Equivalences, Quotients, Universal Algebra and Lattice Theory

Roger Bishop Jones

Abstract

This is a limited development of universal algebra and lattice theory for the purposes of X-Logic.

Created 2010/07/20

Last Change Date: 2010/08/08 15:50:44

http://www.rbjones.com/rbjpub/pp/doc/t039.pdf

Id: t039a.tex,v 1.3 2010/08/08 15:50:44 rbj Exp

Contents

1 Prelude

This document is intended possibly to form a chapter of *Analyses of Analysis* [1], depending on how comprehensive I decide to make that work.

Some elementary Lattice theory is needed for the lattice of trust or assurance at the heart of "X-Logic", The Universal Algebra is an attempt to do once some basic theory which should be applicable both in Lattice theory and in other algebraic theories. It is a matter of interest to discover whether this level of abstractions pays off in the development of algebraic theories (in terms of reduced overall complexity and cost).

Further discussion of what might become of this document in the future may be found the postscript (Section 8).

In this document, phrases in coloured text are hyperlinks, like on a web page, which will usually get you to another part of this document (the blue parts, the contents list, page numbers in the Index) but sometimes take you (the red bits) somewhere altogether different (if you happen to be online) like online copy of this document.

2 Introduction

3 Equivalence Relations

SML

```
open_theory "rbjmisc";
force_new_theory "equiv";
force_new_pc "'equiv";
merge_pcs ["'prove_∃_⇒_conv", "'savedthm_cs_∃_proof"] "'equiv";
set_merge_pcs ["rbjmisc", "'equiv"];
```

SML

```
declare_infix(230, "≡");
declare_infix(230, "≡ₗ");
declare_infix(230, "≡ᵣ");
```

It will be convenient to name an uncurried version of *QuotientSet*:

HOL Constant

$$\textbf{\textit{EquivClasses}} : ('a\ SET \times ('a \to 'a \to BOOL)) \to ('a\ SET\ SET)$$

$$EquivClasses = Uncurry\ QuotientSet$$

$$\textbf{\textit{EquivClasses\_thm}} =$$
$$\vdash EquivClasses\ (X, \$\equiv) = \{A | \exists\ x \bullet\ x \in X \wedge A = EquivClass\ (X, \$\equiv)\ x\}$$

$$\textbf{\textit{EquivClasses\_thm1}} =$$
$$\vdash \forall\ r \bullet\ EquivClasses\ r = \{A | \exists\ x \bullet\ x \in Fst\ r \wedge A = EquivClass\ r\ x\}$$

$$\textbf{\textit{EquivClasses\_sub\_thm}} =$$
$$\vdash \forall\ (D, \$\equiv) \bullet\ Equiv\ (D, \$\equiv) \Rightarrow (\forall\ l \bullet\ l \in EquivClasses\ (D, \$\equiv) \Rightarrow l \subseteq D)$$

$$\textbf{\textit{EquivClasses\_sub\_thm1}} =$$
$$\vdash \forall\ r \bullet\ Equiv\ r \Rightarrow (\forall\ l \bullet\ l \in EquivClasses\ r \Rightarrow l \subseteq Fst\ r)$$

3.1 Products of Equivalence Relations

I am providing an alternative way of lifting operators from some type to equivalence classes over the type. The treatment of dyadic operators, which in ProofPower have to be curried if they are to be used infix, is done by uncurrying and then lifting as if a monadic function over the cartesian product.

This will lift to the product equivalence classes so we need a treatment of products of equivalence relations.

$$
\begin{array}{|l}
\boldsymbol{Equiv\_RelProd\_thm} = \\
\quad \vdash \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)\bullet \\
\qquad Equiv\ (L,\ \$\equiv_l) \wedge Equiv\ (R,\ \$\equiv_r) \Rightarrow Equiv\ ((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r))
\end{array}
$$

$$
\begin{array}{|l}
\boldsymbol{EquivClass\_RelProd\_thm} = \\
\quad \vdash \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)\bullet\ Equiv\ (L,\ \$\equiv_l) \wedge Equiv\ (R,\ \$\equiv_r) \\
\qquad \Rightarrow (\forall\ xl\ xr\ yl\ yr\bullet\ xl \in L \wedge xr \in R \wedge yl \in L \wedge yr \in R \\
\qquad\quad \Rightarrow ((xl,\ xr) \in EquivClass\ ((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r))\ (yl,\ yr) \\
\qquad\qquad \Leftrightarrow xl \in EquivClass\ (L,\ \$\equiv_l)\ yl \\
\qquad\qquad\quad \wedge xr \in EquivClass\ (R,\ \$\equiv_r)\ yr))
\end{array}
$$

$$
\begin{array}{|l}
\boldsymbol{EquivClass\_RelProd\_thm1} = \\
\quad \vdash \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)\bullet\ Equiv\ (L,\ \$\equiv_l) \wedge Equiv\ (R,\ \$\equiv_r) \\
\qquad \Rightarrow (\forall\ l\ r\bullet\ l \in L \wedge r \in R \\
\qquad\quad \Rightarrow EquivClass\ ((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r))\ (l,\ r) \\
\qquad\qquad = (EquivClass\ (L,\ \$\equiv_l)\ l \times EquivClass\ (R,\ \$\equiv_r)\ r))
\end{array}
$$

$$
\begin{array}{|l}
\boldsymbol{EquivClasses\_RelProd\_thm} = \\
\quad \vdash \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)\bullet\ Equiv\ (L,\ \$\equiv_l) \wedge Equiv\ (R,\ \$\equiv_r) \Rightarrow (\forall\ l\ r\bullet \\
\qquad\quad (l \times r) \in EquivClasses\ ((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r)) \\
\qquad\quad \Leftrightarrow l \in EquivClasses\ (L,\ \$\equiv_l) \wedge r \in EquivClasses\ (R,\ \$\equiv_r))
\end{array}
$$

$$
\begin{array}{|l}
\boldsymbol{EquivClasses\_RelProd\_thm1} = \\
\quad \vdash \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)\bullet\ Equiv\ (L,\ \$\equiv_l) \wedge Equiv\ (R,\ \$\equiv_r) \Rightarrow (\forall\ x\bullet \\
\qquad\quad x \in EquivClasses\ ((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r)) \\
\qquad\quad \Leftrightarrow (\exists\ l\ r\bullet\ x = (l \times r) \\
\qquad\qquad \wedge\ l \in EquivClasses\ (L,\ \$\equiv_l) \\
\qquad\qquad \wedge\ r \in EquivClasses\ (R,\ \$\equiv_r)))
\end{array}
$$

$$
\begin{array}{|l}
\boldsymbol{EquivClasses\_RelProd\_thm2} = \\
\quad \vdash \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)\bullet\ Equiv\ (L,\ \$\equiv_l) \wedge Equiv\ (R,\ \$\equiv_r) \\
\qquad \Rightarrow EquivClasses\ ((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r)) \\
\qquad\quad = EquivClasses\ (L,\ \$\equiv_l) \times_D EquivClasses\ (R,\ \$\equiv_r)
\end{array}
$$

3.2 Powers of Equivalence Relations

RelPower_Equiv_thm =

$\vdash \forall\ (D,\ \$\equiv)\ is\bullet\ Equiv\ (D,\ \$\equiv) \Rightarrow Equiv\ (RelPower\ (D,\ \$\equiv)\ is)$

3.3 Lifting Operators Over Quotients

The conditions for a function to be liftable from a structure to a quotient of that structure are now expressed. This will be possible of the function "respects" the equivalence relations which determine the relevant quotient types.

The definition give here differs from *Respects* in theory *equiv_rel* in expressing conditions for lifting both on the domain and the codomain relative to possibly distinct equivalence relation, and therefore requires that if two elements of the domain are equivalent under one relationship then the results of the function on these two elements will be equivalent under the other relationship.

SML

```
declare_infix(200, "Respects1");
declare_infix(230, "≡d");
declare_infix(230, "≡c");
declare_infix(230, "≡e");
declare_infix(230, "≡f");
```

HOL Constant

$\$\textbf{Respects1} : ('a \to 'b) \to (('a\ SET \times ('a \to 'a \to BOOL)) \times ('b\ SET \times ('b \to 'b \to BOOL))) \to BOO$

$\forall f\ \$\equiv_c\ \$\equiv_d\ C\ D\bullet\ (f\ Respects1\ ((D,\ \$\equiv_d),\ (C,\ \$\equiv_c)))$
$\quad \Leftrightarrow \forall x\ y\bullet x \in D \wedge y \in D \wedge x \equiv_d y \Rightarrow f\ x \in C \wedge f\ y \in C \wedge f\ x \equiv_c f\ y$

Respects1_Respects_thm =

$\vdash \forall f\ \$\equiv_d\ C\ D\bullet\ f\ Respects1\ ((D,\ \$\equiv_d),\ Universe,\ \$=) \Leftrightarrow (f\ Respects\ \$\equiv_d)\ D$

Respects1_Refines_thm =

$\vdash \forall f\ \$\equiv_d \equiv_c \$\equiv_e \equiv_f\ D\ C\bullet$
$\quad f\ Respects1\ ((D,\ \$\equiv_d),\ C,\ \$\equiv_c)$
$\quad\quad \wedge\ (\$\equiv_e\ Refines\ \$\equiv_d)\ D$
$\quad\quad \wedge\ (\$\equiv_c\ Refines\ \$\equiv_f)\ C$
$\quad\quad \Rightarrow f\ Respects1\ ((D,\ \$\equiv_e),\ C,\ \$\equiv_f)$

eq_Refines_thm =

$\vdash \forall\ \$\equiv_d\ D\bullet\ Equiv\ (D,\ \$\equiv_d) \Rightarrow (\$=\ Refines\ \$\equiv_d)\ D$

constant_img_thm1 =

$\vdash \forall f\ A\ a\ c\bullet\ a \in A \wedge (\forall x\bullet\ x \in A \Rightarrow f\ x = c)$
$\quad \Rightarrow (\epsilon\ y\bullet \exists x\bullet\ x \in A \wedge y = f\ x) = c$

The following operator lifts a function over a pair of equivalence relations respected by the function. This definition allows that they domain and codomains are of different type, and will provide the basis for lifting monadic and dyadic operators in a single sorted theory.

SML

```
declare_infix(200, "LiftOver");
```

HOL Constant

$$
\begin{array}{l}
\$\boldsymbol{LiftOver} : ('a \to 'b) \\
\quad \to (('a\ SET \times ('a \to 'a \to BOOL)) \times ('b\ SET \times ('b \to 'b \to BOOL))) \\
\quad \to ('a\ SET \to 'b\ SET) \\
\hline
\forall f\ (D, \$\equiv_d)\ (C, \$\equiv_c)\bullet\ (f\ LiftOver\ ((D, \$\equiv_d), (C, \$\equiv_c))) \\
\quad = \lambda x\bullet\ \epsilon y\bullet\ \exists z\bullet\ z \in x \wedge y = EquivClass\ (C, \$\equiv_c)\ (f\ z)
\end{array}
$$

$$
\begin{array}{l}
\boldsymbol{LiftOver\_thm} = \\
\quad \vdash \forall f\ (D, \$\equiv_d)\ (C, \$\equiv_c)\ de\ ce\bullet \\
\qquad Equiv\ (D, \$\equiv_d) \\
\qquad \wedge\ Equiv\ (C, \$\equiv_c) \\
\qquad \wedge\ f\ Respects1\ ((D, \$\equiv_d), C, \$\equiv_c) \\
\qquad \wedge\ de \in D\ /\ \$\equiv_d \\
\qquad \wedge\ ce \in C\ /\ \$\equiv_c \\
\qquad \Rightarrow ((f\ LiftOver\ ((D, \$\equiv_d), C, \$\equiv_c))\ de = ce \\
\qquad\quad \Leftrightarrow (\exists\ d\ c\bullet\ d \in D \\
\qquad\qquad \wedge\ EquivClass\ (D, \$\equiv_d)\ d = de \\
\qquad\qquad \wedge\ EquivClass\ (C, \$\equiv_c)\ c = ce \\
\qquad\qquad \wedge\ c = f\ d))
\end{array}
$$

3.3.1 Lifting Monadic Operators

SML

```
declare_infix(200, "MonOpRespects");
```

HOL Constant

$$
\begin{array}{l}
\$\boldsymbol{MonOpRespects} : ('a \to 'a) \to ('a\ SET \times ('a \to 'a \to BOOL)) \to BOOL \\
\hline
\forall f\ e\bullet\ f\ MonOpRespects\ e \Leftrightarrow f\ Respects1\ (e, e)
\end{array}
$$

$$
\begin{array}{l}
\boldsymbol{MonOpRespects\_thm} = \\
\quad \vdash \forall f\ C\ \$\equiv\bullet\ f\ MonOpRespects\ (C, \$\equiv) \Leftrightarrow \forall x\ y\bullet x \in C \wedge y \in C \wedge x \equiv y \Rightarrow f\ x \in C \wedge f\ y \in C \wedge f
\end{array}
$$

The following operator lifts a monadic operator over an equivalence relations respected by the operator.

SML

```
declare_infix(210, "MonOpLift");
```

HOL Constant

$$\$\boldsymbol{MonOpLift} : ('a \to 'a) \to ('a\ SET \times ('a \to 'a \to BOOL)) \to ('a\ SET \to 'a\ SET)$$

$$\forall\ f\ (D, \$\equiv_d)\bullet\ f\ MonOpLift\ (D, \$\equiv_d) = f\ LiftOver\ ((D, \$\equiv_d), (D, \$\equiv_d))$$

$$\boldsymbol{MonOpLift\_thm} =$$
$$\vdash \forall\ f\ (D, \$\equiv_d)\ de\ ce$$
$$\bullet\ Equiv\ (D, \$\equiv_d)$$
$$\wedge\ f\ Respects1\ ((D, \$\equiv_d), D, \$\equiv_d)$$
$$\wedge\ de \in D\ /\ \$\equiv_d$$
$$\wedge\ ce \in D\ /\ \$\equiv_d$$
$$\Rightarrow ((f\ MonOpLift\ (D, \$\equiv_d))\ de = ce$$
$$\Leftrightarrow (\exists\ d\ c\bullet\ d \in D$$
$$\wedge\ EquivClass\ (D, \$\equiv_d)\ d = de$$
$$\wedge\ EquivClass\ (D, \$\equiv_d)\ c = ce$$
$$\wedge\ c = f\ d))$$

3.3.2 Lifting Dyadic Operators

SML

```
declare_infix(200, "DyOpRespects");
```

HOL Constant

$$\$\boldsymbol{DyOpRespects} : ('a \to 'a \to 'a) \to ('a\ SET \times ('a \to 'a \to BOOL)) \to BOOL$$

$$\forall\ f\ e\bullet\ f\ DyOpRespects\ e \Leftrightarrow (Uncurry\ f)\ Respects1\ (e\ RelProd\ e, e)$$

$$\boldsymbol{DyOpRespects\_thm} =$$
$$\vdash \forall\ f\ C\ \$\equiv\bullet\ f\ DyOpRespects\ (C, \$\equiv)$$
$$\Leftrightarrow (\forall\ x1\ y1\ x2\ y2\bullet\ x1 \in C \wedge y1 \in C \wedge x2 \in C \wedge y2 \in C \wedge x1 \equiv x2 \wedge y1 \equiv y2$$
$$\Rightarrow f\ x1\ y1 \in C \wedge f\ x2\ y2 \in C \wedge f\ x1\ y1 \equiv f\ x2\ y2)$$

The following operator lifts a dyadic operator over an equivalence relations respected by the operator.

SML

```
declare_infix(210, "DyOpLift");
```

HOL Constant

$$\$\boldsymbol{DyOpLift} : ('a \to 'a \to 'a) \to ('a\ SET \times ('a \to 'a \to BOOL)) \to ('a\ SET \to 'a\ SET \to 'a\ SET)$$

$$\forall\ f\ (D, \$\equiv_d)\bullet\ f\ DyOpLift\ (D, \$\equiv_d)$$
$$= \lambda x\ y\bullet((Uncurry\ f)\ LiftOver\ ((D, \$\equiv_d)\ RelProd\ (D, \$\equiv_d), (D, \$\equiv_d)))\ (x \times y)$$

$$
\begin{array}{l}
\textbf{\textit{DyOpLift\_thm}} = \\
\quad \vdash \forall\ f\ (D, \$\equiv_d)\ l\ r\ c \\
\quad\quad \bullet\ \mathit{Equiv}\ (D, \$\equiv_d) \\
\quad\quad\quad \wedge\ f\ \mathit{DyOpRespects}\ (D, \$\equiv_d) \\
\quad\quad\quad \wedge\ l \in D\ /\ \$\equiv_d \\
\quad\quad\quad \wedge\ r \in D\ /\ \$\equiv_d \\
\quad\quad\quad \wedge\ c \in D\ /\ \$\equiv_d \\
\quad\quad\quad \Rightarrow ((f\ \mathit{DyOpLift}\ (D, \$\equiv_d))\ l\ r = c \\
\quad\quad\quad\quad \Leftrightarrow (\exists\ le\ re\ ce \\
\quad\quad\quad\quad \bullet\ le \in D \\
\quad\quad\quad\quad\quad \wedge\ re \in D \\
\quad\quad\quad\quad\quad \wedge\ \mathit{EquivClass}\ (D, \$\equiv_d)\ le = l \\
\quad\quad\quad\quad\quad \wedge\ \mathit{EquivClass}\ (D, \$\equiv_d)\ re = r \\
\quad\quad\quad\quad\quad \wedge\ \mathit{EquivClass}\ (D, \$\equiv_d)\ ce = c \\
\quad\quad\quad\quad\quad \wedge\ ce = f\ le\ re))
\end{array}
$$

3.3.3 Lifting Generic Operators

For Universal Algebra we will represent all operators whatever their arity using a single type. The domain of the operators is a collection of values indexed by an initial segment of the natural numbers. The operator is represented by a pair of which the first element is the number of arguments and the second is a function which accepts a map from natural numbers into the domain of the algebra and returns a value in the domain.

The values of the argument function for natural numbers equal or greater than its arity is to be ignored, as are values for argument sets which are not confined to the domain of the algebra. When lifting, the domain of the equivalence relation is assumed to be that of the algebra.

SML

$$
\mathit{declare\_type\_abbrev}(\texttt{"}UOP\texttt{"}, [\texttt{"}'a\texttt{"}], \ulcorner :\mathbb{N} \times ((\mathbb{N} \to 'a) \to 'a)\urcorner);
$$

The following function effects the lifting of an operator.

HOL Constant

$$
\begin{array}{l}
\textbf{\textit{UniOpLift}} :\ (('b)SET \times ('b \to 'b \to BOOL)) \to 'b\ UOP \to 'b\ SET\ UOP \\
\hline
\forall\ a\ op\ r \bullet\ \mathit{UniOpLift}\ r\ (a, op) = (a, \lambda f \bullet\ (op\ \mathit{LiftOver}\ (\mathit{RelPower}\ r\ \{x{:}\mathbb{N} \mid x < a\}, r)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{g \mid \forall i \bullet\ i < a \Rightarrow g\ i \in f\ i\})
\end{array}
$$

3.4 Equivalence Closure

We are concerned here with obtaining an equivalence relation by taking the closure of some other relationship.

The presentation is slicker if we define inclusion for the kinds of relationship we are working with here. Since there is some junk in the representation of these relations its not immediately obvious which is the best way to define inclusion.

HOL Constant

$$\textbf{\textit{RelIncl}} : (('b)SET \times ('b \to 'b \to BOOL)) \to (('b)SET \times ('b \to 'b \to BOOL)) \to BOOL$$

$$\forall A\ r1\ B\ r2 \bullet RelIncl\ (A,\ r1)\ (B,\ r2) \Leftrightarrow \forall x\ y \bullet\ x \in A \wedge y \in A \wedge r1\ x\ y \Rightarrow x \in B \wedge y \in B \wedge r2\ x\ y$$

SML

```
declare_alias ("⊆", ⌜RelIncl⌝);
```

The obvious way to do this is to take the intersection of the equivalence relationships which contain the given relationship, and for this purpose we need to prove that such an intersection will be an equivalence relation.

$$\begin{array}{l}\textbf{\textit{Equiv\_Equiv\_closure\_thm}} = \\ \quad \vdash \forall\ A\ r \\ \quad\quad \bullet\ Equiv \\ \quad\quad\quad (A, \\ \quad\quad\quad\quad (\lambda\ x\ y \\ \quad\quad\quad\quad\quad \bullet \forall\ \$\equiv_d \\ \quad\quad\quad\quad\quad\quad \bullet\ Equiv\ (A,\ \$\equiv_d) \wedge (\forall\ v\ w \bullet\ r\ v\ w \Rightarrow v \equiv_d w) \Rightarrow x \equiv_d y))\end{array}$$

We therefore define the equivalence closure of a relation thus:

HOL Constant

$$\textbf{\textit{EquivClosure}} : (('b)SET \times ('b \to 'b \to BOOL)) \to (('b)SET \times ('b \to 'b \to BOOL))$$

$$\forall A\ r \bullet\ \ EquivClosure\ (A,\ r) = (A,\ \lambda x\ y \bullet\ \forall \$\equiv_d \bullet\ Equiv(A,\ \$\equiv_d) \wedge (A,\ r) \subseteq (A,\ \$\equiv_d) \Rightarrow x \equiv_d y)$$

The following are then immediate:

$$\textbf{\textit{Equiv\_EquivClosure\_thm}} = \vdash \forall\ A\ r \bullet\ Equiv\ (EquivClosure\ (A,\ r))$$

$$\begin{array}{l}\textbf{\textit{EquivClosure\_MinEquiv\_thm}} = \\ \quad \vdash \forall\ A\ r\ \$\equiv_d \bullet\ Equiv\ (A,\ \$\equiv_d) \wedge (A,\ r) \subseteq (A,\ \$\equiv_d) \\ \quad\quad \Rightarrow EquivClosure\ (A,\ r) \subseteq (A,\ \$\equiv_d)\end{array}$$

$$\textbf{\textit{Fst\_EquivClosure\_thm}} = \vdash \forall\ eq \bullet\ Fst\ (EquivClosure\ eq) = Fst\ eq$$

4 Universal Algebra

SML

```
open_theory "equiv";
force_new_theory "unalg";
force_new_pc "'unalg";
merge_pcs ["'prove_∃_⇒_conv", "'savedthm_cs_∃_proof"] "'unalg";
set_merge_pcs ["rbjmisc", "'unalg"];
```

To make the operators fairly general within the constrainst imposed by the HOL type system we will have operators as functions over indexed sets of values.

The following labelled product is used as a general notion of "structure" independent of signature. Of course any particular algebra will have a definite signature.

A signature is a string indexed set of arities, where an arity is a natural number.

SML

$declare\_type\_abbrev\ (\texttt{"}SIG\texttt{"},\ [],\ \ulcorner{:}(STRING,\ \mathbb{N})\ IX\urcorner);$

The operators over the algebra are represented by functions from indexed sets of operands to a single result value. In this case we pack the arguments into a total function and ignore the values which do not correspond to the signature. The signature (i.e. the arity of each operator) is explicit in this structure, otherwise the range of significance of the operators would not be known.

HOL Labelled Product

STRUCT

SCar	$: 'a\ SET;$
SOps	$: (STRING, 'a\ UOP)IX$

HOL Constant

$$\boldsymbol{Arity_u} : ('a)\ STRUCT \to STRING \to \mathbb{N}$$

$$\forall A\ n\bullet\ Arity_u\ A\ n = Fst\ (ValueOf\ (SOps\ A\ n))$$

$$\boldsymbol{Arity_u\_lemma} =$$
$$\vdash\ \forall\ d\ l\ n\bullet\ Arity_u\ (MkSTRUCT\ d\ (IxPack\ l))\ n = Fst\ (ValueOf\ (IxPack\ l\ n))$$

HOL Constant

$$\boldsymbol{Oper} : ('a)\ STRUCT \to STRING \to ((\mathbb{N} \to 'a) \to 'a)$$

$$\forall A\ n\bullet\ Oper\ A\ n = Snd\ (ValueOf\ (SOps\ A\ n))$$

$$\boldsymbol{Oper\_lemma} =$$
$$\vdash\ \forall\ d\ l\ n\bullet\ Oper\ (MkSTRUCT\ d\ (IxPack\ l))\ n = Snd\ (ValueOf\ (IxPack\ l\ n))$$

The following function extracts the signature from a structure.

HOL Constant

$$\boldsymbol{Sig} : ('a)\ STRUCT \to SIG$$

$$\forall s\bullet\ Sig\ s = IxCompIx\ (SOps\ s)\ (\lambda x\bullet\ Value\ (Fst\ x))$$

$$\boldsymbol{IxDom\_Sig\_thm} =$$
$$\vdash\ \forall\ S\bullet\ IxDom\ (Sig\ S) = IxDom\ (SOps\ S)$$

$$\in\boldsymbol{\_IxDom\_Sig\_thm} =$$
$$\vdash\ \forall\ S\ x\bullet\ x \in IxDom\ (Sig\ S) \Leftrightarrow x \in IxDom\ (SOps\ S)$$

HOL Constant

$$\boldsymbol{Arity_i} : SIG \to STRING \to \mathbb{N}$$

$$\forall A\ n \bullet\ Arity_i\ A\ n = ValueOf\ (A\ n)$$

$$\boldsymbol{sig\_arity\_lemma} =$$
$$\vdash\ \forall\ A\ n \bullet\ n \in IxDom\ (Sig\ A) \Rightarrow Arity_i\ (Sig\ A)\ n = Arity_u\ A\ n$$

IxInc (aliased as $\sqsubseteq$) is inclusion of indexed sets, and suffices for signature inclusion.

$$\boldsymbol{IxDom\_Sig\_SOps\_thm} =$$
$$\vdash\ \forall\ A \bullet\ IxDom\ (Sig\ A) = IxDom\ (SOps\ A)$$

$$\boldsymbol{SigInc\_IxDom\_\subseteq\_thm} =$$
$$\vdash\ \forall\ A\ B \bullet\ Sig\ A \sqsubseteq Sig\ B \Rightarrow IxDom\ (Sig\ A) \subseteq IxDom\ (Sig\ B)$$

$$\boldsymbol{SigInc\_IxDom\_Sops\_\subseteq\_thm} =$$
$$\vdash\ \forall\ A\ B \bullet\ Sig\ A \sqsubseteq Sig\ B \Rightarrow IxDom\ (SOps\ A) \subseteq IxDom\ (SOps\ B)$$

$$\boldsymbol{SigInc\_Arity_i\_thm} =$$
$$\vdash\ \forall\ A\ B\ n \bullet\ Sig\ A \sqsubseteq Sig\ B\ \wedge\ n \in IxDom\ (Sig\ A)$$
$$\Rightarrow\ Arity_i\ (Sig\ A)\ n = Arity_i\ (Sig\ B)\ n$$

$$\boldsymbol{SigInc\_Arity_u\_thm} =$$
$$\vdash\ \forall\ A\ B\ n \bullet\ Sig\ A \sqsubseteq Sig\ B\ \wedge\ n \in IxDom\ (Sig\ A)$$
$$\Rightarrow\ Arity_u\ A\ n = Arity_u\ B\ n$$

This is possibly too crude, it might be better to ignore the irrelevant behaviour of the operators (i.e. values off sig or out of domain).

HOL Constant

$$\boldsymbol{StructInc} : ('a)\ STRUCT \to ('a)\ STRUCT \to BOOL$$

$$\forall s\ t \bullet\ StructInc\ s\ t \Leftrightarrow SCar\ s = SCar\ t\ \wedge\ IxInc\ (SOps\ s)\ (SOps\ t)$$

SML

```
declare_alias("⊑", ⌜StructInc⌝);
```

There is a general requirement on structures that the operators are closed on the domain of the structure.

HOL Constant

$$\boldsymbol{ClosedOp} : 'a\ SET \to (\mathbb{N} \times ((\mathbb{N} \to 'a) \to 'a)) \to BOOL$$

$$\forall s\ p \bullet\ ClosedOp\ s\ p = \forall f \bullet\ (\forall i \bullet\ i < Fst\ p \Rightarrow f\ i \in s) \Rightarrow Snd\ p\ f \in s$$

HOL Constant

$$\boldsymbol{ClosedStruct} : ('a)\ STRUCT \to BOOL$$

$$\forall s \bullet\ ClosedStruct\ s \Leftrightarrow \forall p \bullet\ p \in IxRan\ (SOps\ s) \Rightarrow ClosedOp\ (SCar\ s)\ p$$

4.1 Packing Functions

The form of the operators is not ideal for talking about the structures, so we define some functions which will make more convenient forms readily obtainable.

There are two things we need to be able to do. The first is to convert 0-ary 1-ary and 2-ary operations in their usual convenient representation to the representation in which the arguments are collected into an indexed set. The second is to collect the operators into a name-indexed set (this is done by *IxPack* [2]).

HOL Constant

$$\boldsymbol{pack0op} : 'a \to (\mathbb{N} \times ((\mathbb{N} \to 'a) \to 'a))$$

$$\forall c\bullet\ pack0op\ c = (0,\ \lambda is\bullet\ c)$$

HOL Constant

$$\boldsymbol{unpack0op} : ((\mathbb{N} \to 'a) \to 'a) \to 'a$$

$$\forall f\bullet\ unpack0op\ f = f\ (\epsilon x\bullet T)$$

$$\boldsymbol{unpack0op\_lemma} = \vdash\ \forall\ c\bullet\ unpack0op\ (Snd\ (pack0op\ c)) = c$$

HOL Constant

$$\boldsymbol{pack1op} : ('a \to 'a) \to (\mathbb{N} \times ((\mathbb{N} \to 'a) \to 'a))$$

$$\forall f\bullet\ pack1op\ f = (1,\ \lambda is\bullet\ f\ (is\ 0))$$

HOL Constant

$$\boldsymbol{unpack1op} : ((\mathbb{N} \to 'a) \to 'a) \to ('a \to 'a)$$

$$\forall f\bullet\ unpack1op\ f = \lambda x\bullet\ f\ (\lambda y\bullet\ x)$$

$$\boldsymbol{unpack1op\_lemma} = \vdash\ \forall\ f\bullet\ unpack1op\ (Snd\ (pack1op\ f)) = f$$

HOL Constant

$$\boldsymbol{pack2op} : ('a \to 'a \to 'a) \to (\mathbb{N} \times ((\mathbb{N} \to 'a) \to 'a))$$

$$\forall f\bullet\ pack2op\ f = (2,\ \lambda is\bullet\ f\ (is\ 0)\ (is\ 1))$$

HOL Constant

$$\boldsymbol{unpack2op} : ((\mathbb{N} \to 'a) \to 'a) \to ('a \to 'a \to 'a)$$

$$\forall f\bullet\ unpack2op\ f = \lambda x\ y\bullet\ f\ (\lambda z\bullet\ if\ z = 0\ then\ x\ else\ y)$$

$$\boldsymbol{unpack2op\_lemma} = \vdash\ \forall\ f\bullet\ unpack2op\ (Snd\ (pack2op\ f)) = f$$

It is intended that specific algebras are introduced by defining a constructor which takes a domain and a number of operators of various arities and packs them into an object of type *STRUCT*.

Such a definitions has the general form:

$$\forall\ D\ o_1\ o_2 \ldots\ o_n \bullet\ MkAlg\ D\ o_1\ o_2 \ldots\ o_n = MkSTRUCT\ D\ (IxPack\ [(\text{"}o_1\text{"},\ pack?op\ o_1);\ (\text{"}o_2\text{"},\ pack?op\ o_2);\ \ldots\ ;\ (\text{"}o_n\text{"},\ pack?op\ o_n)])$$

It will be convenient to unpack such structures using pattern matching on the kinds of expression which are used to construct them, but in order for the consistency such definitions to be automatically proven it is necessary to supply a theorem for each particular algebra in use of the general form:

$$\forall cf \bullet\ \exists f \bullet\ \forall D\ o_1\ o_2 \ldots\ o_n \bullet\ f(MkAlg\ D\ o_1\ o_2 \ldots\ o_n) = cf\ D\ o_1\ o_2 \ldots\ o_n$$

The witness for proving the existential can be presented in the form:

$$\lambda s \bullet\ cf\ (SCar\ s)\ (unpack?op\ (Oper\ s\ o_1 name))\ (unpack?op\ (Oper\ s\ o_2 name))\ \ldots\ (unpack?op\ (Oper\ s\ o_{nname}))$$

in which the '?' signs and the operator names are discovered from the definition of the constructor *MkAlg*.

Because the operators have different types according to their arity this cannot be expressed as a general theorem, but it is possible to automate the construction an proof of the required theorem given the definition of the particular *MkAlg* constructor, assuming that this packs up its arguments into a structure in a standard way.

We next undertake the required automation.

SML

```
local
  fun oprev f =
      let val (n, t) = dest_const f
          val (_, [ot1, ot2]) = dest_ctype t
          val (_, [_, ot3]) = dest_ctype ot2
          val nt = list_mk_→_type [ot3, ot1]
          val nop = mk_const (implode((explode "un")@(explode n)), nt)
       in nop
      end

  fun opmap (l, r) =
      let val cname = dest_string l
          val (pakfun, oper) = dest_app r
          val rtype = type_of r
          val (_, [_, ot]) = dest_ctype rtype
          val (ft, [_, tv]) = dest_ctype ot
          val opex = list_mk_app (mk_const ("Oper", inst_type [(tv, ⌜:'a⌝)] ⌜:'a STRUCT → STRING → (
                  [mk_var("s", inst_type [(tv, ⌜:'a⌝)] ⌜:'a STRUCT⌝), l])
          val res = list_mk_app (oprev pakfun, [opex])
      in res
```

```
    end

in fun make_alg_∃_thm tm =
    let val spec = get_spec tm;
        val (vars, body) = strip_binder "∀" (concl spec);
        val (_, [mkalge, mkstructe]) = strip_app body;
        val (mkalg, (d1 :: ops)) = strip_app mkalge;
        val (mkalgname, mkalgtype) = dest_const mkalg;
        val (mkstructetype :: revalgtype) = rev (strip_→_type mkalgtype);
        val (_, [d, packe]) = strip_app mkstructe;
        val (_, [packlist]) = strip_app packe;
        val oplist = dest_list packlist;
        val newoplist = (map (opmap o dest_pair) oplist);
        val codtype = mk_vartype (string_variant (term_tyvars mkalge) "'b");
        val vars = mk_var("s", mkstructetype);
        val varf = mk_var("f", mk_→_type (mkstructetype, codtype));
        val varcf = mk_var("cf", list_mk_→_type (rev(codtype::revalgtype)));
        val cfe = list_mk_app (varcf, d1::ops);
        val fe = mk_app (varf, mkalge);
        val eq1 = mk_eq (fe, cfe);
        val innerall = list_mk_∀ (d1::ops, eq1);
        val exists = mk_∃ (varf, innerall);
        val conjec = mk_∀ (varcf, exists);
        val scars = mk_app (mk_const("SCar", mk_→_type (mkstructetype, type_of d1)) , vars);
        val witness = mk_λ (vars, list_mk_app (varcf, (scars::newoplist)));
        val pat_thm = tac_proof(([], conjec),
                REPEAT ∀_tac THEN ∃_tac witness THEN rewrite_tac [spec, SOps_def]);
        in pat_thm
    end;
end;

fst_packop_lemma =
    ⊢ (∀ k• Fst (pack0op k) = 0)
    ∧ (∀ k• Fst (pack1op k) = 1)
    ∧ (∀ k• Fst (pack2op k) = 2)

UniOpLift_pack0op_lemma =
    ⊢ ∀ C r op• UniOpLift (C, r) (pack0op op) = pack0op (EquivClass (C, r) op)

UniOpLift_pack1op_lemma =
    ⊢ ∀ C r op• UniOpLift (C, r) (pack1op op) = pack1op (op MonOpLift (C, r))

UniOpLift_pack2op_lemma =
    ⊢ ∀ C r op• UniOpLift (C, r) (pack2op op) = pack2op (op DyOpLift (C, r))
```

We need to know that lifting commutes with packing, which is shown by the following theorems.

4.2 Homomorphisms

Now we can define the notion of homomorphism.

First we define the requirement on the homomorphism to map the domain of the source into the domain of the target.

HOL Constant

$$\boldsymbol{FunClosed} : ('b)STRUCT \times ('b \to 'c) \times ('c)\ STRUCT \to BOOL$$

$$\forall\ A\ f\ B\bullet\ FunClosed\ (A,\ f,\ B) \Leftrightarrow \forall\ x\bullet\ x \in SCar\ A \Rightarrow f\ x \in SCar\ B$$

$$\boldsymbol{FunClosed\_trans\_thm} = \vdash \forall\ A\ f\ B\ g\ C\bullet$$
$$FunClosed\ (A,\ f,\ B) \wedge FunClosed\ (B,\ g,\ C) \Rightarrow FunClosed\ (A,\ g\ o\ f,\ C)$$

$$\boldsymbol{FunClosed\_FunImage\_thm} =$$
$$\vdash \forall\ A\ f\ B\bullet\ FunClosed\ (A,\ f,\ B) \Leftrightarrow FunImage\ f\ (SCar\ A) \subseteq SCar\ B$$

Then the requirement that the function respects an operator is expressed:

HOL Constant

$$\boldsymbol{OpRespect} : ('b\ SET \times ('b \to 'c) \times \mathbb{N}) \to ((\mathbb{N} \to 'b) \to 'b) \to ((\mathbb{N} \to 'c) \to 'c) \to BOOL$$

$$\forall\ D\ f\ n\ op1\ op2\bullet\ OpRespect\ (D,\ f,\ n)\ op1\ op2 \Leftrightarrow$$
$$\forall g\bullet\ FunImage\ g\ \{i \mid i < n\} \subseteq D \Rightarrow f\ (op1\ g) = op2\ (\lambda x\bullet\ f\ (g\ x))$$

$$\boldsymbol{OpRespect\_pack0op\_lemma} =$$
$$\vdash \forall\ D\ f\ d\ c\bullet\ OpRespect\ (D,\ f,\ 0)\ (Snd\ (pack0op\ d))\ (Snd\ (pack0op\ c)) \Leftrightarrow f\ d = c$$

$$\boldsymbol{OpRespect\_pack1op\_lemma} =$$
$$\vdash \forall\ D\ f\ d\ c\bullet\ OpRespect\ (D,\ f,\ 1)\ (Snd\ (pack1op\ d))\ (Snd\ (pack1op\ c))$$
$$\Leftrightarrow (\forall\ x\bullet\ x \in D \Rightarrow f\ (d\ x) = c\ (f\ x))$$

$$\boldsymbol{OpRespect\_pack2op\_lemma} =$$
$$\vdash \forall\ D\ f\ \$*_d\ \$*_c\bullet\ OpRespect\ (D,\ f,\ 2)\ (Snd\ (pack2op\ \$*_d))\ (Snd\ (pack2op\ \$*_c))$$
$$\Leftrightarrow (\forall\ x\ y\bullet\ x \in D \wedge y \in D \Rightarrow f\ (x *_d y) = f\ x *_c f\ y)$$

SML

```
declare_infix (200, "*d");
declare_infix (200, "*c");
```

HOL Constant

$$\boldsymbol{HomOp} : ('b)STRUCT \times ('b \to 'c) \times ('c)\ STRUCT \to STRING \to BOOL$$

$$\forall\ A\ s\ B\ f\bullet\ HomOp\ (A,\ f,\ B)\ s \Leftrightarrow$$
$$OpRespect\ (SCar\ A,\ f,\ Arity_u\ A\ s)\ (Oper\ A\ s)\ (Oper\ B\ s)$$

and the requirement that the function respects all the operators in the signature of the domain.

HOL Constant

$$\textbf{\textit{HomOps}} : ('b)STRUCT \times ('b \to 'c) \times ('c)\ STRUCT \to BOOL$$

$$\forall\ A\ B\ f \bullet\ HomOps\ (A,\ f,\ B) \Leftrightarrow \forall s \bullet\ s \in IxDom\ (SOps\ A) \Rightarrow HomOp\ (A,\ f,\ B)\ s$$

$$\textbf{\textit{HomOps\_o\_thm}} =$$
$$\vdash \forall\ A\ f\ B\ g\ C \bullet\ HomOps\ (A,\ f,\ B)$$
$$\wedge\ Sig\ A \sqsubseteq Sig\ B \wedge FunClosed\ (A,\ f,\ B)$$
$$\wedge\ HomOps\ (B,\ g,\ C)$$
$$\Rightarrow HomOps\ (A,\ g\ o\ f,\ C)$$

HOL Constant

$$\textbf{\textit{AlgHom}} : ('b)STRUCT \times ('b \to 'c) \times ('c)\ STRUCT \to BOOL$$

$$\forall\ A\ f\ B \bullet\ AlgHom\ (A,\ f,\ B) \Leftrightarrow$$
$$Sig\ A \sqsubseteq Sig\ B \wedge FunClosed\ (A,\ f,\ B) \wedge HomOps\ (A,\ f,\ B)$$

$$\textbf{\textit{AlgHom\_o\_thm}} =$$
$$\vdash \forall\ A\ f\ B\ g\ C \bullet\ AlgHom\ (A,\ f,\ B) \wedge AlgHom\ (B,\ g,\ C) \Rightarrow AlgHom\ (A,\ g\ o\ f,\ C)$$

4.3 Quotients

Given an equivalence relation which is respected by the operators in an algebra, a quotient algebra can be obtained whose objects are equivalence classes and whose operators are the operators of the original algebra lifted to operate on the equivalence classes.

The quotient operation can then be defined as follows:

HOL Constant

$$\textbf{\textit{QuotientAlg}} : ('b)STRUCT \to ('b \to 'b \to BOOL) \to ('b\ SET)\ STRUCT$$

$$\forall\ A\ r \bullet\ QuotientAlg\ A\ r = MkSTRUCT\ (QuotientSet\ (SCar\ A)\ r)\ (IxComp\ (SOps\ A)\ (UniOpLift\ (SCar$$

With the usual alias:

SML

```
declare_alias ("/", ⌜QuotientAlg⌝);
```

If an equivalence relation respects the operators in an algebra, then the quotient group will be an homomorphic image of the original algebra. We now seek to simplify the necessary conditions on the equivalence relation.

$$\textbf{\textit{QuotientAlg\_Sig\_lemma}} =$$
$$\vdash \forall\ A\ r \bullet\ Sig\ A \sqsubseteq Sig\ (QuotientAlg\ A\ r)$$

$$\textbf{\textit{AlgHom\_EquivClass\_HomOps\_lemma}} =$$
$$\vdash\ AlgHom\ (A,\ EquivClass\ (SCar\ A,\ r),\ QuotientAlg\ A\ r)$$
$$\Leftrightarrow\ HomOps\ (A,\ EquivClass\ (SCar\ A,\ r),\ QuotientAlg\ A\ r)$$

4.4 Algebraic Equations

Algebraic equations in an algebra are preserved under homomorphism.

To prove this general claim we must first define the concept of an algebraic equation, which will be done inductively. We need to be able to talk about the same algebraic equation over two distinct algebras so the notion will be parameterised by an algebra.

We will represent an expression over an algebra by a function from a valuation of variables to a value. We generate the expressions for a specific "signature". The signature is just a triple giving the number of 0-ary, 1-ary and 2-ary operations in the algebra.

SML

$$declare\_type\_abbrev(\texttt{"}EXPR\texttt{"}, [\texttt{"}'a\texttt{"}], \ulcorner:('a)\ STRUCT \rightarrow ((\mathbb{N} \rightarrow 'a) \rightarrow 'a)\urcorner);$$

There is no syntax in this account of polynomials, so the list is a list of the values of the variables.

An equation is a pair of polynomials.

To define the set of polynomials we first define the ways in which new polynomials can be constucted from a set already to hand. This is by the use of any of the operations in the algebra, and the operation involved is functional composition.

HOL Constant

$$\boldsymbol{VExpr} : \mathbb{N} \rightarrow 'a\ EXPR$$

$$\forall n \bullet\ VExpr\ n = \lambda a\ va \bullet\ va\ n$$

HOL Constant

$$\boldsymbol{VExprs} : \mathbb{N} \rightarrow 'a\ EXPR\ SET$$

$$\forall n \bullet\ VExprs\ n = \{p \mid \exists m \bullet\ m < n \wedge p = VExpr\ m\}$$

HOL Constant

$$\boldsymbol{CExprs} : SIG \rightarrow 'a\ EXPR\ SET \rightarrow 'a\ EXPR\ SET$$

$$\begin{array}{l}\forall sig\ es \bullet\ CExprs\ sig\ es = \{e \mid \exists name\ arity\ am \bullet \\ \qquad name \in IxDom\ sig \wedge sig\ name = Value\ arity \\ \quad \wedge \quad (\forall i \bullet\ i < arity \Rightarrow am\ i \in es) \\ \quad \wedge \quad e = \lambda struct\ va \bullet\ (Snd\ (ValueOf\ (SOps\ struct\ name)))\ (\lambda i \bullet\ am\ i\ struct\ va)\}\end{array}$$

Now we define closure under the above operators.

HOL Constant

$$\boldsymbol{ExprClosed} : SIG \rightarrow 'a\ EXPR\ SET \rightarrow BOOL$$

$$\forall s\ es \bullet\ ExprClosed\ s\ es \Leftrightarrow CExprs\ s\ es \subseteq es$$

HOL Constant

$$\boldsymbol{Exprs} : SIG \rightarrow \mathbb{N} \rightarrow 'a\ EXPR\ SET$$

$$\forall s\ n \bullet\ Exprs\ s\ n = \bigcap \{ps \mid ExprClosed\ s\ ps \wedge VExprs\ n \subseteq ps\}$$

5 Lattices (I)

The beginnings of a theory of lattices. This version was done before the work on Univesal Algebra, and will be discarded if the later version conformant with that theory is found satisfactory.

SML

```
open_theory "unalg";
force_new_theory "lattice";
force_new_pc "'lattice";
merge_pcs ["'prove_∃_⇒_conv", "'savedthm_cs_∃_proof"] "'lattice";
set_merge_pcs ["unalg", "'lattice"];
```

SML

```
set_merge_pcs ["hol", "'rbjmisc"] ;
```

5.1 Signature and Defining Property

We will represent a lattice as a triple comprising a carrier set and two-argument join and meet functions.

HOL Labelled Product

LAT

$\mathbf{Car_L}$	$: 'a\ SET$;
$\mathbf{Join_L}$	$: 'a \to 'a \to 'a$;
$\mathbf{Meet_L}$	$: 'a \to 'a \to 'a$

We will use L and M as variables for lattices and the following infixity declarations will be useful as names for the corresponding operations.

SML

```
declare_infix (235, "∨_L");
declare_infix (235, "∨_M");
declare_infix (240, "∧_L");
declare_infix (240, "∧_M");
```

HOL Constant

$$\mathbf{IsLattice} : 'a\ LAT \to BOOL$$

$$\begin{array}{l}
\forall\ L \bullet\ IsLattice\ L \Leftrightarrow \forall\ C\ \$\vee_L\ \$\wedge_L \bullet\ MkLAT\ C\ \$\vee_L\ \$\wedge_L = L \Rightarrow \\
\quad (\forall x\ y \bullet\ x \in C \wedge y \in C \Rightarrow \\
\qquad x \vee_L y \in C \wedge x \wedge_L y \in C \\
\quad \wedge \quad x \vee_L y = y \vee_L x \wedge x \wedge_L y = y \wedge_L x \\
\quad \wedge \quad x \wedge_L (x \vee_L y) = x \wedge x \vee_L (x \wedge_L y) = x \\
\quad \wedge\ (\forall z \bullet\ z \in C \\
\qquad \Rightarrow \quad (x \vee_L y) \vee_L z = x \vee_L (y \vee_L z) \\
\qquad \wedge \quad (x \wedge_L y) \wedge_L z = x \wedge_L (y \wedge_L z)))
\end{array}$$

5.2 Elementary Theorems

$\wedge_{L}$-*idempot_thm* =
  $\vdash \forall\ L$
    • $IsLattice\ L \Rightarrow (\forall\ C\ \$\vee_L\ \$\wedge_L \bullet\ MkLAT\ C\ \$\vee_L\ \$\wedge_L = L$
      $\Rightarrow (\forall\ x \bullet\ x \in C \Rightarrow x\ \wedge_L\ x = x))$

$\wedge_{L}$-*idempot_thm* =
  $\vdash \forall\ L \bullet\ IsLattice\ L \Rightarrow (\forall\ C\ \$\vee_L\ \$\wedge_L \bullet\ MkLAT\ C\ \$\vee_L\ \$\wedge_L = L$
    $\Rightarrow (\forall\ x \bullet\ x \in C \Rightarrow x\ \vee_L\ x = x))$

5.3 Quotient Lattices

The quotient of a lattice with respect to some equivalence relation over its elements is defined as follows:

HOL Constant

QuotientLattice : $'a\ LAT \rightarrow ('a \rightarrow 'a \rightarrow BOOL) \rightarrow 'a\ SET\ LAT$

$\forall\ L\ \$\equiv_d \bullet\ QuotientLattice\ L\ \$\equiv_d =$
  let $D = (Car_L\ L, \$\equiv_d)$
  in let $\$\vee_L = (Join_L\ L)\ DyOpLift\ D$ *and* $\$\wedge_L = (Meet_L\ L)\ DyOpLift\ D$
    in $MkLAT\ (EquivClasses\ D)\ \$\vee_L\ \$\wedge_L$

SML

declare_alias ("/", ⌜*QuotientLattice*⌝);

-GFT

5.4 Lattice Orders

SML

declare_infix (210, "$\leqslant_L$");
declare_infix (210, "$\leqslant_M$");

The ordering on the lattice is derived from the operations as follows,

HOL Constant

LeLat : $'a\ LAT \rightarrow 'a \rightarrow 'a \rightarrow BOOL$

$\forall\ L{:}'a\ LAT \bullet\ LeLat\ L = \lambda x\ y \bullet\ \forall \$\vee_L \bullet\ \$\vee_L = Join_L\ L \Rightarrow x\ \vee_L\ y = y$

6 Lattices (II)

The beginnings of another theory of lattices. This version was done using Universal Algebra, the previous version will be discarded if this version conformant with the approach to universal algebra is found satisfactory.

SML

```
open_theory "unalg";
force_new_theory "latt2";
force_new_pc "'latt2";
merge_pcs ["'prove_∃_⇒_conv", "'savedthm_cs_∃_proof"] "'latt2";
set_merge_pcs ["unalg", "'latt2"];
```

6.1 Signature and Defining Property

We will use the general structure type *STRUCT* to represent lattice structures.

HOL Constant

$$\boldsymbol{LatSig} : SIG$$

$$LatSig = IxPack\ [("\vee_L"\ ,\ 2);\ ("\vee_L"\ ,\ 2)]$$

We will use L and M as variables for lattices and the following infixity declarations will be useful as names for the corresponding operations.

SML

```
declare_infix (235, "∨_L");
declare_infix (235, "∨_M");
declare_infix (240, "∧_L");
declare_infix (240, "∧_M");
```

HOL Constant

$$\boldsymbol{MkLat} : 'a\ SET \to ('a \to 'a \to 'a) \to ('a \to 'a \to 'a) \to ('a)\ STRUCT$$

$$\forall\ C\ \$\vee_L\ \$\wedge_L \bullet\ MkLat\ C\ \$\vee_L\ \$\wedge_L = MkSTRUCT\ C\ (IxPack\ [("\vee_L"\ ,\ pack2op\ \$\vee_L);\ ("\wedge_L"\ ,\ pack2op\ \$\wedge$$

HOL Constant

$$\boldsymbol{IsLat} : ('a)\ STRUCT \to BOOL$$

$$\begin{aligned}
&\forall\ L \bullet\ IsLat\ L \Leftrightarrow \forall\ C\ \$\vee_L\ \$\wedge_L \bullet\ StructInc\ (MkLat\ C\ \$\vee_L\ \$\wedge_L)\ L \Rightarrow \\
&\quad (\forall x\ y \bullet\ x \in C\ \wedge\ y \in C \Rightarrow \\
&\qquad x \vee_L y \in C\ \wedge\ x \wedge_L y \in C \\
&\quad \wedge\quad x \vee_L y = y \vee_L x\ \wedge\ x \wedge_L y = y \wedge_L x \\
&\quad \wedge\quad x \wedge_L (x \vee_L y) = x\ \wedge\ x \vee_L (x \wedge_L y) = x \\
&\quad \wedge\ (\forall z \bullet\ z \in C \\
&\qquad \Rightarrow\quad (x \vee_L y) \vee_L z = x \vee_L (y \vee_L z) \\
&\qquad \wedge\quad (x \wedge_L y) \wedge_L z = x \wedge_L (y \wedge_L z)))
\end{aligned}$$

6.2 Pattern Matching

We now add the the existence proof context the theorem necessary to allow function definitions which pattern-match on patterns formed by *MkLat*. The proof of the theorem is now automatic, from the specification of *MkLat*.

SML

```
val MkLat_∃_lemma = make_alg_∃_thm ⌜MkLat⌝;
```

```
MkLat_∃_lemma =
    ⊢ ∀ cf• ∃ f• ∀ C $∨L $∧L• f (MkLat C $∨L $∧L) = cf C $∨L $∧L
```

This gets plugged into proof context *'latt2* for use in consistency proofs.

SML

```
add_∃_cd_thms [MkLat_∃_lemma] "'latt2";
set_merge_pcs ["unalg", "'latt2"];
```

6.3 Lattice Ordering

Well try out the pattern matching definition by defining the ordering on a lattice.

HOL Constant

```
LeLat : 'a STRUCT → 'a → 'a → BOOL
```

$$\forall C\ \$\vee_L\ \$\wedge_L\bullet\ LeLat\ (MkLat\ C\ \$\vee_L\ \$\wedge_L) = \lambda x\ y\bullet\ x\ \vee_L\ y = y$$

6.4 Elementary Theorems

```
∧L_idempot_thm =
  ⊢ ∀ L• IsLat L ⇒ (∀ C $∨L $∧L• MkLat C $∨L $∧L ⊑ L
      ⇒ (∀ x• x ∈ C ⇒ x ∧L x = x))

∨L_idempot_thm =
  ⊢ ∀ L• IsLat L ⇒ (∀ C $∨L $∧L• MkLat C $∨L $∧L ⊑ L
      ⇒ (∀ x• x ∈ C ⇒ x ∨L x = x))
```

6.5 Homomorphisms

I am hoping that results from Univeral Algebra will not prove difficult to instantiate to Lattice theory.

Let us define a Lattice homomorphism as a homomorphism whose source is a lattice.

HOL Constant

```
LatHom : ('a) STRUCT × ('a → 'b) × 'b STRUCT) → BOOL
```

$$\forall\ A\ f\ B\bullet\ LatHom\ (A, f, B) \Leftrightarrow AlgHom\ (A, f, B) \wedge Sig\ A = LatSig$$

$$\textbf{\textit{LatHom\_o\_thm}} =$$
$$\vdash \forall\ A\ f\ B\ g\ C \bullet\ LatHom\ (A,\ f,\ B) \wedge AlgHom\ (B,\ g,\ C) \Rightarrow LatHom\ (A,\ g\ o\ f,\ C)$$

$$\textbf{\textit{Latt\_HomOps\_lemma}} =$$
$$\vdash \forall\ C\ \$\vee_L\ \$\wedge_L\ f\ D\ \$\vee_M\ \$\wedge_M \bullet\ HomOps\ (MkLat\ C\ \$\vee_L\ \$\wedge_L,\ f,\ MkLat\ D\ \$\vee_M\ \$\wedge_M)$$
$$\Leftrightarrow (\forall\ x\ y \bullet\ x \in C \wedge y \in C \Rightarrow f\ (x\ \wedge_L\ y) = f\ x\ \wedge_M\ f\ y)$$
$$\wedge\ (\forall\ x\ y \bullet\ x \in C \wedge y \in C \Rightarrow f\ (x\ \vee_L\ y) = f\ x\ \vee_M\ f\ y)$$

6.6 Quotient Lattices

A quotient lattice requires no special definition, it is simply a *QuotientAlg* of a lattice. However, as in the case of the homomorphism, it is convenient to have the conditions expressed explicitly in terms of the lattice operations.

7 Conclusions

8 Postscript

My main near term objective for this document is to supply the necessary lattice theory for X-Logic.

For this purpose I need to be able to reason about quotient lattices, and therefore am thinking in terms of a general quotient construction in universal algebra applied in the lattice theory.

A general theory of quotient algebras is therefore my next objective.

A Theory Listings

A.1 The Theory equiv

Parents

rbjmisc

Children

unalg

Constants

EquivClasses	$'a\ \mathbb{P} \times ('a \to 'a \to BOOL) \to 'a\ \mathbb{P}\ \mathbb{P}$
$\$$***Respects*1**	$('a \to 'b)$ $\to ('a\ \mathbb{P} \times ('a \to 'a \to BOOL)) \times 'b\ \mathbb{P} \times ('b \to 'b \to BOOL)$ $\to BOOL$
$\$$***LiftOver***	$('a \to 'b)$ $\to ('a\ \mathbb{P} \times ('a \to 'a \to BOOL)) \times 'b\ \mathbb{P} \times ('b \to 'b \to BOOL)$ $\to 'a\ \mathbb{P}$ $\to 'b\ \mathbb{P}$
$\$$***MonOpRespects***	$('a \to 'a) \to 'a\ \mathbb{P} \times ('a \to 'a \to BOOL) \to BOOL$
$\$$***MonOpLift***	$('a \to 'a) \to 'a\ \mathbb{P} \times ('a \to 'a \to BOOL) \to 'a\ \mathbb{P} \to 'a\ \mathbb{P}$
$\$$***DyOpRespects***	$('a \to 'a \to 'a) \to 'a\ \mathbb{P} \times ('a \to 'a \to BOOL) \to BOOL$
$\$$***DyOpLift***	$('a \to 'a \to 'a)$ $\to 'a\ \mathbb{P} \times ('a \to 'a \to BOOL)$ $\to 'a\ \mathbb{P}$ $\to 'a\ \mathbb{P}$ $\to 'a\ \mathbb{P}$
UniOpLift	$'b\ \mathbb{P} \times ('b \to 'b \to BOOL) \to 'b\ UOP \to 'b\ \mathbb{P}\ UOP$
RelIncl	$'b\ \mathbb{P} \times ('b \to 'b \to BOOL)$ $\to 'b\ \mathbb{P} \times ('b \to 'b \to BOOL)$ $\to BOOL$
EquivClosure	$'b\ \mathbb{P} \times ('b \to 'b \to BOOL) \to 'b\ \mathbb{P} \times ('b \to 'b \to BOOL)$

Aliases

$\subseteq$	*RelIncl* $: 'a\ \mathbb{P} \times ('a \to 'a \to BOOL)$ $\to 'a\ \mathbb{P} \times ('a \to 'a \to BOOL)$ $\to BOOL$

Type Abbreviations

$'a$ ***UOP***	$'a\ UOP$

Fixity

Right Infix 200:
DyOpRespects MonOpRespects
LiftOver Respects1

Right Infix 210:
DyOpLift ***MonOpLift***

Right Infix 230:
$\equiv$ $\equiv_{c}$ $\equiv_{d}$ $\equiv_{e}$ $\equiv_{f}$ $\equiv_{l}$ $\equiv_{r}$

Definitions

EquivClasses $\vdash$ *EquivClasses* = *Uncurry* \$/

Respects1 $\vdash \forall\ f\ \$\equiv_{c}\ \$\equiv_{d}\ C\ D$
- *f Respects1* $((D, \$\equiv_{d}), C, \$\equiv_{c})$
  $\Leftrightarrow (\forall\ x\ y$
  - $x \in D \wedge y \in D \wedge x \equiv_{d} y$
    $\Rightarrow f\ x \in C \wedge f\ y \in C \wedge f\ x \equiv_{c} f\ y)$

LiftOver $\vdash \forall\ f\ (D, \$\equiv_{d})\ (C, \$\equiv_{c})$
- $(f$ *LiftOver* $((D, \$\equiv_{d}), C, \$\equiv_{c}))$
  $= (\lambda\ x$
  - $\epsilon\ y$
    - $\exists\ z\bullet\ z \in x \wedge y =$ *EquivClass* $(C, \$\equiv_{c})\ (f\ z))$

MonOpRespects
$\vdash \forall\ f\ e\bullet\ f$ *MonOpRespects* $e \Leftrightarrow f$ *Respects1* (e, e)

MonOpLift $\vdash \forall\ f\ (D, \$\equiv_{d})$
- f *MonOpLift* $(D, \$\equiv_{d})$
  $= f$
  LiftOver $((D, \$\equiv_{d}), D, \$\equiv_{d})$

DyOpRespects $\vdash \forall\ f\ e$
- f *DyOpRespects* e
  $\Leftrightarrow$ *Uncurry* f *Respects1* $(e$ *RelProd* $e, e)$

DyOpLift $\vdash \forall\ f\ (D, \$\equiv_{d})$
- f *DyOpLift* $(D, \$\equiv_{d})$
  $= (\lambda\ x\ y$
  - $($*Uncurry* f
    LiftOver $((D, \$\equiv_{d})$ *RelProd* $(D, \$\equiv_{d}), D,$
    $\$\equiv_{d}))$
    $(x \times y))$

UniOpLift $\vdash \forall\ a\ op\ r$
- *UniOpLift* $r\ (a, op)$
  $= (a,$
  $(\lambda\ f$
  - $($*op LiftOver* $($*RelPower* $r\ \{x | x < a\}, r))$
    $\{g | \forall\ i\bullet\ i < a \Rightarrow g\ i \in f\ i\}))$

RelIncl $\vdash \forall\ A\ r1\ B\ r2$
- $(A, r1) \subseteq (B, r2)$
  $\Leftrightarrow (\forall\ x\ y$
  - $x \in A \wedge y \in A \wedge r1\ x\ y$
    $\Rightarrow x \in B \wedge y \in B \wedge r2\ x\ y)$

EquivClosure $\vdash \forall\ A\ r$

- $EquivClosure\ (A,\ r)$
  $= (A,$
  $(\lambda\ x\ y$
  - $\forall\ \$\equiv_d$
    - $Equiv\ (A,\ \$\equiv_d)\ \wedge\ (A,\ r) \subseteq (A,\ \$\equiv_d)$
      $\Rightarrow\ x\ \equiv_d\ y))$

Theorems

EquivClasses_thm
$\vdash\ \forall\ (X,\ \$\equiv)$
- $EquivClasses\ (X,\ \$\equiv)$
  $= \{A | \exists\ x\bullet\ x \in X\ \wedge\ A = EquivClass\ (X,\ \$\equiv)\ x\}$

EquivClasses_thm1
$\vdash\ \forall\ r$
- $EquivClasses\ r$
  $= \{A | \exists\ x\bullet\ x \in Fst\ r\ \wedge\ A = EquivClass\ r\ x\}$

EquivClasses_sub_thm
$\vdash\ \forall\ (D,\ \$\equiv)$
- $Equiv\ (D,\ \$\equiv)$
  $\Rightarrow\ (\forall\ l\bullet\ l \in EquivClasses\ (D,\ \$\equiv) \Rightarrow l \subseteq D)$

EquivClasses_sub_thm1
$\vdash\ \forall\ r\bullet\ Equiv\ r \Rightarrow (\forall\ l\bullet\ l \in EquivClasses\ r \Rightarrow l \subseteq Fst\ r)$

Equiv_RelProd_thm
$\vdash\ \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)$
- $Equiv\ (L,\ \$\equiv_l)\ \wedge\ Equiv\ (R,\ \$\equiv_r)$
  $\Rightarrow\ Equiv\ ((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r))$

EquivClass_RelProd_thm
$\vdash\ \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)$
- $Equiv\ (L,\ \$\equiv_l)\ \wedge\ Equiv\ (R,\ \$\equiv_r)$
  $\Rightarrow\ (\forall\ xl\ xr\ yl\ yr$
  - $yl \in L\ \wedge\ yr \in R$
    $\Rightarrow\ ((xl,\ xr)$
    $\in\ EquivClass$
    $((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r))$
    $(yl,\ yr)$
    $\Leftrightarrow\ xl \in EquivClass\ (L,\ \$\equiv_l)\ yl$
    $\wedge\ xr \in EquivClass\ (R,\ \$\equiv_r)\ yr))$

EquivClass_RelProd_thm1
$\vdash\ \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)$
- $Equiv\ (L,\ \$\equiv_l)\ \wedge\ Equiv\ (R,\ \$\equiv_r)$
  $\Rightarrow\ (\forall\ l\ r$
  - $l \in L\ \wedge\ r \in R$
    $\Rightarrow\ EquivClass$
    $((L,\ \$\equiv_l)\ RelProd\ (R,\ \$\equiv_r))$
    $(l,\ r)$
    $=\ (EquivClass\ (L,\ \$\equiv_l)\ l$
    $\times\ EquivClass\ (R,\ \$\equiv_r)\ r))$

EquivClasses_RelProd_thm
$\vdash\ \forall\ (L,\ \$\equiv_l)\ (R,\ \$\equiv_r)$
- $Equiv\ (L,\ \$\equiv_l)\ \wedge\ Equiv\ (R,\ \$\equiv_r)$
  $\Rightarrow\ (\forall\ l\ r$

$\bullet\ (l \times r)$
$\in EquivClasses$
$((L, \$\equiv_l)\ RelProd\ (R, \$\equiv_r))$
$\Leftrightarrow l \in EquivClasses\ (L, \$\equiv_l)$
$\wedge\ r \in EquivClasses\ (R, \$\equiv_r))$

EquivClasses_RelProd_thm1
$\vdash \forall\ (L, \$\equiv_l)\ (R, \$\equiv_r)$
$\bullet\ Equiv\ (L, \$\equiv_l) \wedge Equiv\ (R, \$\equiv_r)$
$\Rightarrow (\forall\ x$
$\bullet\ x \in EquivClasses\ ((L, \$\equiv_l)\ RelProd\ (R, \$\equiv_r))$
$\Leftrightarrow (\exists\ l\ r$
$\bullet\ x = (l \times r)$
$\wedge\ l \in EquivClasses\ (L, \$\equiv_l)$
$\wedge\ r \in EquivClasses\ (R, \$\equiv_r)))$

EquivClasses_RelProd_thm2
$\vdash \forall\ (L, \$\equiv_l)\ (R, \$\equiv_r)$
$\bullet\ Equiv\ (L, \$\equiv_l) \wedge Equiv\ (R, \$\equiv_r)$
$\Rightarrow EquivClasses\ ((L, \$\equiv_l)\ RelProd\ (R, \$\equiv_r))$
$= EquivClasses\ (L, \$\equiv_l)$
$\times_D\ EquivClasses\ (R, \$\equiv_r)$

RelPower_Equiv_thm
$\vdash \forall\ (D, \$\equiv)\ is$
$\bullet\ Equiv\ (D, \$\equiv) \Rightarrow Equiv\ (RelPower\ (D, \$\equiv)\ is)$

Respects1_Respects_thm
$\vdash \forall\ f\ \$\equiv_d\ D$
$\bullet\ f\ Respects1\ ((D, \$\equiv_d), Universe, \$=)$
$\Leftrightarrow (f\ Respects\ \$\equiv_d)\ D$

Respects1_Refines_thm
$\vdash \forall\ f\ \$\equiv_d\ \$\equiv_c\ \$\equiv_e\ \$\equiv_f\ D\ C$
$\bullet\ f\ Respects1\ ((D, \$\equiv_d), C, \$\equiv_c)$
$\wedge\ (\$\equiv_e\ Refines\ \$\equiv_d)\ D$
$\wedge\ (\$\equiv_c\ Refines\ \$\equiv_f)\ C$
$\Rightarrow f\ Respects1\ ((D, \$\equiv_e), C, \$\equiv_f)$

eq_Refines_thm
$\vdash \forall\ \$\equiv_d\ D\bullet\ Equiv\ (D, \$\equiv_d) \Rightarrow (\$=\ Refines\ \$\equiv_d)\ D$

constant_img_thm1
$\vdash \forall\ f\ A\ a\ c$
$\bullet\ a \in A \wedge (\forall\ x\bullet\ x \in A \Rightarrow f\ x = c)$
$\Rightarrow (\epsilon\ y\bullet\ \exists\ x\bullet\ x \in A \wedge y = f\ x) = c$

LiftOver_thm $\vdash \forall\ f\ (D, \$\equiv_d)\ (C, \$\equiv_c)\ de\ ce$
$\bullet\ Equiv\ (D, \$\equiv_d)$
$\wedge\ Equiv\ (C, \$\equiv_c)$
$\wedge\ f\ Respects1\ ((D, \$\equiv_d), C, \$\equiv_c)$
$\wedge\ de \in D\ /\ \$\equiv_d$
$\wedge\ ce \in C\ /\ \$\equiv_c$
$\Rightarrow ((f\ LiftOver\ ((D, \$\equiv_d), C, \$\equiv_c))\ de = ce$
$\Leftrightarrow (\exists\ d\ c$
$\bullet\ d \in D$
$\wedge\ EquivClass\ (D, \$\equiv_d)\ d = de$
$\wedge\ EquivClass\ (C, \$\equiv_c)\ c = ce$
$\wedge\ c = f\ d))$

MonOpRespects_thm

$\vdash \forall\ f\ C\ \$\equiv$
- f *MonOpRespects* $(C, \$\equiv)$
  $\Leftrightarrow (\forall\ x\ y$
  - $x \in C \wedge y \in C \wedge x \equiv y$
    $\Rightarrow f\ x \in C \wedge f\ y \in C \wedge f\ x \equiv f\ y)$

MonOpLift_thm

$\vdash \forall\ f\ (D, \$\equiv_d)\ de\ ce$
- *Equiv* $(D, \$\equiv_d)$
  $\wedge$ f *MonOpRespects* $(D, \$\equiv_d)$
  $\wedge$ $de \in D\ /\ \$\equiv_d$
  $\wedge$ $ce \in D\ /\ \$\equiv_d$
  $\Rightarrow ((f$ *MonOpLift* $(D, \$\equiv_d))\ de = ce$
  $\Leftrightarrow (\exists\ d\ c$
  - $d \in D$
    $\wedge$ *EquivClass* $(D, \$\equiv_d)\ d = de$
    $\wedge$ *EquivClass* $(D, \$\equiv_d)\ c = ce$
    $\wedge\ c = f\ d))$

DyOpRespects_thm

$\vdash \forall\ f\ C\ \$\equiv$
- f *DyOpRespects* $(C, \$\equiv)$
  $\Leftrightarrow (\forall\ x1\ y1\ x2\ y2$
  - $x1 \in C$
    $\wedge\ y1 \in C$
    $\wedge\ x2 \in C$
    $\wedge\ y2 \in C$
    $\wedge\ x1 \equiv x2$
    $\wedge\ y1 \equiv y2$
    $\Rightarrow f\ x1\ y1 \in C$
    $\wedge\ f\ x2\ y2 \in C$
    $\wedge\ f\ x1\ y1 \equiv f\ x2\ y2)$

DyOpLift_thm$\vdash \forall\ f\ (D, \$\equiv_d)\ l\ r\ c$
- *Equiv* $(D, \$\equiv_d)$
  $\wedge$ f *DyOpRespects* $(D, \$\equiv_d)$
  $\wedge$ $l \in D\ /\ \$\equiv_d$
  $\wedge$ $r \in D\ /\ \$\equiv_d$
  $\wedge$ $c \in D\ /\ \$\equiv_d$
  $\Rightarrow ((f$ *DyOpLift* $(D, \$\equiv_d))\ l\ r = c$
  $\Leftrightarrow (\exists\ le\ re\ ce$
  - $le \in D$
    $\wedge\ re \in D$
    $\wedge$ *EquivClass* $(D, \$\equiv_d)\ le = l$
    $\wedge$ *EquivClass* $(D, \$\equiv_d)\ re = r$
    $\wedge$ *EquivClass* $(D, \$\equiv_d)\ ce = c$
    $\wedge\ ce = f\ le\ re))$

Equiv_Equiv_closure_thm

$\vdash \forall\ A\ r$
- *Equiv*
  $(A,$
  $(\lambda\ x\ y$
  - $\forall\ \$\equiv_d$

$\bullet$ $Equiv\ (A,\ \$\equiv_d)\ \wedge\ (A,\ r) \subseteq (A,\ \$\equiv_d)$
$\Rightarrow\ x\ \equiv_d\ y))$

Equiv_EquivClosure_thm

$\vdash\ \forall\ A\ r \bullet\ Equiv\ (EquivClosure\ (A,\ r))$

EquivClosure_MinEquiv_thm

$\vdash\ \forall\ A\ r\ \$\equiv_d$
$\bullet$ $Equiv\ (A,\ \$\equiv_d)\ \wedge\ (A,\ r) \subseteq (A,\ \$\equiv_d)$
$\Rightarrow\ EquivClosure\ (A,\ r) \subseteq (A,\ \$\equiv_d)$

Fst_EquivClosure_thm

$\vdash\ \forall\ eq \bullet\ Fst\ (EquivClosure\ eq) = Fst\ eq$

A.2 The Theory unalg

Parents

equiv

Children

uacat latt2 lattice

Constants

SOps	$'a\ STRUCT \to (STRING, 'a\ UOP)\ IX$
SCar	$'a\ STRUCT \to 'a\ \mathbb{P}$
MkSTRUCT	$'a\ \mathbb{P} \to (STRING, 'a\ UOP)\ IX \to 'a\ STRUCT$
Arity$_u$	$'a\ STRUCT \to STRING \to \mathbb{N}$
Oper	$'a\ STRUCT \to STRING \to (\mathbb{N} \to 'a) \to 'a$
Sig	$'a\ STRUCT \to SIG$
Arity$_i$	$SIG \to STRING \to \mathbb{N}$
StructInc	$'a\ STRUCT \to 'a\ STRUCT \to BOOL$
ClosedOp	$'a\ \mathbb{P} \to 'a\ UOP \to BOOL$
ClosedStruct	$'a\ STRUCT \to BOOL$
pack0op	$'a \to 'a\ UOP$
unpack0op	$((\mathbb{N} \to 'a) \to 'a) \to 'a$
pack1op	$('a \to 'a) \to 'a\ UOP$
unpack1op	$((\mathbb{N} \to 'a) \to 'a) \to 'a \to 'a$
pack2op	$('a \to 'a \to 'a) \to 'a\ UOP$
unpack2op	$((\mathbb{N} \to 'a) \to 'a) \to 'a \to 'a \to 'a$
FunClosed	$'b\ STRUCT \times ('b \to 'c) \times 'c\ STRUCT \to BOOL$
OpRespect	$'b\ \mathbb{P} \times ('b \to 'c) \times \mathbb{N}$ $\to ((\mathbb{N} \to 'b) \to 'b)$ $\to ((\mathbb{N} \to 'c) \to 'c)$ $\to BOOL$
HomOp	$'b\ STRUCT \times ('b \to 'c) \times 'c\ STRUCT \to STRING \to BOOL$
HomOps	$'b\ STRUCT \times ('b \to 'c) \times 'c\ STRUCT \to BOOL$
AlgHom	$'b\ STRUCT \times ('b \to 'c) \times 'c\ STRUCT \to BOOL$
QuotientAlg	$'b\ STRUCT \to ('b \to 'b \to BOOL) \to 'b\ \mathbb{P}\ STRUCT$
VExpr	$\mathbb{N} \to 'a\ EXPR$
VExprs	$\mathbb{N} \to 'a\ EXPR\ \mathbb{P}$
CExprs	$SIG \to 'a\ EXPR\ \mathbb{P} \to 'a\ EXPR\ \mathbb{P}$
ExprClosed	$SIG \to 'a\ EXPR\ \mathbb{P} \to BOOL$
Exprs	$SIG \to \mathbb{N} \to 'a\ EXPR\ \mathbb{P}$

Aliases

$\sqsubseteq$	$StructInc : 'a\ STRUCT \to 'a\ STRUCT \to BOOL$
$/$	$QuotientAlg$ $: 'a\ STRUCT \to ('a \to 'a \to BOOL) \to 'a\ \mathbb{P}\ STRUCT$

Types

'1 ***STRUCT***

Type Abbreviations

SIG	*SIG*
'a ***EXPR***	*'a EXPR*

Fixity

Right Infix 200:

$*_{\boldsymbol{c}}$ $*_{\boldsymbol{d}}$

Definitions

STRUCT	$\vdash \exists\, f\bullet\ TypeDefn\ (\lambda\ x\bullet\ T)\ f$
MkSTRUCT	
SCar	
SOps	$\vdash \forall\ t\ x1\ x2$
	$\bullet\ SCar\ (MkSTRUCT\ x1\ x2) = x1$
	$\wedge\ SOps\ (MkSTRUCT\ x1\ x2) = x2$
	$\wedge\ MkSTRUCT\ (SCar\ t)\ (SOps\ t) = t$
$\boldsymbol{Arity_u}$	$\vdash \forall\ A\ n\bullet\ Arity_u\ A\ n = Fst\ (ValueOf\ (SOps\ A\ n))$
Oper	$\vdash \forall\ A\ n\bullet\ Oper\ A\ n = Snd\ (ValueOf\ (SOps\ A\ n))$
Sig	$\vdash \forall\ s\bullet\ Sig\ s = IxCompIx\ (SOps\ s)\ (\lambda\ x\bullet\ Value\ (Fst\ x))$
$\boldsymbol{Arity_i}$	$\vdash \forall\ A\ n\bullet\ Arity_i\ A\ n = ValueOf\ (A\ n)$
StructInc	$\vdash \forall\ s\ t\bullet\ s \sqsubseteq t \Leftrightarrow SCar\ s = SCar\ t \wedge SOps\ s \sqsubseteq SOps\ t$
ClosedOp	$\vdash \forall\ s\ p$
	$\bullet\ ClosedOp\ s\ p$
	$\Leftrightarrow (\forall\ f\bullet\ (\forall\ i\bullet\ i < Fst\ p \Rightarrow f\ i \in s) \Rightarrow Snd\ p\ f \in s)$
ClosedStruct	$\vdash \forall\ s$
	$\bullet\ ClosedStruct\ s$
	$\Leftrightarrow (\forall\ p\bullet\ p \in IxRan\ (SOps\ s) \Rightarrow ClosedOp\ (SCar\ s)\ p)$
pack0op	$\vdash \forall\ c\bullet\ pack0op\ c = (0, (\lambda\ is\bullet\ c))$
unpack0op	$\vdash \forall\ f\bullet\ unpack0op\ f = f\ (\epsilon\ x\bullet\ T)$
pack1op	$\vdash \forall\ f\bullet\ pack1op\ f = (1, (\lambda\ is\bullet\ f\ (is\ 0)))$
unpack1op	$\vdash \forall\ f\bullet\ unpack1op\ f = (\lambda\ x\bullet\ f\ (\lambda\ y\bullet\ x))$
pack2op	$\vdash \forall\ f\bullet\ pack2op\ f = (2, (\lambda\ is\bullet\ f\ (is\ 0)\ (is\ 1)))$
unpack2op	$\vdash \forall\ f$
	$\bullet\ unpack2op\ f$
	$= (\lambda\ x\ y\bullet\ f\ (\lambda\ z\bullet\ if\ z = 0\ then\ x\ else\ y))$
FunClosed	$\vdash \forall\ A\ f\ B$
	$\bullet\ FunClosed\ (A, f, B)$
	$\Leftrightarrow (\forall\ x\bullet\ x \in SCar\ A \Rightarrow f\ x \in SCar\ B)$
OpRespect	$\vdash \forall\ D\ f\ n\ op1\ op2$
	$\bullet\ OpRespect\ (D, f, n)\ op1\ op2$
	$\Leftrightarrow (\forall\ g$
	$\bullet\ FunImage\ g\ \{i \mid i < n\} \subseteq D$
	$\Rightarrow f\ (op1\ g) = op2\ (\lambda\ x\bullet\ f\ (g\ x)))$

HomOp $\vdash \forall\ A\ s\ B\ f$
- $HomOp\ (A,\ f,\ B)\ s$
  $\Leftrightarrow\ OpRespect$
  $(SCar\ A,\ f,\ Arity_u\ A\ s)$
  $(Oper\ A\ s)$
  $(Oper\ B\ s)$

HomOps $\vdash \forall\ A\ B\ f$
- $HomOps\ (A,\ f,\ B)$
  $\Leftrightarrow (\forall\ s\bullet\ s \in IxDom\ (SOps\ A) \Rightarrow HomOp\ (A,\ f,\ B)\ s)$

AlgHom $\vdash \forall\ A\ f\ B$
- $AlgHom\ (A,\ f,\ B)$
  $\Leftrightarrow\ Sig\ A \sqsubseteq Sig\ B$
  $\wedge\ FunClosed\ (A,\ f,\ B)$
  $\wedge\ HomOps\ (A,\ f,\ B)$

QuotientAlg
$\vdash \forall\ A\ r$
- $A\ /\ r$
  $=\ MkSTRUCT$
  $(SCar\ A\ /\ r)$
  $(IxComp\ (SOps\ A)\ (UniOpLift\ (SCar\ A,\ r)))$

VExpr $\vdash \forall\ n\bullet\ VExpr\ n = (\lambda\ a\ va\bullet\ va\ n)$

VExprs $\vdash \forall\ n\bullet\ VExprs\ n = \{p | \exists\ m\bullet\ m < n \wedge p = VExpr\ m\}$

CExprs $\vdash \forall\ sig\ es$
- $CExprs\ sig\ es$
  $= \{e$
  $|\exists\ name\ arity\ am$
  - $name \in IxDom\ sig$
    $\wedge\ sig\ name = Value\ arity$
    $\wedge\ (\forall\ i\bullet\ i < arity \Rightarrow am\ i \in es)$
    $\wedge\ e$
    $= (\lambda\ struct\ va$
    - Snd
      $(ValueOf\ (SOps\ struct\ name))$
      $(\lambda\ i\bullet\ am\ i\ struct\ va))\}$

ExprClosed $\vdash \forall\ s\ es\bullet\ ExprClosed\ s\ es \Leftrightarrow CExprs\ s\ es \subseteq es$

Exprs $\vdash \forall\ s\ n$
- $Exprs\ s\ n = \bigcap \{ps | ExprClosed\ s\ ps \wedge VExprs\ n \subseteq ps\}$

Theorems

$Arity_u$_lemma
$\vdash \forall\ d\ l\ n$
- $Arity_u\ (MkSTRUCT\ d\ (IxPack\ l))\ n$
  $=\ Fst\ (ValueOf\ (IxPack\ l\ n))$

Oper_lemma $\vdash \forall\ d\ l\ n$
- $Oper\ (MkSTRUCT\ d\ (IxPack\ l))\ n$
  $=\ Snd\ (ValueOf\ (IxPack\ l\ n))$

IxDom_Sig_thm
$\vdash \forall\ S\bullet\ IxDom\ (Sig\ S) = IxDom\ (SOps\ S)$

$\in$_IxDom_Sig_thm
$\vdash \forall\ S\ x\bullet\ x \in IxDom\ (Sig\ S) \Leftrightarrow x \in IxDom\ (SOps\ S)$

sig_arity_lemma

$\vdash \forall\ A\ n$
• $n \in IxDom\ (Sig\ A) \Rightarrow Arity_i\ (Sig\ A)\ n = Arity_u\ A\ n$

IxDom_Sig_SOps_thm
$\vdash \forall\ A \bullet\ IxDom\ (Sig\ A) = IxDom\ (SOps\ A)$

SigInc_IxDom_⊆_thm
$\vdash \forall\ A\ B \bullet\ Sig\ A \sqsubseteq Sig\ B \Rightarrow IxDom\ (Sig\ A) \subseteq IxDom\ (Sig\ B)$

SigInc_IxDom_Sops_⊆_thm
$\vdash \forall\ A\ B$
• $Sig\ A \sqsubseteq Sig\ B \Rightarrow IxDom\ (SOps\ A) \subseteq IxDom\ (SOps\ B)$

SigInc_Arity$_i$_thm
$\vdash \forall\ A\ B\ n$
• $Sig\ A \sqsubseteq Sig\ B \wedge n \in IxDom\ (Sig\ A)$
$\Rightarrow Arity_i\ (Sig\ A)\ n = Arity_i\ (Sig\ B)\ n$

SigInc_Arity$_u$_thm
$\vdash \forall\ A\ B\ n$
• $Sig\ A \sqsubseteq Sig\ B \wedge n \in IxDom\ (Sig\ A)$
$\Rightarrow Arity_u\ A\ n = Arity_u\ B\ n$

fst_packop_lemma
$\vdash (\forall\ k \bullet\ Fst\ (pack0op\ k) = 0)$
$\wedge\ (\forall\ k \bullet\ Fst\ (pack1op\ k) = 1)$
$\wedge\ (\forall\ k \bullet\ Fst\ (pack2op\ k) = 2)$

UniOpLift_pack0op_lemma
$\vdash \forall\ C\ r\ op$
• $UniOpLift\ (C, r)\ (pack0op\ op)$
$= pack0op\ (EquivClass\ (C, r)\ op)$

UniOpLift_pack1op_lemma
$\vdash \forall\ C\ r\ op$
• $UniOpLift\ (C, r)\ (pack1op\ op)$
$= pack1op\ (op\ MonOpLift\ (C, r))$

UniOpLift_pack2op_lemma
$\vdash \forall\ C\ r\ op$
• $UniOpLift\ (C, r)\ (pack2op\ op)$
$= pack2op\ (op\ DyOpLift\ (C, r))$

FunClosed_trans_thm
$\vdash \forall\ A\ f\ B\ g\ C$
• $FunClosed\ (A, f, B) \wedge FunClosed\ (B, g, C)$
$\Rightarrow FunClosed\ (A, g\ o\ f, C)$

FunClosed_FunImage_thm
$\vdash \forall\ A\ f\ B$
• $FunClosed\ (A, f, B) \Leftrightarrow FunImage\ f\ (SCar\ A) \subseteq SCar\ B$

OpRespect_pack0op_lemma
$\vdash \forall\ D\ f\ d\ c$
• $OpRespect$
$(D, f, 0)$
$(Snd\ (pack0op\ d))$
$(Snd\ (pack0op\ c))$
$\Leftrightarrow f\ d = c$

OpRespect_pack1op_lemma
$\vdash \forall\ D\ f\ d\ c$
• $OpRespect$
$(D, f, 1)$

$(Snd\ (pack1op\ d))$
$(Snd\ (pack1op\ c))$
$\Leftrightarrow (\forall\ x\bullet\ x \in D \Rightarrow f\ (d\ x) = c\ (f\ x))$

OpRespect_pack2op_lemma

$\vdash \forall\ D\ f\ \$*_d\ \$*_c$
- $OpRespect$
  $(D,\ f,\ 2)$
  $(Snd\ (pack2op\ \$*_d))$
  $(Snd\ (pack2op\ \$*_c))$
  $\Leftrightarrow (\forall\ x\ y$
  $\bullet\ x \in D\ \wedge\ y \in D \Rightarrow f\ (x\ *_d\ y) = f\ x\ *_c\ f\ y)$

HomOps_o_thm $\vdash \forall\ A\ f\ B\ g\ C$
- $HomOps\ (A,\ f,\ B)$
  $\wedge\ Sig\ A \sqsubseteq Sig\ B$
  $\wedge\ FunClosed\ (A,\ f,\ B)$
  $\wedge\ HomOps\ (B,\ g,\ C)$
  $\Rightarrow HomOps\ (A,\ g\ o\ f,\ C)$

AlgHom_o_thm $\vdash \forall\ A\ f\ B\ g\ C$
- $AlgHom\ (A,\ f,\ B)\ \wedge\ AlgHom\ (B,\ g,\ C)$
  $\Rightarrow AlgHom\ (A,\ g\ o\ f,\ C)$

QuotientAlg_Sig_lemma

$\vdash \forall\ A\ r\bullet\ Sig\ A \sqsubseteq Sig\ (A\ /\ r)$

FunClosed_EquivClass_lemma

$\vdash \forall\ A\ r\bullet\ FunClosed\ (A,\ EquivClass\ (SCar\ A,\ r),\ A\ /\ r)$

AlgHom_EquivClass_HomOps_lemma

$\vdash \forall\ A\ r$
- $AlgHom\ (A,\ EquivClass\ (SCar\ A,\ r),\ A\ /\ r)$
  $\Leftrightarrow HomOps\ (A,\ EquivClass\ (SCar\ A,\ r),\ A\ /\ r)$

A.3 The Theory lattice

Parents

unalg

Constants

$\boldsymbol{Meet_L}$	$'a\ LAT \to 'a \to 'a \to 'a$
$\boldsymbol{Join_L}$	$'a\ LAT \to 'a \to 'a \to 'a$
$\boldsymbol{Car_L}$	$'a\ LAT \to 'a\ \mathbb{P}$
$\boldsymbol{MkLAT}$	$'a\ \mathbb{P} \to ('a \to 'a \to 'a) \to ('a \to 'a \to 'a) \to 'a\ LAT$
$\boldsymbol{IsLattice}$	$'a\ LAT \to BOOL$
$\boldsymbol{QuotientLattice}$	$'a\ LAT \to ('a \to 'a \to BOOL) \to 'a\ \mathbb{P}\ LAT$
$\boldsymbol{LeLat}$	$'a\ LAT \to 'a \to 'a \to BOOL$

Aliases

/	*QuotientLattice* $: 'a\ LAT \to ('a \to 'a \to BOOL) \to 'a\ \mathbb{P}\ LAT$

Types

$'1\ \boldsymbol{LAT}$

Fixity

Right Infix 210:

$\leqslant_{\boldsymbol{L}}$ $\leqslant_{\boldsymbol{M}}$

Right Infix 235:

$\vee_{\boldsymbol{L}}$ $\vee_{\boldsymbol{M}}$

Right Infix 240:

$\wedge_{\boldsymbol{L}}$ $\wedge_{\boldsymbol{M}}$

Definitions

LAT

$\vdash \exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f$

MkLAT

Car$_L$

Join$_L$

Meet$_L$

$\vdash \forall\ t\ x1\ x2\ x3$
$\bullet\ Car_L\ (MkLAT\ x1\ x2\ x3) = x1$
$\wedge\ Join_L\ (MkLAT\ x1\ x2\ x3) = x2$
$\wedge\ Meet_L\ (MkLAT\ x1\ x2\ x3) = x3$
$\wedge\ MkLAT\ (Car_L\ t)\ (Join_L\ t)\ (Meet_L\ t) = t$

IsLattice

$\vdash \forall\ L$
$\bullet\ IsLattice\ L$
$\Leftrightarrow (\forall\ C\ \$\vee_L\ \$\wedge_L$
$\bullet\ MkLAT\ C\ \$\vee_L\ \$\wedge_L = L$

$\Rightarrow (\forall\ x\ y$
$\bullet\ x \in C \wedge y \in C$
$\Rightarrow x \vee_L y \in C$
$\wedge\ x \wedge_L y \in C$
$\wedge\ x \vee_L y = y \vee_L x$
$\wedge\ x \wedge_L y = y \wedge_L x$
$\wedge\ x \wedge_L (x \vee_L y) = x$
$\wedge\ x \vee_L x \wedge_L y = x$
$\wedge\ (\forall\ z$
$\bullet\ z \in C$
$\Rightarrow (x \vee_L y) \vee_L z = x \vee_L y \vee_L z$
$\wedge\ (x \wedge_L y) \wedge_L z$
$= x \wedge_L y \wedge_L z)))$

QuotientLattice

$\vdash \forall\ L\ \$\equiv_d$
$\bullet\ L\ /\ \$\equiv_d$
$= (let\ D = (Car_L\ L, \$\equiv_d)$
$in\ let\ \$\vee_L = Join_L\ L\ DyOpLift\ D$
$and\ \$\wedge_L = Meet_L\ L\ DyOpLift\ D$
$in\ MkLAT\ (EquivClasses\ D)\ \$\vee_L\ \$\wedge_L)$

LeLat $\vdash \forall\ L$
$\bullet\ LeLat\ L$
$= (\lambda\ x\ y\bullet \forall\ \$\vee_L\bullet\ \$\vee_L = Join_L\ L \Rightarrow x \vee_L y = y)$

Theorems

$\wedge_L$-idempot-thm

$\vdash \forall\ L$
$\bullet\ IsLattice\ L$
$\Rightarrow (\forall\ C\ \$\vee_L\ \$\wedge_L$
$\bullet\ MkLAT\ C\ \$\vee_L\ \$\wedge_L = L$
$\Rightarrow (\forall\ x\bullet\ x \in C \Rightarrow x \wedge_L x = x))$

$\vee_L$-idempot-thm

$\vdash \forall\ L$
$\bullet\ IsLattice\ L$
$\Rightarrow (\forall\ C\ \$\vee_L\ \$\wedge_L$
$\bullet\ MkLAT\ C\ \$\vee_L\ \$\wedge_L = L$
$\Rightarrow (\forall\ x\bullet\ x \in C \Rightarrow x \vee_L x = x))$

Bibliography

[1] Roger Bishop Jones. *Analyses of Analysis: Part I - Exegetical Analysis*. RBJones.com. 2009. http://www.rbjones.com/rbjpub/pp/doc/b001.pdf.

[2] Roger Bishop Jones. Miscellaneous Theory Supplements. *RBJones.com*, 2010. http://www.rbjones.com/rbjpub/pp/doc/t006.pdf.

Index