Higher Order Quotients in Higher Order Logic

Peter V. Homeier

U. S. Department of Defense, homeier@saul.cis.upenn.edu
http://www.cis.upenn.edu/~homeier

Abstract. The quotient operation is a standard feature of set theory, where a set is partitioned into subsets by an equivalence relation. We rein-

3 Quotient Types

Lemma 6 ($_c$

Theorem 11.

These operators are defined as indicated below in `quotientTheory`, a theory in this package.

Definition 12. LIST_REL R [] [] = true
LIST_REL R $(a::as)$ [] = false
LIST_REL R [] $(b::bs)$ = false
LIST_REL R $(a::as)$ $(b::bs)$ = [illegible]

A theorem of this form is called a *quotient theorem*.

As will be shown in section 5.2, these three properties support the inference of a quotient theorem for a function type, given the quotient theorems for the domain and the range. This inference is necessary to enable higher order quotients.

5.1 Aggregate Quotient Theorems

Quotient theorems are used to control the process of defining lifted constants and lifting theorems. A quotient theorem is needed for each type involved in a

are statements that the constituent subtypes of the type operator are quotients, and a consequent that states that the extension over the type operator is a quotient. Each antecedent has the form of a quotient theorem, but the types involved are simple type variables in the HOL logic.

|- R1 (abs1:α_1->β_1) rep1. QUOTIENT R1 abs1 rep1 ...

Proof: We need to prove the three properties of definition 16:

Property 1. Prove for all a, $(rep_1$ --> abs

Subgoal 2.3.

Theorem 28 (Existence of Hilbert choice). *There exists an operator* $c : (\alpha \to \texttt{bool}) \to \alpha$

A similar function, `define_`

Evaluating `define_quotient_types` with the argument field `types` containing the record {`name="`*tyname*`", equiv=R_EQUIV`} automatically declares a new type *tyname* in the HOL logic as the quotient type

10 Lifting Theorems of Properties

Previously we have seen how to lift types, and how to lift constants on those

$$\begin{array}{l}
\texttt{|-}\ \ (\texttt{x1}:y_1)\ (\texttt{y1}:y_1)\ \ldots\ (\texttt{xn}:y_n)\ (\texttt{yn}:y_n).\\
\qquad\quad R_1\ \texttt{x1 y1}\quad \ldots\quad R_n\ \texttt{xn yn}\\
\qquad\quad R_c\ (\texttt{C x1}\ \ldots\ \texttt{xn})\ (\texttt{C y1}\ \ldots\ \texttt{yn})
\end{array}$$

where the constant `C` has type y_1 `-> ... ->` y_n `->` y_c, and each relation R_i has type y_i `->` y_i `-> bool` for all i. Depending on the types involved, the partial equivalence relations R_1, ..., R_n, R_c

```
|-  t1 t2. ALPHA t1 t2    (HEIGHT1 t1 = HEIGHT1 t2)
```

HEIGHT1 has one argument, which has type term1, so the partial equivalence relation of the argument is ALPHA. The result type is num, which is not being lifted, so the partial equivalence relation for the result is simple equality.

The function FV1 : term1 -> (var -> bool) is used to calculate the set

5. $abs_i : \gamma_i$ `->` δ_i and $rep_i : \delta_i$ `->` γ_i are the (possibly identity, aggregate, or higher-order) abstraction and representation functions for the type γ_i, for all $i = 1, \ldots, k$, *c.hese are expressions, not simply an* `absi` or `repi` variable.

Depending on the types involved, some of the abstraction or representation functions may be the identity function `I`, in which case they disappear, as illustrated in some of the examples below.

The preservation theorem for the operator o is

```
 |-  R1 (abs1:'a -> 'd) rep1. QUOTIENT R1 abs1 rep1
     R2 (abs2:'b -> 'e) rep2. QUOTIENT R2 abs2 rep2
     R3 (abs3:'c -> 'f) rep3. QUOTIENT R3 abs3 rep3
       f g. f o g =
        (rep1 --> abs3) ((abs2 --> rep3) f o (abs1 --> rep2) g)
```

The operator o has type ('b -> 'c) -> ('a -> 'b) -> ('a -> 'c), which is polymorphic and also higher order. It has three type variabl2/F34fe proven by the user, using the same approach as for the example theorems above, as shown in `quotientScript.sml`.

Whenever there are arguments to the constant, there are multiple equivalent ways to state the preservation theorem. For example, the consequent of the preservation theorem for `o` may be given equally well as any of the following completely equivalent versions:

```
 f g x. (f o g) x =
```

TABLE 1.
Preservation and Respectfulness Theorems for Polymorphic Operators

Lifted Operators	Preservation Theorems	Respectfulness Theorems
_::_._ → _._	FORALL_PRS	RES_FORALL_RSP
_::_._ → _._	EXISTS_PRS	RES_EXISTS_RSP
!!_::_._ → !_._	EXISTS_UNIQUE_PRS	RES_EXISTS_EQUIV_RSP
λ_::_._ → _.	_	RES_ _

3. the type of C is of the form y_1

The operator FST has polymorphic type `('a # 'b) -> 'a`. It has two type variables, `'a` and `'b`, so n = 2. as has one argument, so

10.4 Theorems to be lifted: `old_thms`

But this is not true of all functions of the type `term1 -> bool`. For example, consider the particular function `P1` defined by structural recursion as

(
(P1 u)
(

Then `P1 t` is true if and onl

form349.963,Tf11.872...

the respectfulness principle since, for example,349.963Tf11.872204.607(P1)-52ALPHA (Lam1 ”a” (Var1 ”a”))

```
    hom (App1 t u) = app (hom t) (hom u))
(
```

(

```
OVAR1     :   var -> obj1
OBJ1      :   (string # method1) list -> obj1
INVOKE1   :   obj1 -> string -> obj1
UPDATE1   :   obj1 -> string -> method1 -> obj1
SIGMA1    :   var -> obj1 -> method1
```

It also creates associated theorems for induction, function existance, and one-to-one and distinctiveness properties of the constructors.

The definition above goes beyond simple mutual recursion of types, to involve what is called “nested recursion,” where a type being defined may appear deeply

this lifted theorem took considerable automation, hidden behind the simplicity of the result. In addition, the original theorem was not regular; before lifting, it actually was first quietly converted to:

and higher order quotients. Most normal polymorphic operators both respect and are preserved across such quotients, including higher order quotients.

Quotients are useful in a variety of contexts, as shown in the literature. For example, the syntax of programming languages may be modelled as recursive types. Terms which are alpha-equivalent may be identified by taking quotients. This eases the problem of capture of bound variables.

Some systems are convenient to model initially at one level of granularity,

16. Paulson, L.: ‘Defining Func(‘)(1Ans)-424(on)-423(E)-1(quiv)57(al)1(e)-1(nc(‘)e)-424(C)-1(l)1(ass)-1(es,’)]TJ/F
ComprderScevienL