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

Definition 12.

$$
\begin{array}{lllcl}
\mathrm{LIST\_REL}\ R & [\,] & [\,] & = & \mathrm{true} \\
\mathrm{LIST\_REL}\ R & (a::as) & [\,] & = & \mathrm{false} \\
\mathrm{LIST\_REL}\ R & [\,] & (b::bs) & = & \mathrm{false} \\
\mathrm{LIST\_REL}\ R & (a::as) & (b::bs) & = & R\ a\ b\ \wedge \mathrm{LIST\_REL}\ R\ as\ bs
\end{array}
$$

Definition 13. $(R_1\ \#\#\#\ R_2)\ (a_1, a_2)\ (b_1, b_2) \ = \ R_1\ a_1\ b_1\ \wedge\ R_2 b \wedge$

Why use partial equivalence relations with a weaker re exivity condition? The reason involRes forming quotients of higher order types, that is, functions whose domains or ranges involRe types being lifted. Unlike lists and pairs, the equivalence relations forthe

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 in7

These

De nition 18. $\mathtt{MAP}\ \mathbf{f}\ \ []\ =\ []$
$\mathtt{MAP}\ \mathbf{f}\ (\mathbf{a} :: \mathbf{as})\ =\ (\mathbf{f}\ \mathbf{a}) :: (\mathtt{MAP}\ \mathbf{f}\ \mathbf{as})$

De nition 19. $(\mathbf{f}_1\ \#\#\ \mathbf{f}_2)\ (\mathbf{a}_1;\ \mathbf{a}_2)\ =\ (\mathbf{f}_1\ \mathbf{a}_1;\ \mathbf{f}_2\ \mathbf{a}_2)$

De nition 20. $(\mathbf{f}_1\ {+}{+}\ \mathbf{f}_2)\ (\mathtt{INL}\ \mathbf{a}_1)\ =\ \mathtt{INL}\ (\mathbf{f}_1\ \mathbf{a}_1)$
$(\mathbf{f}_1\ {+}{+}\ \mathbf{f}_2)\ (\mathtt{INR}\ \mathbf{a}_2)\ =\ \mathtt{INR}\ (\mathbf{f}_2\ \mathbf{a}_2)$

De nition 21. $\mathtt{OPTION\_MAP}\ \mathbf{f}\ \ \mathtt{NONE}\ =\ \mathtt{NONE}$
$\mathtt{\_MAP}\ \mathbf{f}\ (\mathtt{SOME}\ \mathbf{a})\ =\ \mathtt{SOME}\ (\mathbf{f}$

are statements that the constituent subtypes of the type operator are quotients, and a consequent that

Property (1) states that for every abstract element a of qty there is a representative

Theorem 28 (Existence of Hilbert cHigher)Orde0.4683428 0 Td(Ord9.09983428 0 Td56 0 (in)2)Tj19ce

A similar function, `define_quotient_types_rule`, takes a single argument which is a record with the same elds as above except for old_thms, and returns an SML function of type `thm -> thm`. This result, typically called `LIFT_RULE`, is then used to lift the old theorems individually, one at a time.

For backwards compatibility with John Harrison's excellent quotients package [9] (which provided much inspiration), the following function is also provided:

```
define_equivalence_type :
        {name: string,
         equiv: thm,
         defs: {def_name: string,
                fname: string,
                func: Term.term,
                fixity: Parse.fixity} list,
         welldefs : thm list,
         old_thms : thm list} ->
```

Evaluating define_quotient_types **with the**

|- **8**(x1: $_1$) (y1: $_1$) ... (xn: $_n$) (yn: $_n$).
    R_1 x1 y1 **^** ... **^** R_n xn yn **)**
    R_c (C x1 ... xn) (C y1 ... yn)

where the constant C **has type** $_1$ -> ... -> $_n$ -> $_c$**, and each relation R_i has type** $_i$ -> $_i$ -> bool **for all i. Depending on the types involved, the partial**

```
|- 8t1 t2. ALPHA t1 t2
```

The

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

alention 26.641980 Td (27)orem

the

TABLE 1.
Preservation and Respectfulness Theorems for Polymorphic Operators

Lifted Operators	Preservation Theorems	Respectfulness Theorems
8_: : _. _ ! 8_		

The operator FST **has polymorphic type** `('a # 'b) -> 'a`**. It has two type variables,** `'a` **and** `'b`**, so n = 2. It has one argument, so k = 1. The argument 'a and**

10.4 Theorems to be lifted: `old_thms`

`old_thms` **are the theorems to be lifted. They should involve only constants**

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

```
(8x.   P1 (Var1 x)   = F) ^
(8t u. P1 (App1 t u) = P1 t _ P1 u) ^
(8x u. P1 (Lam1 x u) = P1 u _ (x = "a"))
```

Then `P1 t` **is true if and only if the term** `t` **contains a subexpression of the**

Finally

```
(8x. hom (Var1 x) = var x) ^
(8t u :: respects ALPHA.
   hom (App1 t u) = app (hom t) (hom u)) ^
(8x (u :: respects ALPHA).
   hom (Lam1 x u) = abs ( y. hom (u <[ [(x,Var1 y)])))
```

and thed lift the proper versiod to the desired higher versiod of this theorem:

```
|- 8var app abs.
     9!hom.
      (8x. hom (Var x) = var x) ^
      (8t u. hom (App t u) = app (hom t) (hom u)) ^
      (8x u. hom (Lam x u) = abs ( y. hom (u <[ [(x,Var y)])))
```

In general, of course, this revising may not work. If the tool7997 0 .40 Td (^717(not)Tj 3217401 0 Td autor

```
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

```
 func= (--`INVOKE1`--), fixity=Prefix},
{def_name="UPDATE_def", fname="UPDATE",
 func= (--`UPDATE1`--), fixity=Prefix},
{def_name="SIGMA_def", fname="SIGMA",
 func= (--`SIGMA1`--), fixity=Prefix},
{def_name="HEIGHT_def", fname="HEIGHT",
 func= (--`HEIGHT1`--), fixity=Prefix},
{def_name="FV_def", fname="FV",
 func= (--`FV1`--), fixity=Prefix},
{def_name="SUB_def", fname="SUB",
 func= (--`SUB1`--), fixity=Prefix},
{def_name="FV_subst_def", fname="FV_subst",
 func= (--`FV_subst1`--), fixity=Prefix},
{def_name="SUBo_def",
```

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 rst quietly converted to:

|- (

```
|- 8P0 P1 P2 P3.
     (8v. P0 (OVAR v)) ^ (8l. P2 l ) P0 (OBJ l)) ^
     (8o'. P0 o' ) 8s. P0 (INVOKE o' s)) ^
     (8o' m. P0 o' ^ P1 m ) 8s. P0 (UPDATE o' s m)) ^
     (8o'. P0 o' ) 8v. P56P 9 896264 Tf -336.54 Tf 596264 Tf -336.=MA(v) .5606788 0 Td(m)))T
```

```
 nil (par::respects($= ===> $= ===> ALPHA_method ===> $=))
(sgm::respects($= ===> ($= ===> ALPHA_obj) ===> $=)).
                                                       (sgm::respects($= ===>
```

```
(par::respects ($= ===> $= ===> ALPHA_method ===> $=))
(sgm::respects ($= ===> ($= ===> ALPHA_obj) ===> $=)).
    9!!(hom_o,hom_d,hom_e,hom_m)::
        (ALPHA_obj ===> $=) ###
        (LIST_REL ($= ### ALPHA_method) ===> $=) ###
        (($= ### ALPHA_method) ===> $=) ###
        (ALPHA_method ===> $=).
      (8x. hom_o (OVAR1 x) = var x) ^
      (8d::respects (LIST_REL ($= ### ALPHA_method)).
         hom_o (OBJ1 d) = obj (hom_d d) d) ^
      (8(a::respectsALPHA obj) l.
         hom_o (INVOKE1 a l) = inv (hom_
```

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