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. Thequotient operationis astandard featureofsettheory,

3 Equivalence Relations and Equivalence Theorems

Definition 1.

$$
\begin{array}{llll}
\texttt{LIST\_REL}\ R & [\,] & [\,] & = \ \textsf{true} \\
\texttt{LIST\_REL}\ R\ (a\texttt{::}as) & [\,] & & = \ \textsf{false} \\
\texttt{LIST\_REL}\ R & [\,] & (b\texttt{::}bs) & = \ \textsf{false} \\
\texttt{LIST\_REL}\ R\ (a\texttt{::}as) & (b\texttt{::}bs) & & = \ R\ a\ b \quad \texttt{LIST\_REL}\ R\ as\ bs
\end{array}
$$

Definition 2. $(R_1 \ \texttt{\#\#\#}\ R_2)\ (a_1, a_2)\ (b_1, b_2) \ = \ R_1\ a_1\ b_1 \quad R_2\ a_2\ b_2$

Definition 3.

$$
\begin{array}{ll}
(R_1 \ \texttt{+++}\ R_2)\ (\texttt{INL}\ a_1)\ (\texttt{INL}\ b_1) & = \ R_1\ a_1\ b_1 \\
(R_1 \ \texttt{+++}\ R_2)\ (\texttt{INL}\ a_1)\ (\texttt{INR}\ b_2) & = \ \textsf{false} \\
(R_1 \ \texttt{+++}\ R_2)\ (\texttt{INR}\ a_2)\ (\texttt{INL}\ b_1) & = \ \textsf{false} \\
(R_1 \ \texttt{+++}\ R_2)\ (\texttt{INR}\ a_2)\ (\texttt{INR}\ b_2) & = \ R_2\ a_2\ b_2
\end{array}
$$

Definition 4. OPTION

$$
\begin{array}{llll}
\texttt{\_REL}\ R & \texttt{NONE} & \texttt{NONE} & = \ \textsf{true} \\
\texttt{OPTION\_REL}\ R\ (\texttt{SOME}\ a) & \texttt{NONE} & & = \ \textsf{false} \\
\texttt{OPTION\_REL}\ R & \texttt{NONE} & (&
\end{array}
$$

Given the type operator $(\alpha_1, ..., \alpha_n)op$, `OP_REL` should be an operator which takes n arguments, which are the equivalence relations `E1` through `En` on the types α_1 through α_n, yielding an equivalence relation for the type $(\alpha_1, ..., \alpha_n)op$.

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

`LIST_EQUIV:` $E.$ `EQUIV` E

Lemma 12.

look like the original. Hence Definition 5 was intentionally designed to preserve the vital type operator structure.

At times one wishes to not only lift a number of types across a quotient operation, but also lift by extension a number of other types which are dependent

These functions prove and return quotient theorems, of the form

QUOTIENT

Goal 2. ( ) 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 R_1, abs_1, rep_1, we also have $R_1\ x\ x$, $R_1\ y\ y$, and $abs_1\ x = abs_1\ y$. By Property 3 of R_2, abs_2, rep_2, the goal is $R_2\ (r\ x)\ (r\ x) \quad R_2\ (s\ y)\ (s\ y) \quad abs_2\ (r\ x) = abs_2\ (s\ y)$. This breaks into three subgoals.

Subgoal 2.1. Prove $R_2\ (r\ x)\ (r\ x$

Thus it is of interest to determine if a design for higher order quotients may

Proof:

8 Lifting Types, Constants, and Theorems

The definition of new types corresponding to the quotients of existing types by equivalence relations is called “lifting” the types from a lower, more representa-

def_name is the name under which the new constant definition is to be stored in the current theory. The process of defining lifted constants is described in §10.

tyop_equivs is a list of equivalence extension theorems for type operators (see §3.1). These are used for bringing into regular form theorems on new type operators, so that they can be lifted (see sections 12 and 13).

tyop_quotients is a list of quotient extension theorems for type operators (see §6.2). These are used for lifting both constants and theorems.

*tyop_*

Furthermore, two more related functions, `define_quotient_types_std` and `define_quotient_types_std_rule`, are provided. These are the same as the

the quotient type as a new type in the HOL logic. It also defines the mapping functions between the types, forming a quotient as described in theorem 16.

All definitions are accomplished as definitional extensions of HOL, and thus preserve HOL's consistency.

Before invoking `define_quotient_types`, the user should define a relation on the original type τ

Please note how the antecedents of each theorem relate to the arguments of the function. The arguments which have types not being lifted are compared between the value of the function applied to arguments of lifted types, and the lifted version of the value of the same function applied to arguments of the lower types. The equalities are conditioned on the component types being quotients.

These preservation theorems have the following general form.

$$
\begin{array}{l}
\quad \texttt{R1 (abs1:}\alpha_1\texttt{->}\beta_1\texttt{) rep1. QUOTIENT R1 abs1 rep1} \\
\texttt{...} \\
\quad \texttt{Rn (absn:}\alpha_n\texttt{->}\beta_n\texttt{) repn. QUOTIENT Rn absn repn} \\
\quad \texttt{(x1:}\delta_1\texttt{) ... (xk:}\delta_k\texttt{).} \\
\qquad\qquad \texttt{C' x1 ... xk = } abs_c \texttt{ (C (}rep_1\texttt{ x1) ... (}rep_k\texttt{ xk))}
\end{array}
$$

where

1. the constant C has a polymorphic

The operator `LENGTH` has polymorphic type `('a)list -> num`. It has one type variable, `'a`, so $n = 1$. It has one argument, so $k = 1$. The argument type is `('a)list`, for which the representation function is `MAP rep`. The result type is `num`, for which the abstraction function is `I`, which disappears.

The preservation theorem for `CONS` is

```
R (abs:'a -> 'b) rep. QUOTIENT R abs rep
  t h. h::t = MAP abs (rep h::MAP rep t)
```

The operator `CONS` has polymorphic type `'a -> ('a)list -> ('a)list`. It has one type variable, `'a`, so $n = 1$. It has two arguments, so $k = 2$. The first argument type is

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_::_._ → λ_._	ABSTRACT_PRS	

11.3 Respectfulness of polymorphic functions: poly_respects

poly_respects

The operator `LENGTH` has polymorphic type `('a)list -> num`. It has one type variable, `'a`, so $n = 1$. It has one argument, so $k = 1$. The argument type is `('a)list`, for which the partial equivalence relation is `LIST_REd[(t)28(yp)-Rt 83]TJ/F899Tf1.30Tdd[(` type is `num`, for which the partial equivalence relation is `=`.

The respectfulness theorem for `CONS` is

```
R (abs:'a -> 'b) rep. QUOTIENT R abs)
```

```
...  f1 f2 g1 g2 x1 x2.
      (R2 ===> R3) f1 f2   (R1 ===> R2) g1 g2   R1 x1 x2
         R3 ((f1 o g1) x1) ((f2 o g2) x2)
...  f1 f2 g1 g2. (R2 ===> R3) f1 f2   (R1 ===> R2) g1 g2
          (R1 ===> R3) (f1 o g1) (f2 o g2)
...  f1 f2. (R2 ===> R3) f1 f2
          ((R1 ===> R2) ===> (R1 ===> R3)) ($o f1) ($o f2)
... ((R2 ===> R3) ===> (R1 ===> R2) ===> (R1 ===> R3)) $o $o
```

The last version has higher order and lesser arity than t(an9R3))-57Hier versions.3) type. This means that theorems that mention the equality of elements of the

with such restricted quantifications from a user-supplied theorem with normal quantification in those places.

In addition, theorems which are universal quantifications at the outer level of the theorem, may imply the restricted universal quantifications (their proper form) over respectful values. The proper form is automatically proven and then lifted. For example, the tool can lift the lambda calculus induction theorem given mentioned in the types argument, or an extension of these, using the relation operators mentioned in sections 3 and 4. If the type of any constant in any theorem involves typ the 8-342o5luotsss8-342o5d[(th)n-342o5d[(th)-342o5aspy th

We specify the respectfulness theorems to assist the lifting:

```
val respects =
    [OVAR1_RSP, OBJ1_RSP, INVOKE1_RSP, UPDATE1_RSP, SIGMA1_RSP,
     HEIGHT_obj1_RSP, HEIGHT_dict1_RSP, HEIGHT_entry1_RSP,
     HEIGHT_method1_RSP, FV_obj1_RSP, FV_dict1_RSP, FV_entry1_RSP,
     FV_method1_RSP, SUB1_RSP, FV_subst_RSP, vsubst1_RSP,
     SUBo_RSP, SUBd_RSP, SUBe_RSP, SUBm_RSP,
     ... ]
```

We specify the polymorphic preservation theorems to assist the lifting:

```
val polyprs = [BV_subst_PRS, COND_PRS, CONS_PRS, NIL_PRS,
               COMMA_PRS, FST_PRS, SND_PRS,
               LET_PRS, o_PRS, UNCURRY_PRS,
               FORALL_PRS, EXISTS_PRS,
               EXISTS_UNIQUE_PRS, ABSTRACT_PRS];
```

We specify the polymorphic respectfulness theorems to assist the lifting:

```
val polyrsp = [BV_subst_RSP, COND_RSP, CONS_RSP, NIL_RSP,
               COMMA_RSP, FST_RSP, SND_RSP,
               LET_RSP, o_RSP, UNCURRY_RSP,
               RES_FORALL_RSP, RES_EXISTS_RSP,
               RES_EXISTS_EQUIV_RSP, RES_ABSTRACT_RSP];
```

The old theorems to be lifted are too many to list, but some will be shown later.

```
- val old_thms = [...];
```

We now define the pure sigma calculus types obj and method, and lift all constants and theorems:

```
- val new_thms =
define_quotient_types
   {types = [{name = "obj",    equiv = ALPHA_obj_EQUIV},
             {name = "method", equiv = ALPHA_method_EQUIV}],
    defs = defs,
    tyop_equivs = [LIST_EQUIV, PAIR_EQUIV],
    tyop_quotients = [LIST_QUOTIENT, PAIR_QUOTIENT,
                      FUN_QUOTIENT],
    tyop_simps = [LIST_REL_EQ, LIST_MAP_I,
                  PAIR_REL_EQ, PAIR_MAP_I,
                   FUN_REL_EQ,  FUN_MAP_I],
    respects = respects,
    poly_preserves = polyprs,
    poly_respects = polyrsp,
    old_thms = old_thms};
```

One of the most difficult theorems to lift is the induction theorem, because

9. Harrison, J.: *Theorem Proving with the Real Numbers,* §2.11, pp. 33-37. Springer-Verlag 1998.