FINITE UNIVERSAL ALGEBRA: 2004 WORKSHOP ON CATEGORICAL PROGRAMMING LANGUAGES WITH AN EMPHASIS ON ALDOR

DAVID CASPERSON

Abstract. We present work in progress on writing a generic Aldor package for computations on finite universal algebras.

These are notes for a talk on Finite Universal Algebras given at the *2004 Workshop on Categorical Programming Languages, with an Emphasis on Aldor* held at the University of Cantabria, Santander Spain, July 8 and 9, 2004.

1. Definitions from Universal Algebra

In this section, I briefly introduce standard terminology from universal algebra. (See also [5][1].)

A **language** consists in part of logical connectors, quantification symbols, equations, and terms. Terms are composed from a fixed set of function symbols, and projection operators. Each **term** is either a projection operator (a variable), or a function symbol applied to a fixed non-negative number of terms. The **arity** of a function symbol is the number of terms to which it is applied.

For instance, the function symbols of the language of groups might be "$\cdot$" (arity two), "$(\ )^{-1}$" (arity one), and "$\mathbf{1}$" (arity zero).

The collection of function symbols and their associated arities form the **signature** of the language.

Fix a language. An **algebra** consists of a set called the **universe** together with an **interpretation** of the function symbols of the language into functions of the corresponding arity that act on the universe of the algebra.

For instance, in the language of groups, the set of integers together with the interpretation of "$\cdot$" as integer addition, "$(\ )^{-1}$" as negation, and "$\mathbf{1}$" as the (constant function) zero, form an algebra. The interpretation of function symbols into functions on the universe of the algebra

Date: July 30, 2004.

extends naturally to an interpretation of terms. The corresponding functions are called **term functions**.

1.1. **Classes of algebras.** Given a collection of algebras that share a language, three common methods for extending the collection are: forming **product algebras**, forming **subalgebras**, and forming **homomorphic images**. We write $\mathbf{B} \leq \mathbf{A}$ to mean that $\mathbf{B}$ is a subalgebra of $\mathbf{A}$, and we write $\mathbf{Hom}(\mathbf{C}, \mathbf{A})$ for the collection of homomorphisms from $\mathbf{C}$ to $\mathbf{A}$.

If $\mathcal{K}$ is a collection of algebras, then the collection of all possible product algebras is denoted $\mathsf{P}(\mathcal{K})$, the collection of all possible subalgebras by $\mathsf{S}(\mathcal{K})$, and the collection of all possible homomorphic images by $\mathsf{H}(\mathcal{K})$.

A collection of algebras closed under H, S, and P is called a **variety**. Birkhoff's theorem says that a variety is characterized by a set of universally quantified equations. For instance, the class of groups (which is closed under $\mathsf{H}\,\mathsf{S}\,\mathsf{P}$) is characterized by:

$$\begin{aligned}(x \cdot y) \cdot z &= x \cdot (y \cdot y)\\ x \cdot (x^{-1}) &= 1 = (x^{-1}) \cdot x\\ x \cdot 1 &= x = 1 \cdot x\end{aligned}$$

Abelian groups, rings, Boolean algebras, and lattices are other examples of varieties. Fields do not form a variety because the collection of fields is not closed under P, and there is no way to write a universally quantified equation that says that every non-zero element has a multiplicative inverse.

A collection of algebras that is closed under isomorphism, (I), S, and P, is called a **quasi-variety**. A quasi-variety is characterized by its **quasi-equations**.

1.2. **Congruences.** A homomorphism $h : \mathbf{A} \to \mathbf{B}$ induces an equivalence relation $\sim$ on $\mathbf{A}$ defined by $a_1 \sim a_2$ if and only if $h(a_1) = h(a_2)$. Such an equivalence relation is called a **congruence**. An equivalence relation $\sim$ on A is a congruence of $\mathbf{A}$ if for every k-ary term function f of $\mathbf{A}$ we have that

$$a_1 \sim b_1, \ldots a_k \sim b_k \Rightarrow f(a_1, \ldots, a_k) \sim f(b_1, \ldots, b_k).$$

The collection of congruences on $\mathbf{A}$ forms a lattice under the partial ordering induced by refinement. (That is, $\sim_1 \leq \sim_2$ iff $x \sim_1 y \Rightarrow x \sim_2$

y.) This lattice is denoted $\mathbf{Con}\,\mathbf{A}$. The bottom and top of this lattice are denoted $\Delta_{\mathbf{A}}$ and $\nabla_{\mathbf{A}}$ where $x \,\Delta\, y$ iff $x = y$, and $x \,\nabla\, y$ always holds.

2. Motivations from Universal Algebra

In this section, I list some of the areas of universal algebra where researchers would like to have tools for computations on finite universal algebras.

2.1. **Natural Duality Theory.** See [2]. Given a quasi-variety $\mathsf{ISP}(\mathbf{A})$ generated by a finite algebra $\mathbf{A}$, the dual topological class has objects whose universes are given by $\mathbf{Hom}(\mathbf{B}, \mathbf{A})$ where $\mathbf{B} \in \mathsf{SP}(\mathbf{A})$.

One computation where computer assistance is valuable is computing the subalgebras of $\mathbf{C}$ for some $\mathbf{C} \in \mathsf{P}_{\text{fin}}(A)$. Another is computing $\mathbf{Hom}(\mathbf{B}, \mathbf{A})$ for some $\mathbf{B} \leq \mathbf{C}$.

Computational assistance in computing the double duals is also valuable.

2.2. **Tame Congruence Theory.** Tame congruence theory determines structural information about a finite algebra $\mathbf{A}$ by considering cover pairs in the congruence lattice $\mathbf{Con}\,\mathbf{A}$. Each such cover pair gives rise to an associated algebra which is either:

(I) unary,
(II) a module over a ring,
(III) a semi-lattice,
(IV) a lattice, or
(V) a Boolean algebra.

See [4].

Ralph Freese and Emil Kiss have already created a "calculator" that performs many of these calculations. (See `http://www.math.hawaii.edu/~ralph/software/uaprog/index.html`.)

The Freese-Kiss calculator has efficient lattice algorithms for computing in $\mathbf{Con}\,\mathbf{A}$ for an arbitrary finite algebra $\mathbf{A}$, and can compute $\mathbf{Sg}_{\mathbf{A}}(S)$, $\mathbf{A}^k$. The Freese-Kiss calculator is written in C (Kiss) and Java (Freese). It comes with software for the graphical display of congruence lattices. However, for input and output it requires the underlying universe and term functions to be represented using the integers $0..n-1$.

2.3. **Lattice of Interpretability Types.** There exists a notion of interpreting an entire variety into another. One well known example is Boolean rings and Boolean algebras, which are interpretable into one another. See [3]. Here, one computation of interest is to search the

term functions of a given arity and of a particular algebra for one that has certain specified properties. This provides preliminary information regarding the interpretability of one variety into another. In particular information about term functions that satisfy the equations

$$x \star x = x$$
$$(x \star y) \star (w \star z) = x \star z$$

provides information about the interpretability of the equivalence class of the meet of two varieties.

3. Goals

I would like to build an Aldor library that:

(1) incorporates the calculation capabilities of the Freese-Kiss calculator.
(2) provides generators for $\mathsf{S}(\mathbf{A})$ and $\mathbf{Hom}(\mathbf{B}, \mathbf{A})$.
(3) provides a generator for the term functions $t^{\mathbf{A}}$ of an algebra $\mathbf{A}$. (For a finite algebra $\mathbf{A}$) there are at most $|A|^{|A|^k}$ term functions of arity k, even though there are infinitely many distinct terms.)
(4) allows easy *universal algebrification* of any algebra that be created with the Algebra library;
(5) provides extensions for countably infinite objects where appropriate.
(6) provides hooks for users to provide additional information or more efficient algorithms for particular algebras (for instance, the *strategy* argument to the *Homs* function on page 10).

Item 4 is important to researchers in universal algebra because researchers tend to create examples from familiar algebras whose structures they already partly understand. For this reason, it is important to be able to compute in $\mathbf{Hom}(\mathrm{Mat}_2(\mathbb{Z}_3), \mathrm{Mat}_2(\mathbb{Z}_3))$ (that is, the endomorphism monoid of the ring of two-by-two matrices over $\mathbb{Z}_3$) without being forced to compute an isometric algebra that acts on an initial segment of the natural numbers.

Item 5 is possible because Aldor generators provide a natural way to represent some countably infinite objects.

4. Existing Code

Signatures for preliminary Aldor code are provided in the Appendix. In particular, I can compute $\mathbf{Hom}(\mathbf{A}, \mathbf{B})$ for small algebras, and compute the finest congruence Θ compatible with a given equivalence

relation on a given algebra $\mathbf{A}$. Also given an algebra $\mathbf{A}$ and a set $S \subseteq A$, I can compute the smallest subalgebra $\mathbf{B} \leq \mathbf{A}$ such that $S \subseteq B$.

5. Difficulties

I wish to be able to represent algebras as having universes whose elements come from an arbitrary domain in some fixed category (currently EltType in Table 3 (page 7). This is important in being able to provide representations of elements that are useful to the user in other contexts. This means that the type of elements of the universe of $\mathbf{A}^n$ should be a function of the type of elements in the universe of $\mathbf{A}$ and of n. Unfortunately, it seems impossible to provide generic functions to compute $\mathbf{A}^n$ from $\mathbf{A}$ without running into grief from the Aldor compiler.

References

[1] Stanley Burris and H.P. Sankappanavar, *A course in universal algebra*, Graduate Texts in Mathematics, vol. 78, Springer-Verlag, 1981, out of print. See Burris' website to get a copy.

[2] David M. Clark and Brian A. Davey, *Natural duiatities for the working algebraist*, Cambridge University Press, 1998.

[3] O.C. Garcia and W. Taylor, *The lattice of interpretability types of varieties*, vol. 50, Memoirs, no. 305, American Mathematical Society, July 1984.

[4] David Hobby and Ralph McKenzie, *The structure of finite algebras*, Contemporay Mathematics, vol. 76, American Mathematical Society, 1998.

[5] Ralph N. McKenzie, George F. McNulty, and Walter Taylor, *Algebras, lattices, and varieties: Volume i*, Mathematics Series, Wadsworth and Brooks/Cole, 1987.

Department of Computer Science, University of Northern British Columbia, Prince George, BC V2N 4Z9, Canada

E-mail address: casper@unbc.ca

Appendix A. Aldor Software

Table 1. Equivalence relations.

```
EquivalenceType (Elt:PrimitiveType) : Category == CopyableType
with {
  delta      : ()              -> %; ++ create new identity relation
  nabla      : Generator Elt   -> %; ++ unite every element generated
  principal  : (Elt,Elt)       -> %; ++ principal equivalence
                                     ++   generated by a pair
  principal  : Generator Cross (Elt,Elt) -> %;
                                     ++ equivalence generated by a
                                     ++   list of pairs
  fromClasses : List List Elt ->  %;
                                     ++ from [[1,3],[2,4]] form ;
  basis      : % -> Generator Cross(Elt,Elt) ;
                                     ++ the inverse of principal in
                                     ++ the sense that
                                     ++ principal(basis x)=x.

  meet       : (%,%)           -> %; ++ standard lattice operation
  join       : (%,%)           -> %; ++ standard lattice operation
  restrictionOf? : (%,%)->Boolean ;++ first partition is finer
                                     ++   than the second
  unite!     : (%,Elt,Elt)   -> Boolean ;
                                     ++ unite two elements. return true iff
                                     ++ they were not united before
  united?    : (%,Elt,Elt)   -> Boolean ;
                                     ++ the same as repr(me,x1)=repr(me,x2)
  repr       : (%,Elt)  -> Elt ;     ++ canonical element in the class
  count      : (%,Elt)  -> MachineInteger ;
                                     ++ size of the equivalence class
  bigClasses : %-> Generator Elt ;
                                     ++ all classes with more than
                                     ++   one element
  classes    :(%, Generator Elt) -> Generator Elt;
                                     ++ like (repr(me,x) for x in g)
                                     ++ but generates each class once
  if Elt has OutputType then OutputType ;
}
```

TABLE 2. Equivalence relations continued.

```
HashEquivalenceType(Base:PrimitiveType) : Category
        == Join(HashType, EquivalenceType(Base)) ;

HASHEQUIVALENCE (BASE:PrimitiveType,h:BASE->MI)
        : HashEquivalenceType(Base) ;
```

TABLE 3. Elements

```
EltType : Category == Join(PrimitiveType,OutputType) with {
    card : MACHINEINTEGER ;
    asInt: % -> MACHINEINTEGER ;
    +    : MACHINEINTEGER -> % ;
    all  : () -> GENERATOR % ;
}
```

The following category is intended for use as a carrier for product algebras. There seem to be problems when actually attempting to create algebras with carriers of this type.

```
powerType(ELT:EltType) : Category == EltType with {
  BFLST ==> BoundedFiniteLinearStructureType ;
  base     : EltType;
  expt     : MACHINEINTEGER ;
  bracket  : TUPLE ELT              -> % ;
  bracket  : LIST ELT               -> % ;
  powerElt : GENERATOR ELT          -> % ;
  proj     : MACHINEINTEGER         -> (% -> ELT) ;
  as       : (%,COLLECTION:BFLST Elt) -> COLLECTION
}
```

TABLE 4. Functions

```
FunctionType (dom:EltType) : Category == with {
    size  : %                -> MachineInteger ;
    type  : %                -> FunctionType dom ;
    domain: %                -> EltType ;
    arity : %                -> MachineInteger ;
    apply : (%,List dom) -> dom ;
}

Function (Values:EltType) : FunctionType(Values) with {
  MI ==> MachineInteger ;
  make : (size:MI,arity:MI,fn:List Values->Values) -> % ;
  make : (size:MI,arity:MI,fn:Array Values)        -> % ;
  make : (size:MI,g:Values -> Values)              -> % ;
  make : (size:MI,g2:(Values,Values)->Values)      -> % ;
} ;
```

TABLE 5. Algebras

```
FiniteAlgebraType(CARRIER:EltType) : Category == with {
  MI ==> MachineInteger;

  -- constructors
  algebra   : (MI,List Function Carrier) -> % ;
  subAlgebra: (%, Set Carrier)           -> % ; -- no check for closed
  -- power  : (%, n:MI)                  -> dependency problems!!!
  factor    : (%, h:Carrier->MI,HashEquivalence(Carrier,h)) -> % ;

  -- properties relating to the underlying universe
  size      : %                          -> MI ;
  generator : %                          -> Generator Carrier ;
  univ      : %                          -> Set Carrier ;

  -- properties related to the type and language
  signature : %                          -> List MI ;
  fns       : %                          -> Generator Function Carrier ;
  kthFn     : (%,MI)                     -> Function Carrier ;

  --- algebraic closure operations
  sGuniv    : (%, Set Carrier)           -> Set Carrier ;
  sG        : (%, Set Carrier)           -> % ;
  congruence: (%, h:Carrier->MI, HashEquivalence (Carrier,h))
              -> HashEquivalence (Carrier,h) ;
                      ++ compute the minimal congruence on this
                      ++ algebra that respects the equivalence
                      ++ relation given.

  export from Function Carrier ;
}

Algebra (Carrier : EltType) : FiniteAlgebraType Carrier ;
```

TABLE 6. Homomorphisms

```
HomType (DomainType:EltType, CodomainType:EltType) : Category == _
 Join(CopyableType,OutputType) with {
    ALG ==> Algebra ;
    Gen ==> Generator ;
    DoT ==> DomainType ;
    CoT ==> CodomainType ;

    apply    : (%,DoT)           -> CoT ;
    domain   : %                 -> Gen DoT ;
    domain   : %                 -> Set DoT ;
    domain   : %                 -> ALG DoT ;
    codomain : %                 -> Gen CoT ;
    codomain : %                 -> Set CoT ;
    codomain : %                 -> ALG CoT ;
    range    : %                 -> Gen CoT ;

    empty    : (ALG DoT,ALG CoT) -> % ;
}

Hom (DT:EltType,CDT:EltType) : HomType (DT,CDT) with {
    set!          : (%,DT,CDT) -> % ;
    extend!       : (%,DT,CDT) -> % ;
    partialDomain : %          -> Generator DT ;
} ;

Homs : (DT:EltType,CDT:EltType)
       (A:Algebra DT,
        B:Algebra CDT,
        strategy:(Algebra DT, Algebra  DT, Hom(DT,CDT))->DT
           ==DefaultStrategy(DT,CDT)
       )
    -> Generator (Hom (DT,CDT))   ;
```