A Design Structure for Higher Order Quotients

Peter V. Homeier

U. S. Department of Defense
palantir@trustworthytools.com, http://www.trustworthytools.com

Abstract. The quotient operation is a standard feature of set theory, where a set is partitioned into subsets by an equivalence relation. We reinterpret this idea for higher order logic, where types are divided by an equivalence relation to create new types, called quotient types. We present a design to mechanically construct quotient types as new types in the logic, and to support the automatic lifting of constants and theorems about the original types to corresponding constants and theorems about the quotient types. This design exceeds the functionality of Harrison's package, creating quotients of multiple mutually recursive types simultaneously, and supporting the equivalence of aggregate types, such as lists and pairs. Most importantly, this design supports the creation of higher order quotients, which enable the automatic lifting of theorems with quantification over functions of any higher order.

1 Introduction

The quotient operation is a standard feature of mathematics, including set theory and abstract algebra. It provides a way to cleanly identify elements that previously were distinct. This simplifies the system by removing unneeded structure.

Traditionally, quotients [4] have found many applications. Classic examples are the construction of the integers from pairs of non-negative natural numbers, or the rationals from pairs of integers. In the lambda calculus [1], it is common to identify terms which are alpha-equivalent, that differ only by the choice of local names used by binding operators. Other examples include the construction of bags from lists by ignoring order, and sets from bags by ignoring duplicates.

The ubiquity of quotients has recommended their investigation within the field of mechanical theorem proving. The first to appear was Ton Kalker's 1989 package for HOL88 [11]. Isabelle/HOL [14] has mechanical support for the creation of higher order quotients by Oscar Slotosch [19], using partial equivalence relations represented as a type class, with equivalence relations as a subclass. That system provides a definitional framework for establishing quotient types, including higher order. Independently, Larry Paulson has shown a construction of first-order quotients in Isabelle without any use of the Hilbert choice operator [17]. PVS uses quotients to support theory interpretations [15]. MetaPRL has quotients in its foundations, as a type with a new equality [16]. Coq, based on the Calculus of Constructions [9], supports first order quotients [5] but has some difficulties with higher order [3]. These systems provide little automatic support. In particular, there is no automatic lifting of constants or theorems.

John Harrison has developed a package for the HOL theorem prover which supports first order quotients, including automation to define new quotient types and to lift to the quotient level both constants and theorems previously established [8]. This automatic lifting is key to practical support for quotients. A quotient of a group would be incomplete without also mapping the original group operation to a corresponding one for the quotient group. Similarly, theorems about the group which are independent of the equivalence relation should also be true of the quotient group. Mechanizing this lifting is vital for avoiding the repetition of proofs at the higher level which were already proved at the lower level. Such automation is not only practical, but mathematically incisive.

Despite the quality of Harrison's excellent package, it does have limitations. It can only lift one type at a time, and does not deal with aggregate types, such as lists or pairs involving types being lifted, which makes it difficult to lift a family of mutually nested recursive types. Most importantly, it is limited to lifting only first order theorems, where quantification is permitted over the type being lifted, but not over functions or predicates involving the type being lifted.

In this paper we describe a design for a new package for quotients [10] for the Higher Order Logic theorem prover that meets all these concerns. It provides a tool for lifting multiple types across multiple equivalence relations simultaneously. Aggregate equivalence relations are produced and used automatically. But most significantly, this package supports the automatic lifting of theorems that involve higher order functions, including quantification, of any finite order. This is possible through the use of *partial equivalence relations* [2, 18], as a possibly non-reflexive variant of equivalence relations, enabling the creation of quotients of function types. The relationship between these partial equivalence relations and their associated abstraction and representation functions (mapping between the lower and higher types) is expressed in *quotient theorems*, which are central.

The precise definition in section 3 of the quotient relationship between the original and lifted types, and the proof in section 5.2 of that relationship's preservation for a function type, given existing quotients for the function's domain and range, are the heart of this paper, and are presented in full detail. These form the core theory that justifies the treatment of all higher order functions, including higher order universal, existential, and unique existential quantification.

The structure of this paper is as follows. Section 2 discusses equivalence relations and their extensions. Section 3 defines partial equivalence relations and quotient theorems. Section 4 shows the construction of a new quotient type in HOL. Section 5 explains the extension of quotients for aggregate and function types. Section 6 explores an alternative design that avoids use of the Axiom of Choice. Section 7 touches on highlights of the implementation and its required inputs. Section 8 exhibits an example. Finally, section 9 presents our conclusions.

We thank the TPHOLs referees for their cogent and constructive comments. We are grateful for the helpful comments and suggestions made by Rob Arthan, Randolph Johnson, Sylvan Pinsky, Yvonne V. Shashoua, and Konrad Slind, and especially Michael Mislove for identifying partial equivalence relations, and William Schneeberger for the key idea in the proof of Theorem 19.

2 Equivalence Relations and Equivalence Theorems

Before considering quotients, we examine equivalence relations, on which such traditional quotients as those mentioned in the introduction have been based.

Let τ be any type. A binary relation R on τ can be represented in HOL as a curried function of type $\tau \to (\tau \to \texttt{bool})$. We will take advantage of the curried nature of R, where $R\ x\ y = (R\ x)\ y$.

An equivalence relation is a binary relation E satisfying

$$\begin{array}{lll} \textit{reflexivity:} & \forall x : \tau. & E\ x\ x \\ \textit{symmetry:} & \forall x\ y : \tau. & E\ x\ y\ \Rightarrow\ E\ y\ x \\ \textit{transitivity:} & \forall x\ y\ z : \tau. & E\ x\ y\ \wedge\ E\ y\ z\ \Rightarrow\ E\ x\ z \end{array}$$

These three properties are encompassed in the *equivalence property:*

$$\textit{equivalence:} \qquad \texttt{EQUIV}\ E \stackrel{\text{def}}{=} \forall x\ y : \tau.\ E\ x\ y\ \Leftrightarrow\ (E\ x = E\ y)$$

A theorem of the form $\vdash \texttt{EQUIV}\ E$ is called an *equivalence theorem* on type τ.

2.1 Equivalence Extension Theorems

Given an equivalence relation $E : \tau \to \tau \to \texttt{bool}$ on values of type τ, there is a natural extension of E to values of lists of τ. This is expressed as `LIST_REL` E, which forms an equivalence relation of type τ `list` $\to \tau$ `list` $\to$ `bool`. Similarly, equivalence relations on pairs, sums, and options may be formed from their constituent types' equivalence relations by the following operators.

Type	Operator	Type of operator
list	`LIST_REL :`	`('a -> 'a -> bool) ->` `'a list -> 'a list -> bool`
pair	`### :`	`('a -> 'a -> bool) -> ('b -> 'b -> bool) ->` `'a # 'b -> 'a # 'b -> bool`
sum	`+++ :`	`('a -> 'a -> bool) -> ('b -> 'b -> bool) ->` `'a + 'b -> 'a + 'b -> bool`
option	`OPTION_REL :`	`('a -> 'a -> bool) ->` `'a option -> 'a option -> bool`

These operators are easily defined in the expected way [10]. They are used to build an equivalence relation with a structure analogous to the type operator structure of the type of the elements compared by the relation.

Using these relation extension operators, the aggregate type operators `list`, `prod`, `sum`, and `option` have the following *equivalence extension theorems:*

`LIST_EQUIV:` $\vdash \forall E.\ \texttt{EQUIV}\ E\ \Rightarrow\ \texttt{EQUIV}\ (\texttt{LIST\_REL}\ E)$
`PAIR_EQUIV:` $\vdash \forall E_1\ E_2.\ \texttt{EQUIV}\ E_1\ \Rightarrow\ \texttt{EQUIV}\ E_2\ \Rightarrow\ \texttt{EQUIV}\ (E_1\ \texttt{\#\#\#}\ E_2)$
`SUM_EQUIV:` $\vdash \forall E_1\ E_2.\ \texttt{EQUIV}\ E_1\ \Rightarrow\ \texttt{EQUIV}\ E_2\ \Rightarrow\ \texttt{EQUIV}\ (E_1\ \texttt{+++}\ E_2)$
`OPTION_EQUIV:` $\vdash \forall E.\ \texttt{EQUIV}\ E\ \Rightarrow\ \texttt{EQUIV}\ (\texttt{OPTION\_REL}\ E)$

3 Partial Equivalence Relations and Quotient Theorems

In this section we introduce a new definition of the quotient relationship, based on *partial equivalence relations* (PERs), related to but different from equivalence relations. Every equivalence relation is a partial equivalence relation, but not every partial equivalence relation is an equivalence relation. An equivalence relation is reflexive, symmetric and transitive, while a partial equivalence relation is symmetric and transitive, but not necessarily reflexive on all of its domain.

Why use partial equivalence relations with a weaker reflexivity condition? The reason involves forming quotients of higher order types, that is, functions whose domains or ranges involve types being lifted. Unlike lists and pairs, the equivalence relations for the domain and range do not naturally extend to a useful equivalence relation for functions from the domain to the range.

The reason is that not all functions which are elements of the function type are *respectful* of the associated equivalence relations, as described in [10]. For example, given an equivalence relation $E : \tau \to \tau \to \texttt{bool}$, the set of functions from τ to τ may contain a function $f^?$ where for some x and y which are equivalent $(E\ x\ y)$, the results of $f^?$ are not equivalent $(\neg(E\ (f^?\ x)\ (f^?\ y)))$. Such disrespectful functions cannot be worked with; they do not correspond to any function at the abstract quotient level. Suppose instead that $f^?$ did lift. Let $\lceil\phi\rceil$ be the lifted version of ϕ. As $\lceil f^?\rceil$ is the lifted version of $f^?$, it should act just like $f^?$ on its argument, except that it should not consider the lower level details that E disregards. Thus $\forall u.\ \lceil f^?\rceil\lceil u\rceil = \lceil f^?\ u\rceil$. Then certainly $\forall u\ v.\ E\ u\ v \Leftrightarrow (\lceil u\rceil = \lceil v\rceil)$, and because $E\ x\ y$, we must have $\lceil x\rceil = \lceil y\rceil$. Applying $\lceil f^?\rceil$ to both sides, $\lceil f^?\rceil\lceil x\rceil = \lceil f^?\rceil\lceil y\rceil$. But this implies $\lceil f^?\ x\rceil = \lceil f^?\ y\rceil$, which means that $E\ (f^?\ x)\ (f^?\ y)$, which we have said is false, a contradiction. Therefore such disrespectful functions cannot be lifted, and we must exclude them. Using partial equivalence relations accomplishes this exclusion.

First, we say an element r *respects* R if and only if $R\ r\ r$.

Definition 1 (Quotient). *A relation R with abstraction function abs and representation function rep (between the representation, lower type τ and the abstract, quotient type ξ) is a quotient (notated as $\langle R,abs,rep\rangle$) if and only if*

$$
\begin{array}{ll}
(1) & \forall a : \xi.\ abs\ (rep\ a) = a \\
(2) & \forall a : \xi.\ R\ (rep\ a)\ (rep\ a) \\
(3) & \forall r, s : \tau.\ R\ r\ s \Leftrightarrow R\ r\ r \wedge R\ s\ s \wedge (abs\ r = abs\ s)
\end{array}
$$

Property 1 states that *rep* is a right inverse of *abs*.

Property 2 states that the range of *rep* respects R.

Property 3 states that two elements of τ are related by R if and only if each element respects R and their abstractions are equal.

These three properties (1-3) describe the way the partial equivalence relation R works together with *abs* and *rep* to establish the correct quotient relationship between the lower type τ and the quotient type ξ. The precise definition of this quotient relationship is a central contribution of this work. This relationship is defined in the HOL logic as a new predicate:

```
QUOTIENT (R:'a -> 'a -> bool) (abs:'a -> 'b) (rep:'b -> 'a) ⇔
     (∀a. abs (rep a) = a) ∧
     (∀a. R (rep a) (rep a)) ∧
     (∀r s. R r s ⇔ R r r ∧ R s s ∧ (abs r = abs s))
```

The relationship that R with *abs* and *rep* is a quotient is expressed in HOL as

$$\vdash \texttt{QUOTIENT}\ R\ abs\ rep\ .$$

A theorem of this form is called a *quotient theorem*. The identity is $\vdash \langle \texttt{\$=}, \texttt{I}, \texttt{I} \rangle$.

These three properties support the inference of a quotient theorem for a function type, given quotient theorems for the domain and the range. This key inference is central and necessary to enable higher order quotients.

4 Quotient Types

The user may specify a quotient of a type τ by a relation R (written τ/R) by giving either a theorem that the relation is an equivalence relation, of the form

$$\vdash \forall x\ y.\ R\ x\ y\ \Leftrightarrow\ (R\ x = R\ y)\ , \tag{1}$$

or one that the relation is a nonempty partial equivalence relation, of the form

$$\vdash (\exists x.\ R\ x\ x)\ \wedge\ (\forall x\ y.\ R\ x\ y\ \Leftrightarrow\ R\ x\ x \wedge R\ y\ y \wedge (R\ x = R\ y))\ . \tag{2}$$

In this section we will develop the second, more difficult case. The first follows immediately. In the following, x, y, r, $s : \tau$, $c : \tau \to \texttt{bool}$, and $a : \tau/R$.

New types may be defined in HOL using the function `new_type_definition` [6, sections 18.2.2.3-5]. This function requires us to choose a representing type, and a predicate on that type denoting a subset that is nonempty.

Definition 2. *We define the new quotient type τ/R as isomorphic to the subset of the representing type $\tau \to \texttt{bool}$ by the predicate $P : (\tau \to \texttt{bool}) \to \texttt{bool}$, where $P\ c \stackrel{\text{def}}{=} \exists x.\ R\ x\ x \wedge (c = R\ x)$.*

P is nonempty because $P\ (R\ x)$ for the $x : \tau$ such that $R\ x\ x$ by (2). Let $\xi = \tau/R$. The HOL tool `define_new_type_bijections` [6] automatically defines a function $abs_c : (\tau \to \texttt{bool}) \to \xi$ and its right inverse $rep_c : \xi \to (\tau \to \texttt{bool})$ satisfying

Definition 3. (a) $\forall a : \xi.\ abs_c\ (rep_c\ a) = a$
(b) $\forall c : \tau \to \texttt{bool}.\ P\ c\ \Leftrightarrow\ rep_c\ (abs_c\ c) = c$

PER classes are subsets of τ (of type $\tau \to \texttt{bool}$) which satisfy P. Then abs_c and rep_c map between the quotient type ξ and PER classes (hence the "c").

Lemma 4 (rep_c maps to PER classes). $\forall a.\ P\ (rep_c\ a)$.

Proof: By Definition 3(a), $abs_c\ (rep_c\ a) = a$, so taking the rep_c of both sides, $rep_c\ (abs_c\ (rep_c\ a)) = rep_c\ a$. By Definition 3(b), $P\ (rep_c\ a)$. □

Lemma 5. $\forall r.\ R\ r\ r \Rightarrow (rep_c\ (abs_c\ (R\ r)) = R\ r)$.

Proof: Assume $R\ r\ r$; then $P\ (R\ r)$. By Definition 3(b), the goal follows.

Lemma 6 (abs_c is one-to-one on PER classes).
$\forall r\ s.\ R\ r\ r \Rightarrow R\ s\ s \Rightarrow (abs_c\ (R\ r) = abs_c\ (R\ s) \ \Leftrightarrow\ R\ r = R\ s)$.

Proof: Assume $R\ r\ r$ and $R\ s\ s$. The right-to-left implication of the biconditional is trivial. Assume $abs_c\ (R\ x) = abs_c\ (R\ y)$. Applying rep_c to both sides gives us $rep_c\ (abs_c\ (R\ x)) = rep_c\ (abs_c\ (R\ y))$. Then by Lemma 5 twice, $R\ x = R\ y$. □

The functions abs_c and rep_c map between PER classes of type $\tau \to \texttt{bool}$ and the quotient type ξ. Using these functions, we can define new functions *abs* and *rep* between the original type τ and the quotient type ξ as follows.

Definition 7 (Quotient abstraction and representation functions).

$$\begin{array}{ll} abs : \tau \to \xi & abs\ r \stackrel{\text{def}}{=} abs_c\ (R\ r) \\ rep : \xi \to \tau & rep\ a \stackrel{\text{def}}{=} \texttt{\$@}\ (rep_c\ a)\ \ (=\ \texttt{@}r.\ rep_c\ a\ r\ ) \end{array}$$

The `@` operator is a higher order version of Hilbert's choice operator ε [6, 12]. It has type $(\alpha \to \texttt{bool}) \to \alpha$, and is usually used as a binder, where $\texttt{\$@}\ P = \texttt{@}x.\ P\ x$. (The `$` converts an operator to prefix syntax.) `@` satisfies the HOL axiom $\forall P\ x.\ P\ x \Rightarrow P\ (\texttt{\$@}\ P)$. Given any predicate P on a type, if any element of the type satisfies the predicate, then $\texttt{\$@}\ P$ returns an arbitrary element of that type which satisfies P. If no element of the type satisfies P, then $\texttt{\$@}\ P$ will return simply some arbitrary, unknown element of the type. Such definitions have been questioned by constructivist critics of the Axiom of Choice. An alternative design for quotients avoiding the Axiom of Choice is described in section 6.

Lemma 8. $\forall r.\ R\ r\ r \Rightarrow (R\ (\texttt{\$@}\ (R\ r)) = R\ r)$.

Proof: The axiom for the `@` operator is $\forall P\ x.\ P\ x \Rightarrow P\ (\texttt{\$@}\ P)$. Taking $P = R\ r$ and $x = r$, we have $R\ r\ r \Rightarrow R\ r\ (\texttt{\$@}\ R\ r)$. Assuming $R\ r\ r$, $R\ r\ (\texttt{\$@}\ (R\ r))$ follows. Then by (2), $R\ r\ (\texttt{\$@}\ (R\ r))$ implies the equality $R\ r = R\ (\texttt{\$@}\ (R\ r))$. □

Theorem 9. $\forall a.\ abs\ (rep\ a) = a$

Proof: By Lemma 4 and the definition of P, for each a there exists an r such that $R\ r\ r$ and $rep_c\ a = R\ r$. Then by Lemma 8, $R\ (\texttt{\$@}\ (R\ r)) = R\ r$. Now by Definition 7, $abs\ (rep\ a) = abs_c\ (R\ (\texttt{\$@}\ (rep_c\ a)))$, which simplifies by the above and Definition 3(a) to a. □

Theorem 10. $\forall a.\ R\ (rep\ a)\ (rep\ a)$.

Proof: As before, for each a there exists an r such that $R\ r\ r$ and $rep_c\ a = R\ r$.

$$\begin{array}{lll} R\ (rep\ a)\ (rep\ a) & \Leftrightarrow\ R\ (\texttt{\$@}\ (rep_c\ a))\ (\texttt{\$@}\ (rep_c\ a)) & \text{Definition 7} \\ & \Leftrightarrow\ R\ (\texttt{\$@}\ (R\ r))\ (\texttt{\$@}\ (R\ r)) & \text{selection of } r \\ & \Leftrightarrow\ R\ r\ (\texttt{\$@}\ (R\ r)) & \text{Lemma 8} \\ & \Leftrightarrow\ R\ (\texttt{\$@}\ (R\ r))\ r & \text{symmetry of } R \\ & \Leftrightarrow\ R\ r\ r\ \Leftrightarrow\ T & \text{Lemma 8, selection of } r \end{array}$$

□

Theorem 11. $\forall r\ s.\ \ R\ r\ s\ \Leftrightarrow\ R\ r\ r\ \wedge\ R\ s\ s\ \wedge\ (abs\ r = abs\ s)$

Proof:

$$\begin{aligned} R\ r\ s &\Leftrightarrow R\ r\ r\ \wedge\ R\ s\ s\ \wedge\ (R\ r = R\ s) && (2)\\ &\Leftrightarrow R\ r\ r\ \wedge\ R\ s\ s\ \wedge\ (abs_c\ (R\ r) = abs_c\ (R\ s)) && \text{Lemma 6}\\ &\Leftrightarrow R\ r\ r\ \wedge\ R\ s\ s\ \wedge\ (abs\ r = abs\ s) && \text{Definition 7} \end{aligned}$$

□

Theorem 12. $\langle R,\ abs,\ rep\rangle$.

Proof: By Theorems 9, 10, and 11, with Definition 1. □

5 Aggregate and Higher Order Quotient Theorems

Traditional quotients that lift τ to a set of τ also lift lists of τ to sets of lists of τ. These sets are isomorphic to lists, but *they are not lists*. In this design, when τ is lifted to ξ, then we lift lists of τ to lists of ξ. We preserve the type operator structure built on top of the types being lifted. Similarly, we want to preserve polymorphic constants. In a theorem being lifted, we want an occurrence of $\texttt{HD} : \tau\ \texttt{list} \rightarrow \tau$ to lift to an occurrence of $\texttt{HD} : \xi\ \texttt{list} \rightarrow \xi$. If such a constant is not lifted to itself, the lifted version of the theorem will not look like the original. Hence Definition 1 was designed to preserve the vital type operator structure.

In the process of lifting constants and theorems, quotient theorems are needed for each argument and result type of each constant being lifted. For aggregate and higher order types, the tool automatically proves any needed quotient theorems from the available quotient theorems for the constituent subtypes. To accomplish this, the tool uses *quotient extension theorems* (section 5.2). These are provided preproven for some standard type operators. For others, new quotient extension theorems may be manually proven and then included to extend the tool's power.

5.1 Aggregate and Higher Order PERs and Map Operators

Some aggregate equivalence relation operators have been already described in section 2, and these can equally be used to build aggregate partial equivalence relations. In addition, for function types, the following is added:

Type Operator	Type of operator
fun	`===> : ('a -> 'a -> bool) -> ('b -> 'b -> bool) -> ('a -> 'b) -> ('a -> 'b) -> bool`

Definition 13. $(R_1\ \texttt{===>}\ R_2)\ f\ g\ \Leftrightarrow\ \forall x\ y.\ R_1\ x\ y \Rightarrow R_2\ (f\ x)\ (g\ y)$.

Note R_1 `===>` R_2 is *not* in general an equivalence relation (it is not reflexive). It is reflexive at a function f, $(R_1\ \texttt{===>}\ R_2)\ f\ f$, if and only if f is respectful.

The quotient theorems created for aggregate types involve not only aggregate partial equivalence relations, but also aggregate abstraction and representation functions. These are constructed from the component abstraction and representation functions using the following "map" operators.

Type	Operator	Type of operator, examples of *abs* and *rep* fns
list	`MAP` :	`('a -> 'b) -> 'a list -> 'b list` *examples:* (`MAP` *abs*) , (`MAP` *rep*)

pair	`##` :	`('a -> 'b) -> ('c -> 'd) ->`
		`'a # 'c -> 'b # 'd`
		examples: (abs_1 `##` abs_2) , (rep_1 `##` rep_2)

sum	`++` :	`('a -> 'b) -> ('c -> 'd) ->`
		`'a + 'c -> 'b + 'd`
		examples: (abs_1 `++` abs_2) , (rep_1 `++` rep_2)

option	`OPTION_MAP` :	`('a -> 'b) -> 'a option -> 'b option` *examples:* (`OPTION_MAP` *abs*) , (`OPTION_MAP` *rep*)

fun	`-->` :	`('a -> 'b) -> ('c -> 'd) ->`
		`('b -> 'c) -> 'a -> 'd`
		examples: (rep_1 `-->` abs_2) , (abs_1 `-->` rep_2)

The above operators are easily defined in the expected way [10], if not already present in standard HOL. The identity quotient map operator is the identity operator $\mathtt{I} : \alpha \to \alpha$. The function map operator definition is of special interest:

Definition 14. $(f$ `-->` $g)\ h\ x \stackrel{\text{def}}{=} g\ (h\ (f\ x))$.

5.2 Quotient Extension Theorems

Here are the quotient extension theorems for the `list`, `prod`, `sum`, `option`, and, most significantly, `fun` type operators:

`LIST_QUOTIENT`:
$\vdash \forall R\ abs\ rep.\ \langle R, abs, rep\rangle \Rightarrow \langle$`LIST_REL` R, `MAP` abs, `MAP` $rep\rangle$

`PAIR_QUOTIENT`:
$\vdash \forall R_1\ abs_1\ rep_1.\ \langle R_1, abs_1, rep_1\rangle \Rightarrow\ \forall R_2\ abs_2\ rep_2.\ \langle R_2, abs_2, rep_2\rangle \Rightarrow$
$\langle R_1$ `###` R_2, abs_1 `##` abs_2, rep_1 `##` $rep_2\rangle$

`SUM_QUOTIENT`:
$\vdash \forall R_1\ abs_1\ rep_1.\ \langle R_1, abs_1, rep_1\rangle \Rightarrow\ \forall R_2\ abs_2\ rep_2.\ \langle R_2, abs_2, rep_2\rangle \Rightarrow$
$\langle R_1$ `+++` R_2, abs_1 `++` abs_2, rep_1 `++` $rep_2\rangle$

`OPTION_QUOTIENT`:
$\vdash \forall R\ abs\ rep.\ \langle R, abs, rep\rangle \Rightarrow$
$\langle$`OPTION_REL` R, `OPTION_MAP` abs, `OPTION_MAP` $rep\rangle$

FUN_QUOTIENT:

$$\vdash \forall R_1\ abs_1\ rep_1.\ \langle R_1, abs_1, rep_1\rangle \Rightarrow\ \forall R_2\ abs_2\ rep_2.\ \langle R_2, abs_2, rep_2\rangle \Rightarrow \langle R_1 \texttt{===>} R_2,\ \ rep_1 \texttt{-->} abs_2,\ \ abs_1 \texttt{-->} rep_2\rangle$$

This last theorem is of central and critical importance to forming higher order quotients. We present here its proof in detail.

Theorem 15 (Function quotients). *If relations R_1 and R_2 with abstraction functions abs_1 and abs_2 and representation functions rep_1 and rep_2, respectively, are quotients, then R_1 ===> R_2 with abstraction function rep_1 --> abs_2 and representation function abs_1 --> rep_2 is a quotient.*

Proof: We need to prove the three properties of Definition 1:

Property 1. Prove for all a, $(rep_1 \texttt{-->} abs_2)\ ((abs_1 \texttt{-->} rep_2)\ a) = a$.
Proof: The equality here is between functions, and by extension, true if for all values x in a's domain, $(rep_1 \texttt{-->} abs_2)\ ((abs_1 \texttt{-->} rep_2)\ a)\ x = a\ x$.
By the definition of -->, this is $abs_2\ ((abs_1 \texttt{-->} rep_2)\ a\ (rep_1\ x)) = a\ x$, and then $abs_2\ (rep_2\ (a\ (abs_1\ (rep_1\ x)))) = a\ x$. By Property 1 of $\langle R_1,abs_1,rep_1\rangle$, $abs_1\ (rep_1\ x) = x$, and by Property 1 of $\langle R_2,abs_2,rep_2\rangle$, $\forall b.\ abs_2\ (rep_2\ b) = b$, so this reduces to $a\ x = a\ x$, true.

Property 2. Prove $(R_1 \texttt{===>} R_2)\ ((abs_1 \texttt{-->} rep_2)\ a)\ ((abs_1 \texttt{-->} rep_2)\ a)$.
Proof: By the definition of ===>, this is
$\forall x, y.\ R_1\ x\ y\ \Rightarrow R_2\ ((abs_1 \texttt{-->} rep_2)\ a\ x)\ ((abs_1 \texttt{-->} rep_2)\ a\ y)$. Assume $R_1\ x\ y$, and show $R_2\ ((abs_1 \texttt{-->} rep_2)\ a\ x)\ ((abs_1 \texttt{-->} rep_2)\ a\ y)$. By the definition of -->, this is $R_2\ (rep_2\ (a\ (abs_1\ x)))\ (rep_2\ (a\ (abs_1\ y)))$. Now since $R_1\ x\ y$, by Property 3 of $\langle R_1,abs_1,rep_1\rangle$, $abs_1\ x = abs_1\ y$. Substituting this into our goal, we must prove $R_2\ (rep_2\ (a\ (abs_1\ y)))\ (rep_2\ (a\ (abs_1\ y)))$. But this is an instance of Property 2 of $\langle R_2,abs_2,rep_2\rangle$, and so the goal is proven.

Property 3. Prove $(R_1 \texttt{===>} R_2)\ r\ s \Leftrightarrow$
$(R_1 \texttt{===>} R_2)\ r\ r \wedge (R_1 \texttt{===>} R_2)\ s\ s \wedge ((rep_1 \texttt{-->} abs_2)\ r = (rep_1 \texttt{-->} abs_2)\ s)$.
The last conjunct on the right side is equality between functions, so by extension this is $(R_1 \texttt{===>} R_2)\ r\ s \Leftrightarrow (R_1 \texttt{===>} R_2)\ r\ r \wedge (R_1 \texttt{===>} R_2)\ s\ s \wedge (\forall x.\ (rep_1 \texttt{-->} abs_2)\ r\ x = (rep_1 \texttt{-->} abs_2)\ s\ x)$.
By the definitions of ===> and -->, this is (1) $\Leftrightarrow$ (2) $\wedge$ (3) $\wedge$ (4), where

(1) $(\forall x\ y.\ R_1\ x\ y\ \Rightarrow R_2\ (r\ x)\ (s\ y))$
(2) $(\forall x\ y.\ R_1\ x\ y\ \Rightarrow R_2\ (r\ x)\ (r\ y))$
(3) $(\forall x\ y.\ R_1\ x\ y\ \Rightarrow R_2\ (s\ x)\ (s\ y))$
(4) $(\forall x.\ (abs_2\ (r\ (rep_1\ x)) = abs_2\ (s\ (rep_1\ x))))$.

We prove (1) $\Leftrightarrow$ (2) $\wedge$ (3) $\wedge$ (4) as a biconditional with two goals.

Goal 1. ($\Rightarrow$) Assume (1). Then we must prove (2), (3), and (4).

Subgoal 1.1. (Proof of (2)) Assume $R_1\ x\ y$. We must prove $R_2\ (r\ x)\ (r\ y)$. From $R_1\ x\ y$ and Property 3 of $\langle R_1,abs_1,rep_1\rangle$, we also have $R_1\ x\ x$ and $R_1\ y\ y$. From (1) and $R_1\ x\ y$, we have $R_2\ (r\ x)\ (s\ y)$. From (1) and $R_1\ y\ y$, we have $R_2\ (r\ y)\ (s\ y)$. Then by symmetry and transitivity of R_2, the goal is proven.

Subgoal 1.2. (Proof of (3)) Similar to the previous subgoal.

Subgoal 1.3. (Proof of (4)) $R_1\ (rep_1\ x)\ (rep_1\ x)$ follows from Property 2 of $\langle R_1,abs_1,rep_1\rangle$. From (1), we have $R_2\ (r\ (rep_1\ x))\ (s\ (rep_1\ x))$. Then the goal follows from this and the third conjunct of Property 3 of $\langle R_2,abs_2,rep_2\rangle$.

Goal 2. ($\Leftarrow$) Assume (2), (3), and (4). We must prove (1). Assume $R_1\ x\ y$. Then we must prove $R_2\ (r\ x)\ (s\ y)$. From $R_1\ x\ y$ and Property 3 of $\langle R_1,abs_1,rep_1\rangle$, we also have $R_1\ x\ x$, $R_1\ y\ y$, and $abs_1\ x = abs_1\ y$. By Property 3 of $\langle R_2,abs_2,rep_2\rangle$, the goal is $R_2\ (r\ x)\ (r\ x) \wedge R_2\ (s\ y)\ (s\ y) \wedge abs_2\ (r\ x) = abs_2\ (s\ y)$. This breaks into three subgoals.

Subgoal 2.1. Prove $R_2\ (r\ x)\ (r\ x)$. This follows from $R_1\ x\ x$ and (2).

Subgoal 2.2. Prove $R_2\ (s\ y)\ (s\ y)$. This follows from $R_1\ y\ y$ and (3).

Subgoal 2.3. Prove $abs_2\ (r\ x) = abs_2\ (s\ y)$.

> **Lemma.** *If $\langle R,abs,rep\rangle$ and $R\ z\ z$, then $R\ (rep\ (abs\ z))\ z$.*
> $R\ (rep\ (abs\ z))\ (rep\ (abs\ z))$, by Property 2 of $\langle R,abs,rep\rangle$. From the hypothesis, $R\ z\ z$. From Property 1 of $\langle R,abs,rep\rangle$, $abs\ (rep\ (abs\ z)) = abs\ z$. From these three statements and Property 3 of $\langle R,abs,rep\rangle$, we have $R\ (rep\ (abs\ z))\ z$. □

By the lemma and $R_1\ x\ x$, we have $R_1\ (rep_1\ (abs_1\ x))\ x$. Similarly, by the lemma and $R_1\ y\ y$, we have $R_1\ (rep_1\ (abs_1\ y))\ y$. Then by (2), we have $R_2\ (r\ (rep_1\ (abs_1\ x)))\ (r\ x)$, and by (3), $R_2\ (s\ (rep_1\ (abs_1\ y)))\ (s\ y)$. From these and Property 3 of $\langle R_2,abs_2,rep_2\rangle$,

$$abs_2\ (r\ (rep_1\ (abs_1\ x))) = abs_2\ (r\ x) \text{ and}$$
$$abs_2\ (s\ (rep_1\ (abs_1\ y))) = abs_2\ (s\ y).$$

But by $abs_1\ x = abs_1\ y$ and (4), the left hand sides of these two equations are equal, so their right hand sides must be also, $abs_2\ (r\ x) = abs_2\ (s\ y)$, which proves the goal. □

6 The Axiom of Choice

Gregory Moore wrote that "Rarely have the practitioners of mathematics, a discipline known for the certainty of its conclusions, differed so vehemently over one of its central premises as they have done over the Axiom of Choice. Yet without the Axiom, mathematics today would be quite different" [13]. Today, this discussion continues. Some theorem provers are based on classical logic, and others on a constructivist logic. In higher order logic, the Axiom of Choice is represented by Hilbert's ε-operator [12, §4.4], also called the indefinite description operator. Paulson's lucid recent work [17] exhibits an approach to quotients which avoids the use of Hilbert's ε-operator, by instead using the definite description operator ι [14, §5.10]. These two operators may be axiomatized as follows:

$$\forall P\ x.\ P\ x \Rightarrow P(\varepsilon\ P) \qquad or \qquad \forall P.\ (\exists x.\ P\ x) \Rightarrow P(\varepsilon\ P)$$
$$\forall P\ x.\ P\ x \Rightarrow (\forall y.\ P\ y \Rightarrow x = y) \Rightarrow P(\iota\ P) \qquad or \qquad \forall P.\ (\exists! x.\ P\ x) \Rightarrow P(\iota\ P)$$

The ι-operator yields the single element of a singleton set, $\iota\{z\} = z$, but its result on non-singleton sets is indeterminate. By contrast, the ε-operator chooses some

indeterminate element of any non-empty set, even if nondenumerable. The ι-operator is weaker than the ε-operator, and less objectionable to constructivists.

Thus it is of interest to determine if a design for higher order quotients may be formulated using only ι, not ε. Inspired by Paulson, we investigate this by forming an alternative design, eliminating the representation functions.

Definition 16 (Constructive quotient, replacing Definition 1). *A relation R with abstraction function abs (between the representation type τ and the abstraction type ξ) is a quotient (notated as $\langle R, abs\rangle$) if and only if*

$$\begin{array}{ll}(1) & \forall a : \xi.\ \exists r : \tau.\ R\ r\ r \wedge (abs\ r = a)\\ (2) & \forall r\ s : \tau.\ R\ r\ s \Leftrightarrow R\ r\ r \wedge R\ s\ s \wedge (abs\ r = abs\ s)\end{array}$$

Property 1 states that for every abstract element a of ξ there exists a representative in τ which respects R and whose abstraction is a.

Property 2 states that two elements of τ are related by R if and only if each element respects R and their abstractions are equal.

The quotients for new quotient types based on (partial) equivalence relations may now be constructed by a modified version of §4, where the representation function *rep* is omitted from Definition 7, so there is no use of the Hilbert ε-operator. Property 1 follows from Lemma 4. The identity quotient is $\langle \texttt{\$=}, \texttt{I}\rangle$. From Definition 16 also follow analogs of the quotient extension theorems, e.g.,

$$\forall R\ abs.\ \langle R,\ abs\rangle \Rightarrow \langle \texttt{LIST\_REL}\ R,\ \texttt{MAP}\ abs\rangle$$

for lists and similarly for pairs, sums and option types. Functions are lifted by the abstraction operation for functions, which requires two new definitions:

$$\begin{array}{rcl}(abs \Downarrow R)\ a\ r & \stackrel{\text{def}}{=} & R\ r\ r\ \wedge\ abs\ r = a\\ (reps\ \texttt{+->}\ abs)\ f\ x & \stackrel{\text{def}}{=} & \iota\ (\texttt{IMAGE}\ abs\ (\texttt{IMAGE}\ f\ (reps\ x)))\end{array}$$

Note that for the identity quotient, $(\texttt{I} \Downarrow \texttt{\$=}) = \texttt{\$=}$.

The critical quotient extension theorem for functions has also been proven:

Theorem 17 (Function quotient extension).

$$\langle R_1,\ abs_1\rangle \Rightarrow \langle R_2,\ abs_2\rangle \Rightarrow \langle R_1\ \texttt{===>}\ R_2,\ (abs_1 \Downarrow R_1)\ \texttt{+->}\ abs_2\rangle$$

Unfortunately, the proof requires using the Axiom of Choice. In fact, this theorem implies the Axiom of Choice, in that it implies the existence of an operator which obeys the axiom of the Hilbert ε-operator, as seen by the following development.

Theorem 18 (Partial abstraction quotients). *If f is any function from type α to β, and Q is any predicate on α, such that $\forall y{:}\beta.\ \exists x{:}\alpha.\ Q\ x \wedge (f\ x = y)$, then the partial equivalence relation $R = \lambda r\ s.\ Q\ r \wedge Q\ s \wedge (f\ r = f\ s)$ with abstraction function f is a quotient ($\langle R, f\rangle$).*

Proof: Follows easily from substituting R in Definition 16 and simplifying. □

Theorem 19 (Partial inverses exist). *If f is any function from type α to β, and Q is any predicate on α, such that $\forall y{:}\beta.\ \exists x{:}\alpha.\ Q\ x \wedge (f\ x = y)$, then there exists a function g such that $f \circ g = \mathtt{I}$ and $\forall y.\ Q\ (g\ y)$.* [William Schneeburger]

Proof: Assuming the function quotient extension theorem 17, we apply it to two quotient theorems; first, the identity quotient $\langle \mathtt{\$=}, \mathtt{I} \rangle$ for type β, and second, the partial abstraction quotient $\langle R, f \rangle$ from Theorem 18. This yields the quotient $\langle \mathtt{\$=\ ===>}\ R,\ \mathtt{\$=\ +->}\ f \rangle$, since $(\mathtt{I} \Downarrow \mathtt{\$=}) = \mathtt{\$=}$. By Property 1 of Definition 16, $\forall a.\ \exists r.\ (\mathtt{\$=\ ===>}\ R)\ r\ r \wedge ((\mathtt{\$=\ +->}\ f)r = a)$. Specializing $a = \mathtt{I} : \beta \to \beta$, and renaming r as g, we obtain $\exists g.\ (\mathtt{\$=\ ===>}\ R)\ g\ g \wedge (\mathtt{\$=\ +->}\ f)g = \mathtt{I})$. By the definition of $\mathtt{===>}$, $(\mathtt{\$=\ ===>}\ R)g\ g$ is $\forall x\ y.\ x = y \Rightarrow R\ (g\ x)\ (g\ y)$, which simplifies by the definition of R to $\forall y.\ Q\ (g\ y)$. The right conjunct is $(\mathtt{\$=\ +->}\ f)g = \mathtt{I}$, which by the definition of $\mathtt{+->}$ is $(\lambda x.\ \iota\ (\mathtt{IMAGE}\ f\ (\mathtt{IMAGE}\ g\ (\mathtt{\$=}\ x)))) = \mathtt{I}$. But $\mathtt{\$=}\ x$ is the singleton $\{x\}$, so since $\mathtt{IMAGE}\ h\ \{z\} = \{h\ z\}$, $\iota\{z\} = z$, and $(\lambda x.\ f\ (g\ x)) = f \circ g$, this simplifies to $f \circ g = \mathtt{I}$, and the conclusion follows. □

Theorem 20 (Existence of Hilbert choice). *There exists an operator $c : (\alpha \to \mathtt{bool}) \to \alpha$ which obeys the Hilbert choice axiom, $\forall P\ x.\ P\ x \Rightarrow P\ (c\ P)$.*

Proof: In Theorem 19, let $Q = (\lambda(P{:}\alpha \to \mathtt{bool},\ a{:}\alpha).\ (\exists x.\ P\ x) \Rightarrow P\ a)$ and $f = \mathtt{FST}$. Then its antecedent is $\forall P'.\ \exists (P, a).\ ((\exists x.P\ x) \Rightarrow P\ a) \wedge (\mathtt{FST}(P, a) = P')$. For any P', take $P = P'$, and if $\exists x.\ P\ x$, then take a to be such an x. Otherwise take a to be any value of α. In either case the antecedent is true. Therefore by Theorem 19 there exists a function g such that $\mathtt{FST} \circ g = \mathtt{I}$ and $\forall P.\ Q\ (g\ P)$, which is $\forall P.\ (\exists x.\ (\mathtt{FST}\ (g\ P))\ x) \Rightarrow (\mathtt{FST}\ (g\ P))\ (\mathtt{SND}\ (g\ P))$. The operator c is taken as $\mathtt{SND} \circ g$, and since $\mathtt{FST}\ (g\ P) = P$, the Hilbert choice axiom follows. □

The significance of Theorem 20 is that even if we are able to avoid all use of the Axiom of Choice up to this point, it is not possible to prove the function quotient extension theorem 17 without it. This section's design may be used to build a theory of quotients which is constructive and which extends naturally to quotients of lists, pairs, sums, and options. However, it is not possible to extend it to higher order quotients while remaining constructive. Therefore the designs presented in this paper cannot be used to create higher order quotients in strictly constructive theorem provers. Alternatively, in theorem provers like HOL which admit the Hilbert choice operator, if higher order quotients are desired, there is no advantage in avoiding using the Axiom of Choice through using the design of this section. The main design presented earlier is much simpler to automate.

7 Implementation

The design for higher order quotients presented here has been implemented in a package for the Higher Order Logic theorem prover. This section will touch on only a few interesting aspects of the implementation; for further details, see [10].

This implementation provides a tool which accomplishes all the three tasks of lifting types, constants, and theorems. To do these, the tool requires inputs of several kinds. For each new quotient type to be created, the user must provide a

(partial) equivalence theorem (§4). For each kind of aggregate type involved, the user must provide a quotient extension theorem, and if possible, an equivalence extension theorem. For every constant which is mentioned in a theorem to be lifted, there must be a *respectfulness theorem* showing that the constant respects the equivalence relations. In addition, for polymorphic constants that can apply to arguments of either the lower or the quotient types, both a respectfulness theorem and a *preservation theorem* must be provided, which shows that the function of the polymorphic constant is preserved across the quotient operation.

$$\texttt{COND\_RSP:}\ \langle R, abs, rep\rangle \Rightarrow (a_1 = a_2)\ \wedge\ R\ b_1\ b_2\ \wedge\ R\ c_1\ c_2\ \Rightarrow\ R\ (\textbf{if}\ a_1\ \textbf{then}\ b_1\ \textbf{else}\ c_1)\ (\textbf{if}\ a_2\ \textbf{then}\ b_2\ \textbf{else}\ c_2)$$

$$\texttt{COND\_PRS:}\ \langle R, abs, rep\rangle \Rightarrow \textbf{if}\ a\ \textbf{then}\ b\ \textbf{else}\ c = abs\ (\textbf{if}\ a\ \textbf{then}\ rep\ b\ \textbf{else}\ rep\ c)$$

$$\texttt{RES\_FORALL\_RSP:}\ \langle R, abs, rep\rangle \Rightarrow (R\ \texttt{===>}\ \texttt{\$=})\ f\ g\ \Rightarrow\ \texttt{RES\_FORALL}(\texttt{respects}\ R)\ f = \texttt{RES\_FORALL}(\texttt{respects}\ R)\ g$$

$$\texttt{FORALL\_PRS:}\ \langle R, abs, rep\rangle \Rightarrow (\texttt{\$}\forall\ f = \texttt{RES\_FORALL}(\texttt{respects}\ R)\ ((abs\ \texttt{-->}\ \texttt{I})\ f)$$

Interestingly, $\forall$ is not respectful. To lift, theorems using $\forall$ are automatically converted to ones using `RES_FORALL`. `RES_FORALL` (`respects` R) P is the universal quantification of P, restricted to values of the argument of P which respect R. A large number of these respectfulness and preservation theorems have been preproven for standard operators, including, e.g., the unique existential quantifier. The natural power of higher order quotients is smoothly exploited in enabling these respectfulness and preservation theorems to be used to lift theorems containing curried operators with none, some, or all of their arguments present.

8 Example: Finite Sets

To demonstrate the higher order quotients package, we create finite sets as a new type, starting from the existing type of lists, `'a list`.

Lists are represented in HOL as a free algebra with two distinct constructors, `NIL` and `CONS`, also written as `[]` and infix `::` respectively. Let *A, B, C* be lists.

We intend to create the new type of finite sets as the quotient of lists by the equivalence relation $\sim$, generated by rule induction on the following six rules:

$$\frac{}{a::(b::A) \sim b::(a::A)} \qquad \frac{}{[] \sim []} \qquad \frac{A \sim B}{B \sim A}$$

$$\frac{}{a::(a::A) \sim a::A} \qquad \frac{A \sim B}{a::A \sim a::B} \qquad \frac{A \sim B,\ \ B \sim C}{A \sim C}$$

It is easy to prove that $\sim$ is in fact an equivalence relation, reflexive, symmetric, and transitive, and so `fset_EQUIV:` $\vdash \forall A\ B.\ A \sim B \Leftrightarrow (\texttt{\$}\sim A = \texttt{\$}\sim B)$.

Theorems may be proved by induction using the list induction principle:

$$\forall P : \texttt{'a list} \rightarrow \texttt{bool}.\ P\ [] \ \wedge\ (\forall t.\ P\ t \Rightarrow \forall h.\ P\ (h::t))\ \Rightarrow\ \forall l.\ P\ l$$

Membership and concatenation (which lifts to "union") are predefined:

$$\begin{array}{rcl@{\quad\wedge\quad}rcl} \texttt{MEM}\ x\ [] & = & F & \texttt{MEM}\ x\ (h::t) & = & (x = h) \vee \texttt{MEM}\ x\ t \\ \texttt{APPEND}\ []\ l & = & l & \texttt{APPEND}\ (h::l_1)\ l_2 & = & h::(\texttt{APPEND}\ l_1\ l_2) \end{array}$$

We define new constants by primitive recursion, and prove extensionality:

$$\begin{array}{rcl}
\texttt{[]}\ \texttt{Delete1}\ x & = & \texttt{[]} \\
(a\texttt{::}A)\ \texttt{Delete1}\ x & = & \textbf{if}\ a = x\ \textbf{then}\ A\ \texttt{Delete1}\ x\ \textbf{else}\ a\texttt{::}(A\ \texttt{Delete1}\ x)
\end{array}$$

$$\begin{array}{rcl}
\texttt{Fold1}\ f\ g\ z\ \texttt{[]} & = & z \\
\texttt{Fold1}\ f\ g\ z\ (a\texttt{::}A) & = & \textbf{if}\ (\forall u\ v.\ f\ u\ v = f\ v\ u) \wedge \\
 & & \quad (\forall u\ v\ w.\ f\ u\ (f\ v\ w) = f\ (f\ u\ v)\ w) \\
 & & \textbf{then if}\ \texttt{MEM}\ a\ A\ \textbf{then}\ \texttt{Fold1}\ f\ g\ z\ A \\
 & & \qquad\qquad \textbf{else}\ f\ (g\ a)\ (\texttt{Fold1}\ f\ g\ z\ A) \\
 & & \textbf{else}\ z
\end{array}$$

$$A \sim B\ \Leftrightarrow\ \forall a.\ \texttt{MEM}\ a\ A \Leftrightarrow \texttt{MEM}\ a\ B$$

Before invoking the quotient package, we must first prove the respectfulness theorems of each of the operators we wish to lift, `NIL_RSP`, `CONS_RSP`, etc., e.g.,

$$\frac{}{\texttt{[]} \sim \texttt{[]}} \quad \frac{A \sim B}{a\texttt{::}A \sim a\texttt{::}B} \quad \frac{A \sim B}{\texttt{MEM}\ a\ A = \texttt{MEM}\ a\ B} \quad \frac{A_1 \sim A_2, \quad B_1 \sim B_2}{\texttt{APPEND}\ A_1\ B_1 \sim \texttt{APPEND}\ A_2\ B_2}$$

$$\frac{A \sim B}{\texttt{Card1}\ A = \texttt{Card1}\ B} \quad \frac{A \sim B}{A\ \texttt{Delete1}\ a \sim B\ \texttt{Delete1}\ a} \quad \frac{A \sim B}{\texttt{Fold1}\ f\ g\ z\ A = \texttt{Fold1}\ f\ g\ z\ B}$$

We intend to lift the following constants on lists to new ones on finite sets:

`[]` ↦ `Empty`	`MEM` ↦ `In`	`APPEND` ↦ `Union`	`Delete1` ↦ `Delete`
`::` ↦ `Insert`	`Card1` ↦ `Card`	`Inter1` ↦ `Inter`	`Fold1` ↦ `Fold`

We now create the new type `'a finite_set` from the quotient of lists by ∼.

```
val [In, Union, finite_set_EXTENSION, ... finite_set_INDUCT] =
 define_quotient_types{
  types = [{name = "finite_set", equiv = fset_EQUIV}],
  defs=[{def_name="In_def",fname="In", fixity=Infix(NONASSOC,425),
         func=``MEM:'a -> 'a list -> bool``}, ... ],
  tyop_equivs = [],
  tyop_quotients = [FUN_QUOTIENT],
  tyop_simps = [FUN_REL_EQ, FUN_MAP_I],
  respects = [NIL_RSP, CONS_RSP, MEM_RSP, APPEND_RSP,
              Card1_RSP, Delete1_RSP, Inter1_RSP, Fold1_RSP],
  poly_preserves = [FORALL_PRS, EXISTS_PRS, COND_PRS],
  poly_respects = [RES_FORALL_RSP, RES_EXISTS_RSP, COND_RSP],
  old_thms = [MEM, APPEND, list_EXTENSION, ... list_INDUCT]};
```

This proves and stores the quotient theorem

$$\vdash \texttt{QUOTIENT}\ \texttt{\$}\sim\ \texttt{finite\_set\_ABS}\ \texttt{finite\_set\_REP}.$$

It also defines the lifted versions of the constants, for example

$$\vdash \forall T_1\ T_2.\ T_1\ \texttt{Insert}\ T_2 = \texttt{finite\_set\_ABS}\ (T_1\ \texttt{::}\ \texttt{finite\_set\_REP}\ T_2)$$

The theorems listed in `old_thms` are automatically soundly lifted to the quotient level, with the types changed, now concerning not lists but finite sets, e.g.,

$$\begin{array}{rcl@{\quad}rcl} x\ \texttt{In}\ \texttt{Empty} & = & F & \wedge & x\ \texttt{In}\ (h\ \texttt{Insert}\ t) & = & (x = h) \vee x\ \texttt{In}\ t \\ \texttt{Empty}\ \texttt{Union}\ l & = & l & \wedge & (h\ \texttt{Insert}\ l_1)\ \texttt{Union}\ l_2 & = & h\ \texttt{Insert}\ (l_1\ \texttt{Union}\ l_2) \end{array}$$

$$\begin{array}{rcl} \texttt{Empty}\ \texttt{Delete}\ x & = & \texttt{Empty} \\ (a\ \texttt{Insert}\ A)\ \texttt{Delete}\ x & = & \textbf{if}\ a = x\ \textbf{then}\ A\ \texttt{Delete}\ x \\ & & \qquad\qquad \textbf{else}\ a\ \texttt{Insert}\ (A\ \texttt{Delete}\ x) \\ \texttt{Fold}\ f\ g\ z\ \texttt{Empty} & = & z \\ \texttt{Fold}\ f\ g\ z\ (a\ \texttt{Insert}\ A) & = & \textbf{if}\ (\forall u\ v.\ f\ u\ v = f\ v\ u) \wedge \\ & & \quad (\forall u\ v\ w.\ f\ u\ (f\ v\ w) = f\ (f\ u\ v)\ w) \\ & & \textbf{then if}\ a\ \texttt{In}\ A\ \textbf{then}\ \texttt{Fold}\ f\ g\ z\ A \\ & & \qquad\qquad \textbf{else}\ f\ (g\ a)\ (\texttt{Fold}\ f\ g\ z\ A) \\ & & \textbf{else}\ z \end{array}$$

$$A = B \Leftrightarrow \forall a.\ a\ \texttt{In}\ A \Leftrightarrow a\ \texttt{In}\ B$$

$$\forall P.\ P\ \texttt{Empty} \wedge (\forall t.\ P\ t \Rightarrow \forall h.\ P\ (h\ \texttt{Insert}\ t)) \Rightarrow \forall l.\ P\ l$$

The **if** ... **then** ... **else** in the `Delete` definition now yields a finite set, not a list. The last theorem requires higher order quotients to lift, because it involves quantification over functions, in this case P of type `'a finite_set` $\rightarrow$ `bool`.

9 Conclusions

We have presented a design for mechanically creating higher order quotients which is a conservative, definitional extension of higher order logic. The package implemented from this design [10] automatically lifts not only types, but also constants and theorems from the original level to the quotient level.

The relationship between the lower type and the quotient type is characterized by the partial equivalence relation, the abstraction function, and the representation function. As a key contribution, three necessary algebraic properties have been identified for these to properly describe a quotient, which are preserved in the creation of both aggregate and higher order quotients.

The Axiom of Choice was used in this design. We showed that an alternative design may be constructed without dependence on the Axiom of Choice, but that it may not be extended to higher order quotients while remaining constructive.

Prior to this work, only Harrison [8] went beyond support for modeling the quotient types to provide automation for the lifting of constant definitions and theorems from their original statements concerning the original types to the corresponding analogous statements concerning the new quotient types. This is important for the practical application of quotients to sizable problems like quotients on the syntax of complex, realistic programming or specification languages. These may be modelled as recursive types, where terms which are partially equivalent by being well-typed and alpha-equivalent are identified by taking quotients. This eases the traditional problem of the capture of bound variables [7].

Such quotients may now be more easily and practically modeled within a variety of theorem provers, using the design described here.

Soli Deo Gloria.

References

1. Barendregt, H.P.: *The Lambda Calculus, Syntax and Semantics.* North-Holland, 1981.
2. Bruce, K., Mitchell, J. C.: 'PER models of subtyping, recursive types and higher-order polymorphism', in *Principles of Programming Languages 19*, Albequerque, New Mexico, 1992, pp. 316-327.
3. Chicli, L., Pottier, L., Simpson, C.: 'Mathematical Quotients and Quotient Types in Coq', Proceedings of *TYPES 2002*, Lecture Notes in Computer Science, vol. 2646 (Springer-Verlag, 2002).
4. Enderton, H. B.: *Elements of Set Theory.* Academic Press, 1977.
5. Geuvers, H., Pollack, R., Wiekijk, F., Zwanenburg, J.: 'A constructive algebraic hierarchy in Coq', in *Journal of Symbolic Computation*, 34(4), 2002, pp. 271-286.
6. Gordon, M. J. C., Melham, T. F.: *Introduction to HOL.* Cambridge University Press, Cambridge, 1993.
7. Gordon, A. D., Melham, T. F.: 'Five Axioms of Alpha Conversion', in *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, edited by J. von Wright, J. Grundy and J. Harrison, Lecture Notes in Computer Science, vol. 1125 (Springer-Verlag, 1996), pp. 173-190.
8. Harrison, J.: *Theorem Proving with the Real Numbers,* §2.11, pp. 33-37. Springer-Verlag 1998.
9. Hofmann, M.: 'A simple model for quotient types,' in *Typed Lambda Calculus and Applications*, Lecture Notes in Computer Science, vol. 902 (Springer-Verlag, 1995), pp. 216-234.
10. Homeier, P. V.: 'Higher Order Quotients in Higher Order Logic.' In preparation; draft available at `http://www.cis.upenn.edu/~hol/quotients`.
11. Kalker, T.: at `www.ftp.cl.cam.ac.uk/ftp/hvg/info-hol-archive/00xx/0082`.
12. Leisenring, A. C.: *Mathematical Logic and Hilbert's ε-Symbol.* Gordon and Breach, 1969.
13. Moore, G. H.: *Zermelo's Axiom of Choice: It's Origins, Development, and Influence.* Springer-Verlag 1982.
14. Nipkow, T., Paulson, L. C., Wenzel, M.: *Isabelle/HOL.* Springer-Verlag 2002.
15. Owre, S., Shankar, N.: *Theory Interpretations in PVS*, Technical Report SRI-CSL-01-01, Computer Science Lab., SRI International, Menlo Park, CA, April 2001.
16. Nogin, A.: 'Quotient Types: A Modular Approach,' in *Theorem Proving in Higher Order Logics: 15th International Conference, TPHOLs 2002*, edited by V. A. Carreño, C. Muñoz, and S. Tahar, Lecture Notes in Computer Science, vol. 2410 (Springer-Verlag, 2002), pp. 263-280.
17. Paulson, L.: 'Defining Functions on Equivalence Classes,' *ACM Transactions on Computational Logic*, in press. Previously issued as Report, Computer Lab, University of Cambridge, April 20, 2004.
18. Robinson, E.: 'How Complete is PER?', in *Fourth Annual Symposium on Logic in Computer Science* (LICS), 1989, pp. 106-111.
19. Slotosch, O.: 'Higher Order Quotients and their Implementation in Isabelle HOL', in *Theorem Proving in Higher Order Logics: 10th International Conference, TPHOLs'97*, edited by Elsa L. Gunter and Amy Felty, Lecture Notes in Computer Science, vol. 1275 (Springer-Verlag, 1997), pp. 291-306.