













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

**Definition 12.**  $\text{LIST\_REL } [] \ [] = \text{true}$   
 $\text{LIST\_REL } ( :: ) \ [] = \text{false}$   
 $\text{LIST\_REL } [] \ ( :: ) = \text{false}$   
 $\text{LIST\_REL } ( :: ) \ ( :: ) = \text{LIST\_REL} \ ^{\wedge} \text{LIST\_REL}$

**Definition 13.**  $( \_1 \ ### \_2 ) ( \_1 \_2 ) ( \_1 \_2 ) = \_1 \_1 \_1 \ ^{\wedge} \_2 \ ^{\wedge}$





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

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

**Definition 18.**  $\text{MAP } [] = []$   
 $\text{MAP } (x :: xs) = (f x) :: (\text{MAP } f xs)$

**Definition 19.**  $(x_1 \# x_2) (f_1 f_2) = (f_1 x_1 f_2 x_2)$

**Definition 20.**  $(x_1 ++ x_2) (\text{INL } f_1) = \text{INL } (f_1 x_1)$   
 $(x_1 ++ x_2) (\text{INR } f_2) = \text{INR } (f_2 x_2)$

**Definition 21.**  $\text{OPTION\_MAP } \text{NONE} = \text{NONE}$   
 $\text{OPTION\_MAP } (\text{SOME } x) = \text{SOME } (f x)$

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





Property (1) states that for every abstract element of  $q\tau y$  there is a representative



Higher Order 0.4683428 0 Td (Ord 9.09983428 0 Td 56 0 (in) 2) Tj 19 ce

```
    tyop_quotients : thc list,  
    tyop_simps : thc list,  
    respects : thc list,  
    poly_preserves : thc list,  
    poly_respects : thc list,  
    old_thcs : thc list} ->  
thc list
```

```
define_quotient
```

A similar function, `define_quotient_types_rule`, takes a single argument which is a record with the same fields 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**



$$| - \mathbf{8}(x1: \tau_1) (y1: \tau_1) \dots (xn: \tau_n) (yn: \tau_n). \\ \tau_1 \ x1 \ y1 \ \wedge \ \dots \ \wedge \ \tau_n \ xn \ yn \ ) \\ c \ (C \ x1 \ \dots \ xn) \ (C \ y1 \ \dots \ yn)$$

where the constant  $C$  has type  $\tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow c$ , and each relation  $\tau_i$  has type  $\tau_i \rightarrow \tau_i \rightarrow \text{bool}$  for all  $i$ . Depending on the types involved, the partial

| -  $\exists t_1 t_2. \text{ALPHA } t_1 t_2$







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)oren

the

TABLE 1

Relationships between the lifted operators and the original operators

Lifted Operators	Preservation Theorems	Respectfulness Theorems
$\forall_{\perp} : \text{---} \rightarrow \forall_{\perp}$		



The operator `FST` has polymorphic type  $('a \# 'b) \rightarrow 'a$ . It has two type variables, `'a` and `'b`, so `arity` = 2. It has one argument, so `arity` = 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` defined by structural recursion as

$$\begin{aligned} (\lambda x. \quad & P1 \ (Var1 \ x) \quad = F) \wedge \\ (\lambda t \ u. \quad & P1 \ (App1 \ t \ u) = P1 \ t \ \_ \ P1 \ u) \wedge \\ (\lambda x \ u. \quad & P1 \ (Lam1 \ x \ u) = P1 \ u \ \_ \ (x = "a")) \end{aligned}$$

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 then lift the proper version to the desired higher version 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 tool



```
OVAR1      : var -> obj 1
OBJ1       : (string # method1) list -> obj 1
INVOKE1    : obj 1 -> string -> obj 1
UPDATE1    : obj 1 -> string -> method1 -> obj 1
SIGMA1     : var -> obj 1 -> method1
```

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

```

func= (--`INVOKE1`--), fi xi ty=Prefi x},
{def_name="UPDATE_def", fname="UPDATE",
func= (--`UPDATE1`--), fi xi ty=Prefi x},
{def_name="SIGMA_def", fname="SIGMA",
func= (--`SIGMA1`--), fi xi ty=Prefi x},
{def_name="HEIGHT_def", fname="HEIGHT",
func= (--`HEIGHT1`--), fi xi ty=Prefi x},
{def_name="FV_def", fname="FV",
func= (--`FV1`--), fi xi ty=Prefi x},
{def_name="SUB_def", fname="SUB",
func= (--`SUB1`--), fi xi ty=Prefi x},
{def_name="FV_subst_def", fname="FV_subst",
func= (--`FV_subst1`--), fi xi ty=Prefi x},
{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 first quietly converted to:

| - (



| -  $\exists P_0 P_1 P_2 P_3.$

$(\exists v. P_0 (OVAR\ v)) \wedge (\exists l. P_2\ l) \wedge P_0 (OBJ\ l)) \wedge$

$(\exists o'. P_0\ o') \wedge \exists s. P_0 (INVOKE\ o'\ s)) \wedge$

$(\exists o' m. P_0\ o' \wedge P_1\ m) \wedge \exists s. P_0 (UPDATE\ o'\ s\ m)) \wedge$

$(\exists o'. P_0\ o') \wedge \exists v. P_5\ 9\ 596264\ Tf\ -336.54\ Tf\ 596264\ Tf\ -336. =MA(v) .5606788\ 0\ Td(m)))T$

```

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 ==> $=).
(8x. hom_o (OVAR1 x) = var x) ^
(8d. list_respects _REL ($= 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

