g(x_1, \ldots, x_n)$, respectively. The λ-abstraction is written as `\x. e`. Recall that in Isabelle's meta logic all functions must be uncurried and function application is always written using parentheses $f(x_1, \ldots, x_n)$.

Among others, the HOL logic provides the following symbols:

- Conjunction `/\`, disjunction `\/`, negation `~`, implication `==>`.
- Equality `=`.
- Universal quantification `!`, existential quantification `?`.
- Choice operator `@`.

Only the choice operator requires further explanation. It is a binder like the quantifiers which is used to obtain an arbitrary (fixed) value such that a certain predicate is satisfied; if this is not possible, it yields any value of the underlying type of the predicate (all types are non-empty to allow this).

Among the built-in types of the HOL system we find `bool`, which denotes the type of boolean truth values `T` and `F`, and `num`, which denotes the type of natural numbers whose elements are constructed from `0` and `SUC`.

2.2 Theorem Proving

The theorem proving infrastructure of Isabelle is mainly provided by the principle of resolution, which is based on (higher-order) unification. This elegantly supports a very small number of tools for forward and backward proof. By forward resolution, a theorem can be used like a HOL inference rule, whilst backward resolution turns the theorem into a HOL tactic. In HOL there is a separate ML function for each inference rule and tactic.

In some sence, theorem proving in HOL is more primitive than in Isabelle. HOL's notion of resolution is a derived form of a one way matching modus ponens which is performed on the assumptions of a goal using a conditional theorem. Further, it does not support quantifier reasoning via unknown variables as in Isabelle. Existentials must be instantiated manually and on the spot (though some very primitive user-contributed tools exist).

3 Set Theory

HOL-ST and Isabelle/ZF provide two slightly different ZF-like set theories. Isabelle/ZF is an axiomatic extension of the object logic FOL, which provides a first order logic. HOL-ST is an axiomatic extension of HOL's higher order logic. This means that HOL's set theory is slightly more powerful than Isabelle's (see [5]), though we shall not exploit this in an essential way in this paper. Apart from this difference in logic, the two axiomatizations of set theory are essentially the same. We will not consider the axioms of the set theories in this paper since they are not important; they are easy to look up in [5] and [9]. Instead we will focus on syntactic issues, to introduce set theory syntax used later.

3.1 Representation and Notation

HOL is extended with set theory by declaring a new type `V` and a new constant ':' (set membership) of type `":V#V->bool"`,[1] and then postulating eight new axioms about `V` and ':'. Similarly, Isabelle/FOL is extended by declaring a new type `i` and a new constant ':' (set membership) of type `"[i,i] => o"`, and then postulating the eight axioms of set theory. In Isabelle/ZF, the type `i` (for individuals) instantiates the many-sorted first order logic. Hence, we can quantify over sets as in `ALL x. x:X --> f(x):Y` (elements of sets are themselves sets), which could equivalently be written as `ALL x:X. f(x):Y`, and compare sets by the FOL equality `=`. The subset relation $\subseteq$ is written `Subset` in HOL-ST and `<=` in Isabelle/ZF.

Both systems provide a notation for set abstraction $\{x \in X \mid P[x]\}$. In Isabelle/ZF this is written `{x:X . P(x)}` and in HOL-ST `{x::X | P x}`.

3.2 Pairs and Numbers

Isabelle/FOL does not provide pairs or natural numbers but these are provided in set theory. Pairs are written `<x,y>`, which is a set and therefore has type `i`, and the usual destructors `fst` and `snd` are provided (both have the type `"i => i"`). The binary product `X * Y` consists of all pairs whose first component is in `X` and whose second component is in `Y`. In HOL-ST, we use pairs of higher order logic, but pairs are also available in set theory.

[1] In the type of the set membership operator, note that elements of sets are themselves sets. Generally speaking, new sets must be constructed from existing sets some way, in principle starting from the empty set and then using axioms.

In Isabelle/ZF, the set `nat` provides natural numbers, so `0:nat` and `succ(n):nat`, for any `n:nat`. In HOL-ST, natural numbers are available as both the type `":num"` and the set `Num`. The type and the set are isomorphic with translation functions called `num2Num` and `Num2num`. The HOL constants `0` and `SUC` are used with the translation functions to build members of `Num`.

3.3 Functions and Dependent Products

We distinguish between set and logical functions. Set functions are elements of the function set, written `X->Y` in both systems. A set function is represented by a set of pairs, which must satisfy the obvious conditions to specify a function (definedness and uniqueness). In HOL-ST, logical functions are functions of higher order logic. In Isabelle/ZF, logical functions are functions of the meta logic (the object logic FOL does not provide functions). Set function application is written `f ^^ x` in HOL-ST and `f ` x` in Isabelle/ZF. If `f` is in `X->Y` and `x` in `X` then we can conclude that `f` applied to `x` is in `Y`, otherwise not.

Set functions may be written using a certain (dependent) lambda abstraction whose syntax is `Fn x::X. b[x]` in HOL-ST and `lam x:X. b[x]` in Isabelle/ZF. The lambda abstraction consists of all set theory pairs of the form `<x,b[x]>`, where `x` is in `X`. Set function identity is written as `id` in Isabelle/ZF and as `Id` in HOL-ST. Set function composition is written as `O` in both systems.

Finally, both systems also provide a dependent product construction. The syntax is `PI x::X. Y[x]` in HOL-ST (see [1]) and `PROD x:X. Y[x]` in Isabelle/ZF. The elements of a dependent product are sets of pairs, corresponding to set functions that map elements `x` of the first set `X` to elements of the second (dependent) set `Y[x]`.

4 Definitions

In this section, we study the definitions of two formalizations of the inverse limit construction in HOL-ST and Isabelle/ZF. We try to compare the definitions and discuss the issue of which notions to represent in the meta logic of Isabelle/ZF and which to represent in first order logic and set theory. This is related to the question of whether to work in higher order logic or set theory in HOL-ST.

4.1 Basic Concepts

Recall that domain theory is the study of complete partial orders (cpos) and continuous functions. A partial order is usually thought of as a set with an associated ordering relation which is reflexive, transitive and antisymmetric on all elements of the set. A complete partial order is a partial order in which all non-decreasing chains (sequences) of elements of the partial order have a least upper bound. Continuous functions are monotonic functions that preserve such least upper bounds.

The differences between HOL-ST and Isabelle/ZF appear already in the first definitions of basic concepts of domain theory. In HOL-ST, we can represent the notion of partial order as a higher order logic pair consisting of a set and a relation. The predicate `po` for partial orders can then be defined by

```
po D =
 (!x :: set D. rel D x x) /\
 (!x y z :: set D. rel D x y /\ rel D y z ==> rel D x z) /\
 (!x y :: set D. rel D x y /\ rel D y x ==> (x = y)),
```

where the type of `po` is `":V#(V->V->bool)"`, and for convenience

```
set D = FST D
rel D = SND D.
```

Neither Isabelle's meta logic nor Isabelle/FOL support pairs. So, our only choice is to use pairs of set theory, though, alternatively, we could represent a partial order as a relation, defining the set as the reflexive elements. We chose the pair approach because it is closer to the HOL-ST formalization, which was done first, and because we then avoid proofs to show what the elements of the set component of cpo constructions are. Hence, the type of the constant `po` is `"i=>i"` in Isabelle/ZF. Its definition is the same as the one above, though the right-hand side uses symbols of first order logic instead of higher order logic. The `set` and `rel` constants are defined by

```
"set(D) == fst(D)"
"rel(D, x, y) == <x, y> : snd(D)".
```

This makes `rel` a meta logic function of type `"[i,i,i] => o"`; `set` has type `"i => i"`. It later turns out (when we consider subcpos) that it would have been more appropriate to define `rel` slightly differently as a meta logic function of just one argument (`D`) which returns a set function (or

relation) of two arguments (`x` and `y`). In terms, we would then write the less convenient `rel(D) ‘ x ‘ y` (or `<x, y> : rel(D)`) instead of the present `rel(D, x, y)`.

Next, we introduce the notion of a chain, which is a sequence of values in non-decreasing order:

```
chain D X =
 (!n. (X n) :: (set D)) /\ (!n. rel D(X n)(X(n + 1))).
```

Hence, in HOL-ST a chain is represented as a logical function of type `":num->V"`. Complete partial orders are then defined as follows

```
cpo D = po D /\ (!X. chain D X ==> (?x. islub D X x))
```

where the notion of least upper bound (lub) is defined by

```
isub D X x = x :: (set D) /\ (!n. rel D(X n)x)
islub D X x =
 isub D X x /\ (!y. isub D X y ==> rel D x y).
```

Using Hilbert's choice operator we can give an expression for the least upper bound (if it exists):

```
lub D X = (@x. islub D X x).
```

In Isabelle/ZF, neither the meta logic nor first order logic support natural numbers so we must turn to the set of natural numbers, called `nat`, to represent infinite sequences this way. One could represent chains as a meta logic function of type `"i=>i"`, where the first `i` would correspond to the set of natural numbers and the second would correspond to the underlying set of a cpo. However, this representation is problematic. In the definition of `cpo` we must quantify over chains but in first order logic this is impossible since we can only quantify over individuals, not meta logic functions like such chains of type `"i=>i"`. Hence, in Isabelle, chains must be represented as functions in set theory; thus, chains have type `i`. The Isabelle/ZF definition of `chain` looks as follows:

```
"chain(D,X) ==
  X:nat->set(D) & (ALL n:nat. rel(D,X‘n,X‘(succ(n))))",
```

which use more set theory than the HOL-ST definition. The first order logic definition of `cpo` is

```
"cpo(D) ==
  po(D) & (ALL X. chain(D,X) --> (EX x. islub(D,X,x)))",
```

where `islub` is defined as in HOL-ST. As above, we define a constant for the least upper bound but we use the definite description operator instead of Hilbert's choice operator (which is not available, though it probably could be axiomatized):

```
"lub(D, X) == THE x. islub(D, X, x)".
```

The term `"THE x. P(x)"` is read 'the x such that $P(x)$' and requires both existence and uniqueness. In contrast, the HOL logic provides the choice operator, which just requires existence, and this is inherited by set theory, which thus automatically satisfies the axiom of choice. The use of the definite description operator made a few proofs slightly more complicated in Isabelle/ZF than in HOL-ST, due to the additional obligation of proving uniqueness.

A consequence of representing chains as functions in set theory is that the type checking, which ensures arguments of chains are numbers, must be done manually. Similarly, proving that chains are functions, and not just relations, and proving that they are functions on the right domains must be done manually as well (though usually the λ-abstraction is used and then it is only necessary to check the body of this due to a pre-proved theorem). Thus, proving that terms are chains is more complicated in Isabelle/ZF than in HOL-ST.

4.2 Continuous Functions

Monotonic and continuous functions have essentially the same definitions in the two systems. We only list the HOL-ST definitions:

```
mono(D,E) =
 {f :: (set D) -> (set E) |
  !x y :: set D. rel D x y ==> rel E(f ^^ x)(f ^^ y)}
cont(D,E) =
 {f :: mono(D,E) |
  !X. chain D X ==> (f^^(lub D X) = lub E(\n. f^^(X n)))}.
```

Functions are represented in set theory because we wish continuous functions to constitute a cpo, called the continuous function space, and the underlying set of a cpo must be a set. The continuous function space construction is defined as follows in HOL-ST:

```
cf(D,E) =
 cont(D,E),(\f g. !x :: set D. rel E(f ^^ x)(g ^^ x)).
```

However, the construction is defined slightly differently in Isabelle/ZF, due to the fact that the cpo pair, and more importantly the underlying relation, must be defined in set theory entirely:

```
"cf(D,E) ==
  <cont(D,E),
   {y : cont(D,E) * cont(D,E).
    ALL x:set(D). rel(E,(fst(y))`x,(snd(y))`x)}>".
```

In Isabelle/ZF, the relation must be constructed from existing sets, i.e. it must be constructed from the domains `D` and `E` in the function space. In contrast, the HOL-ST relation is just a higher order logic function. The Isabelle relation not only looks more complicated: due to additional type checking, it is also more complicated to use. Each time we define a construction on cpos, which we do twice below, we will experience a similar complication due to the set relation.

4.3 The Inverse Limit Construction

Next, we consider the definitions of some concepts associated with the inverse limit construction. Inverse limits may be viewed as "least upper bounds" of "chains" of cpos, not just of chains of elements of cpos as above. The ordering on elements of cpos is generalized to the notion of embedding morphisms between cpos. A certain constant `Dinf`, parameterized by a chain of cpos, can be proven once and for all to yield the inverse limit of the chain. This cannot be defined in higher order logic directly (assuming that the underlying set of a cpo is represented as a subset of a HOL type, as in [2]), since it yields a cpo of infinite tuples whose components may be in different cpos (subsets of types). Defining the construction in higher order logic would require a (probably difficult) conservative derivation of a universal type with dependent products. However, formalizing the construction is straightforward in set theory, exploiting the dependent product construction on sets (see Section 3.3).

Embedding morphisms come in pairs with projections, forming the so-called embedding projection pairs. The HOL-ST definition of (embedding) projection pairs is stated as follows:

```
projpair(D,E)(e,p) =
 e :: (cont(D,E)) /\
 p :: (cont(E,D)) /\
 (p O e = Id(set D)) /\
 rel(cf(E,E))(e O p)(Id(set E)).
```

The Isabelle/ZF definition is similar. The conditions make sure that the structure of `E` is richer than that of `D` (and can contain it). `D` is embedded into `E` by `e` (one-one) which in turn is projected onto `D` by `p`.

Embeddings uniquely determine projections (and vice versa). Hence, it is enough to consider embeddings

```
emb(D,E)e = (?p. projpair(D,E)(e,p))
```

and define the associated projections, or retracts as they are often called, using the choice operator:

```
Rp(D,E)e = (@p. projpair(D,E)(e,p)).
```

Again, these are the HOL-ST definitions; the Isabelle/ZF definitions are similar (though the definite description operator is used instead of the choice operator as in Section 4.1).

Embeddings are used to form chains of cpos in a similar way to the formation of chains from elements of cpos. Recall that standard chains are represented as logical functions of type `":num->V"` in HOL-ST and as set functions of type `i` in Isabelle/ZF. We choose to stick to this difference when representing embedding chains of cpos. Hence, the HOL-ST definition is stated like this:

```
emb_chain DD ee =
 (!n. cpo(DD n)) /\ (!n. emb(DD n,DD(SUC n))(ee n)).
```

And the Isabelle/ZF definition is:

```
"emb_chain(DD, ee) ==
  (ALL n:nat. cpo(DD ` n)) &
  (ALL n:nat. emb(DD ` n, DD ` succ(n), ee ` n))".
```

We do not quantify over embedding chains in any definitions immediately and therefore we could perhaps represent such chains as meta logic functions of type `"i=>i"` in Isabelle. However, the above choice is safer, in case it turns out that we later wish to quantify over chains.

One is often in a situation where a function can be represented in the meta logic or in the object logic (set theory). In general, one should only choose the first alternative if the function is not really part of a formalization and thus never would appear in the right-hand side of definitions (without its arguments). Hence, it is fine to use meta logic functions for constants in abbreviations (but one must be careful which I was not when I defined `rel` and `rho_emb`, see below). However, a choice must be made when functions are arguments of constants. For instance, due to the above criteria, we would use a meta logic function for the predicate of the following construction

```
"mkcpo(D, P) ==
  <{x: set(D) . P(x)},
   {x: set(D) * set(D) . rel(D, fst(x), snd(x))}>",
```

which is useful for constructing a subcpo of a cpo by restricting the set component according to a predicate. Thus, the type of `mkcpo` is `"[i,i=>o]=>i"`. But most constants with function arguments would have a type of the form `"[i,i]=>o"`, e.g. `emb_chain` above, where functions are set functions.

The notion of subcpo is defined as follows in Isabelle/ZF:

```
"subcpo(D, E) ==
  set(D) <= set(E) &
  (ALL x:set(D).
    ALL y:set(D). rel(D, x, y) <-> rel(E, x, y)) &
  (ALL X. chain(D, X) --> lub(E, X) : set(D))".
```

Both this and the previous definition of `mkcpo` have simpler formulations in HOL-ST:

```
mkcpo D P = {x :: set D | P x},rel D
subcpo D E =
 (set D) Subset (set E) /\
 (rel D = rel E) /\
 (!X. chain D X ==> (lub E X) :: (set D)).
```

In both Isabelle/ZF definitions, the complications are due to a mismatch between the type of `"rel(D)"`, namely `"[i,i]=>o"`, and the type of the relation component of cpos, namely `i`. As mentioned above, we probably made a bad choice in not representing `"rel(D)"` as a set function (or a set relation) instead of as a logical function. The problem in the `mkcpo` definition arises due to the fact that each component of a pair must be

a set. The problem in the `subcpo` definition is that meta logic functions cannot be compared using FOL equality `=`.

The constant `mkcpo` is used to define the inverse limit construction on cpos as a subcpo of the infinite Cartesian product cpo. Let us first consider the HOL-ST definition of the infinite product:

```
iprod DD =
 (PI n :: Num. set(DD(Num2num n))),
 (\x y. !n. rel(DD n)(x ^^ (num2Num n))(y ^^ (num2Num n))).
```

The relation is defined componentwise and the set is the infinite tuples whose `i`'th component is in `"DD i"`; in general, the dependent product `"PI x :: X. Y[x]"` consists of the functions that map an element `x` of `X` to an element of `Y[x]`. This construction cannot be defined on HOL types (though it might be possible to derive a universal type with this construction). The annoying `num2Num` and `Num2num` conversions could be avoided by using the set of numbers `Num` instead of the type of numbers `":num"` to represent chains of cpos. However, the present choice makes proofs simpler in the long run (see [1]). The reason for this is associated with the choice of using the type of numbers in the representation of ordinary chains. To avoid the translation functions, we would have to stay within set theory all the time, since the dependent product construction is only available there.

The Isabelle/ZF definition of the infinite product construction is:

```
"iprod(DD) ==
  <PROD n:nat. set(DD ` n),
   {x: (PROD n:nat. set(DD`n)) * (PROD n:nat. set(DD`n)) .
    ALL n:nat. rel(DD ` n, fst(x) ` n, snd(x) ` n)}>".
```

Here, the translation functions are avoided since we do not have the choice of leaving set theory. However, for the same reason, the definition and use of the componentwise relation is much more complicated, since the relation must be a set constructed from existing sets.

The definitions of the inverse limit construction are essentially the same in the two system, both use the `mkcpo` constant. The HOL-ST definition is stated as follows:

```
Dinf DD ee =
 mkcpo
 (iprod DD)
 (\x.
```

```
!n.
 (Rp(DD n,DD(SUC n))(ee n)) ^^ (x ^^ (num2Num(SUC n))) =
 x ^^ (num2Num n)).
```

The only difference is that the Isabelle/ZF definition quantifies over elements of the set of natural numbers (instead of over elements of the type as above) and it does not use translation functions. Informally, the underlying set of `Dinf` is defined as the subset of all infinite tuples x on which the n-th projection (retract) e_n^R maps the $(n+1)$-st index to the n-th index for all n: $e_n^R(x_{n+1}) = x_n$. The underlying relation is inherited from the infinite product construction.

It takes a fairly large development to prove that `Dinf` yields an inverse limit of any chain of cpos. For the proof, we need an embedding of any element `"DD n"` of the chain `"(DD,ee)"` into the inverse limit `"Dinf DD ee"`. This embedding is defined as follows in HOL-ST

```
rho_emb DD ee n =
 (Fn x :: set(DD n).
   Fn m :: Num. (eps DD ee n(Num2num m)) ^^ x),
```

which was copied almost directly to Isabelle/ZF (removing the translation function):

```
 "rho_emb(DD, ee, n) ==
   lam x:set(DD ` n). lam m:nat. eps(DD, ee, n, m) ` x".
```

The definitions of the constant called `eps` in both systems are not important here[2]. While the definition of `rho_emb` worked fine in HOL-ST we realized at a later stage that the Isabelle/ZF definition should have been

```
"rho_emb(DD, ee) ==
  lam n:nat. lam x:set(DD`n). lam m:nat. eps(DD,ee,n,m)`x",
```

where `rho_emb` is a logical function of just two arguments instead of three; thus, while `"rho_emb(DD,ee)"` above was a logical function of type `"i => i"`, it should have been a set function (of type `i`). The present representation would be unfortunate if we wanted to define a constant for the property that `Dinf` always yields the inverse limit of a cpo. This is not possible using a meta logic function for `"rho_emb(DD,ee)"`, since the definition would

[2] By composing embeddings (and projections) `eps` generalizes the embeddings `"ee n"` between consecutive cpos of a chain to convert between any two cpos.

need to quantify over such sequences of embeddings. Furthermore, similar sequences like the sequences of cpos `DD` and embeddings `ee` are represented as object logic functions. So, a constant may be a meta logic function of some arguments and an object logic function of other arguments. If one is not careful the wrong choices are made.

5 Proofs

In the previous section, we concentrated on the differences between using HOL-ST and Isabelle/ZF to formalize the definitions of the inverse limit construction of domain theory. In this section, we discuss how the two systems support the proofs of related theorems (see [3] for a more detailed discussion).

Due to limitations of Isabelle's first-order set theory, we were forced to work in set theory in situations where we could stay in higher order logic in HOL-ST. As mentioned above, this obviously yields more complicated proofs in Isabelle, in the sense of more type conditions to prove. For instance, in a backward proof of a statement saying that two functions are related by the continuous function space relation, we would first rewrite with the HOL-ST theorem

```
rel(cf(D,E))f g =
 (!x. x :: (set D) ==> rel E(f ^^ x)(g ^^ x)),
```

but in the Isabelle/ZF the first step would be to resolve with the theorem

```
"[| !!x. x : set(D) ==> rel(E, f ` x, g ` x);
    f : cont(D, E); g : cont(D, E) |] ==>
 rel(cf(D, E), f, g)",
```

which, in contrast to the HOL-ST theorem, contains type assumptions. The same thing is true of the other constructions on cpos, like the infinite Cartesian product and the inverse limit constructions. Similarly, the necessity of representing chains as set functions yields a number of additional proof obligations in the Isabelle/ZF proofs.

Despite these additional proof obligations, Isabelle proofs are usually shorter in terms of number of lines, and easier to write. Usually, backward proofs are reduced in size (number of lines) by more than 50% and in some cases by 75%. The main reasons for this are Isabelle's support for unknown variables for quantifier reasoning and the design of its proof infrastructure.

The main method of proof in Isabelle is based on resolution (with higher order unification), which supports both the forward and the backward style of proof. In fact, the same theorem can be used as an inference rule by forward resolution and as a tactic by backward resolution. In this way, Isabelle elegantly avoids the need for a large collection of ML functions implementing derived inference rules and tactics. Furthermore, it supports a compact notation for proofs since the main resolution tactic can be employed with a theorem list argument, and repeated.

Further, the notion of resolution in Isabelle supports 'real' backward proof better than in HOL. One almost always works from the conclusion of a goal backward towards the assumptions, which is supported by Isabelle resolution tactics. In HOL, one often ends up doing a lot of sometimes ugly assumption hacking working forward using HOL resolution from the assumptions towards the conclusion. More natural backward strategies like conditional rewriting and a matching modus ponens style strategy (which may be viewed as a simplified version of Isabelle resolution) are not supported well in HOL.

Real backward strategies are useful due the fact that many theorems have assumptions. It is irritating to have to first derive the antecedents of theorems for HOL resolution. On the other hand, a negative consequence of using theorems in a real backward fashion is that existential quantifiers are often introduced. For instance, this happens when we employ the transitivity of a cpo relation or the fact that function composition preserves the function set (or continuity or embeddings):

```
"[| g : A -> B; f : B -> C |] ==> f O g : A -> C".
```

In HOL, we must provide witnesses for existentials on the spot and manually. But in Isabelle, both universally and existentially quantified variables are represented as unknown variables that are usually instantiated behind the scenes in proofs, possibly in stages.

Finally, the Isabelle subgoal module provides a kind of flat structure on proof states which makes it possible to access all goals at any time and to prove many (or all) subgoals by just repeating a tactic—no matter where the subgoals would appear in a HOL proof tree. Isabelle takes care of applying the theorems for resolution, instantiating unknowns and proving the assumptions by adding them as new subgoals; we do not have to think about the tree structure of proofs and about which tactics (or theorems) to apply where.

6 Conclusions

We have presented a comparison of HOL-ST and Isabelle/ZF based on a case study from domain theory. The case study formalizes an important construction which yields the inverse limit of any embedding projection chains of cpos. This formalization exploits set theory in an essential way since it requires a dependent product construction that cannot be defined on HOL types. The main observations say that HOL-ST is supported by the powerful HOL logic whereas Isabelle/ZF provides better proof support for set theory. The HOL logic gives a more convenient set theory because set and type theoretic reasoning can be mixed to advantage. However, generally speaking, HOL lacks ways of handling conditional theorems conveniently, and does not provide the support for unknown variables for quantifier reasoning available in Isabelle. Set theory introduces a lot of set membership assumptions in theorems as well as the need for real backward proof strategies and good support for quantifier reasoning.

It is advantageous to be able to exploit higher order logic where possible, as in HOL-ST, since one of the main disadvantages of set theory is the presence of explicit type (set membership) conditions. This means that type checking is done late by theorem proving whereas in higher order logic type checking is done early in ML. Furthermore, type checking is automatic in HOL but cannot be fully automated in set theory. On the other hand, a disadvantage of mixing higher order logic and set theory as in HOL-ST is the need for translation functions to identify HOL types with their corresponding sets. Therefore, it is not obvious whether set theory in higher order logic is right, or just more support for set theory in first order logic is needed.

Acknowledgements

The research described here is partly supported by EPSRC grant GR/G23654 and partly by an HCMP fellowship under the EuroForm network. I am grateful to Mike Gordon for encouragements and discussions concerning this work. I would also like to thank Larry Paulson for answering my questions about Isabelle promptly and the Isabelle users at Cambridge, in particular Sara Kalvala, for help and discussions. Mark Staples commented on a draft of the paper.

References

[1] S. Agerholm. Formalising a model of the λ-calculus in HOL-ST. Technical Report 354, University of Cambridge Computer Laboratory, November 1994.

[2] S. Agerholm. *A HOL Basis for Reasoning about Functional Programs.* PhD thesis, BRICS, Department of Computer Science, University of Aarhus, December 1994. Available as Technical Report RS-94-44.

[3] S. Agerholm. A comparison of HOL-ST and Isabelle/ZF. Technical Report 369, University of Cambridge Computer Laboratory, 1995.

[4] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem-proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.

[5] M.J.C. Gordon. Merging HOL with set theory: preliminary experiments. Technical Report 353, University of Cambridge Computer Laboratory, November 1994.

[6] L. C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF.* Cambridge Tracts in Theoretical Computing 2, Cambridge University Press, 1987.

[7] L. C. Paulson. Set theory for verification: I. From foundations to functions. *Journal of Automated Reasoning*, 11(3):353–389, 1993.

[8] L. C. Paulson. Set theory for verification: II. Induction and recursion. Technical Report 312, University of Cambridge Computer Laboratory, 1993.

[9] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.

[10] G. Plotkin. *Domains.* Course notes, Department of Computer Science, University of Edinburgh, 1983.

[11] M. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11, 1982.