An Implementation of Metarouting using Coq

Vilius Naudžiūnas
Computer Laboratory
University of Cambridge
Vilius.Naudziunas@cl.cam.ac.uk

Timothy G. Griffin
Computer Laboratory
University of Cambridge
Timothy.Griffin@cl.cam.ac.uk

***Abstract*—We describe work-in-progress on a tool called the Metarouting Environment (MrE, pronounced "Mr. E" or "mystery"). MrE implements a domain-specific language of Combinators for Algebraic Structures (CAS).**

The main focus of MrE is on the specification and solution of path problems in weighted graphs — *routing* in networks. A user of MrE constructs expressions in CAS that are translated into implementations of algebraic structures comprised of data types (for path weights and policy) and operations (for path selection and policy application). The user can then construct weighted graphs over these algebraic structures and solve various path problems using a fixed menu of graph algorithms (including a generalized Dijkstra's algorithm).

The correct application of a selected algorithm may require that specific algebraic properties hold for the structure used. For example, the generalized Dijkstra's algorithm requires a total order on path weights. A key feature of the design of CAS is that the algebraic properties required for the correct use of each algorithm are automatically verified or refuted.

The core of MrE has been implemented using the Coq theorem prover. However, a user of MrE is running code that has been *extracted* from our Coq libraries. In this way MrE users do not run Coq directly or need to know any of the Coq-related implementation details.

I. Introduction

We describe an implementation of Metarouting [7] embodied in a tool called MrE which provides users with a domain-specific language of Combinators for Algebraic Structures (CAS). Our goal in the design of CAS is to provide programmers with a high-level declarative language for implementing *path metrics*, one important component of any routing protocol. The benefit of using CAS is that all of the algebraic properties required for the correct use of routing protocols (Section II) are automatically verified or refuted.

Here we present a small fragment of CAS together with a demonstration of MrE (in Section III). The core of MrE was implemented using the Coq theorem prover [2] (Section IV). However, MrE users do not need to know to use Coq since MrE is implemented with Ocaml code *extracted* from our Coq libraries [9]. MrE can be downloaded from http://www.cl.cam.ac.uk/~tgg22/metarouting.

II. Instantiating generic algorithms

The classical approach to shortest paths in a graph uses an algebraic structure $(\mathbb{N} \cup \{\infty\}, \min, +)$, where $\min$ is used to select best path weights and $+$ is used to compute the weight of a path from its arc weights. Many of the algorithms used to find shortest paths (Dijkstra's algorithm, Bellman-Ford) can be generalized to work with other algebraic structures. For example, the structure $(\mathbb{N} \cup \{\infty\}, \max, \min)$ can be used to find paths of maximum capacity (bandwidth, for example).

This leads to the study of *path algebras* (or *semirings*), structures of the form $(S, \oplus, \otimes)$ which can be seen as generalizing the ring $(\mathbb{R}, +, \times)$ from linear algebra. A lot of effort has gone into the identification of algebraic properties required for the correct usage of specific algorithms. The reader should consult [5] for a wide-ranging survey of work in this area spanning the last forty years. A more recent book [1] highlights applications to network routing.

We list some typical properties required by common path algorithms in Figure 1. Space does not allow us to describe the details of any algorithm, but we list four algorithms in Figure 2 together with algebraic properties required for their correct use with structures of the form $(S, \oplus, \otimes)$.

The first two algorithms are classical methods used with semirings — they find *globally optimal* path weights in a graph. That is, the best path weights over all possible paths in a graph. Distributivity properties are critical for obtaining global optimality.

Perhaps the most novel aspect of some recent work on Internet routing is understanding that it is still possible to route with some metrics that are not distributive [14], [13]. This work has grown out of an analysis of the Border Gateway Protocol where the notion of *local optimality* makes sense. By local optimality we mean that an equilibrium is reached where each node obtains the best path that it can, given the paths its neighbors have obtained.

The problem. The difficulty now is in *instantiating* such generic algorithms with implementations of algebras that meet the requirements. For complicated algebras, this can be a very difficult task. The idea behind Metarouting and the design of CAS is that much of this tedium can be automated.

III. MrE demonstration

In this section we demonstrate some features of the MrE tool and present a small subset of CAS. The grammar for expressions in our CAS fragment is

$$
\begin{array}{rcl}
E & ::= & \texttt{bNatMinPlus} \\
 & | & \texttt{bNatMaxMin} \\
 & | & \texttt{bAddOne}\ c\ E \\
 & | & \texttt{bAddZero}\ c\ E \\
 & | & \texttt{bLex}\ E\ E \\
 & | & \texttt{bSelLex}\ E\ E
\end{array}
$$

description	notation	context	meaning
Associativity	$\mathsf{ass}(S, \circ)$		$\forall x\ y\ z \in S, x \circ (y \circ z) = (x \circ y) \circ z$
Commutativity	$\mathsf{com}(S, \circ)$		$\forall x\ y \in S, x \circ y = y \circ x$
Idempotence	$\mathsf{idm}(S, \circ)$		$\forall x \in S, x \circ x = x$
Selectivity	$\mathsf{sel}(S, \circ)$		$\forall x\ y \in S, x \circ y \in \{x, y\}$
Identity	$\mathsf{ide}(S, \circ)$		$\exists i \in S, \forall x \in S, i \circ x = x = x \circ i$
Annihilator	$\mathsf{ann}(S, \circ)$		$\exists w \in S, \forall x \in S, w \circ x = w = x \circ w$
Consistency	$\mathsf{con}(S, \circ, \star)$	$\mathsf{ide}(S, \circ) \wedge \mathsf{ann}(S, \star)$	$\mathcal{W}(\mathsf{ide}(S, \circ)) = \mathcal{W}(\mathsf{ann}(S, \star))$
Left absorbing	$\mathsf{abs}(S, \circ, \star)$		$\forall x\ y \in S, x \circ (y \star x) = x$
Left strict absorbing	$\mathsf{str}(S, \circ, \star)$		$\forall x\ y \in S, x \circ (y \star x) = x \wedge x \neq y \star x$
Left distributivity	$\mathsf{l.d}(S, \circ, \star)$		$\forall x\ y\ z \in S, z \star (x \circ y) = (z \star x) \circ (z \star y)$
Right distributivity	$\mathsf{r.d}(S, \circ, \star)$		$\forall x\ y\ z \in S, (x \circ y) \star z = (x \star z) \circ (y \star z)$

Fig. 1. The key algebraic properties. $\circ$ and $\star$ are binary operators. The *witness* associated with an existential property $\exists x \in S, P(x)$, denoted $\mathcal{W}(\exists x \in S, P(x))$ represents an element $s \in S$ such that $P(s)$ holds. If we have a proof of $\exists x \in S, P(x)$, then the witness is always defined since we are working in a constructive logic. The **context** is simply a formula that must be true in order for a particular property to make sense.

algorithm	citation	additional requirements	optimality
Matrix iteration	[5], [1]	$\mathsf{l.d}(S, \oplus, \otimes)$, $\mathsf{r.d}(S, \oplus, \otimes)$	global
Dijkstra's Algorithm	[5], [1]	$\mathsf{l.d}(S, \oplus, \otimes)$, $\mathsf{r.d}(S, \oplus, \otimes)$, $\mathsf{sel}(S, \oplus)$	global
Path vector	[14]	$\mathsf{str}(S, \oplus, \otimes)$	local
Dijkstra's Algorithm	[13]	$\mathsf{abs}(S, \oplus, \otimes)$, $\mathsf{sel}(S, \oplus)$	local

Fig. 2. A selection of path algorithms. The **common requirements** that are shered between these algorithms are properties $\mathsf{ass}(S, \oplus)$, $\mathsf{ass}(S, \otimes)$, $\mathsf{com}(S, \oplus)$, $\mathsf{idm}(S, \oplus)$, $\mathsf{ide}(S, \oplus)$, $\mathsf{ann}(S, \oplus)$, $\mathsf{ide}(S, \otimes)$, $\mathsf{ann}(S, \otimes)$, $\mathsf{con}(S, \oplus, \otimes)$, and $\mathsf{con}(S, \otimes, \oplus)$.

where c represents constants supplied by the user. The semantics of expressions will be described as we walk through an MrE session. An expression may not be *well formed*, in which case an error is generated. Otherwise, the semantics of an expression E is a structure of the form

$$[\![E]\!] = (S,\ \oplus,\ \otimes),$$

where both $\oplus$ and $\otimes$ are guaranteed to be associative. We call such a structure a *bisemigroup*, which accounts for the "b" prefix on each constructor.

Using MrE in the interactive mode we get a command-line prompt.

```
MrE>
```

We can then enter various commands, including those of the form `id <- expression`, which binds the identifier `id` to the algebraic structure denoted by `expression`.

Let's try that with a simple expression, `bNatMaxMin`, with semantics

$$[\![\texttt{bNatMaxMin}]\!] = (\mathbb{N},\ \max,\ \min),$$

where $\max$ is the additive operator and $\min$ is the multiplicative operator.

```
MrE> max_min <- bNatMaxMin
```

```
------- additive properties -------------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           TRUE  : 0
Annihilator        FALSE : -
------- multiplicative properties --------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           FALSE : -
Annihilator        TRUE  : 0
------- bisemigroup properties -----------
Consistency(+,*)   TRUE  : -
Consistency(*,+)   IRRELEVANT
L absorbing        TRUE  : -
L strict absorbing FALSE : 0, 0
L distributivity   TRUE  : -
R distributivity   TRUE  : -
```

MrE automatically prints the algebraic properties (Figure 1) of this structure. In fact, MrE can display over 50 properties that it associates with bisemigroups, but for space reasons we

show only outputs corresponding to Figure 1.

Consider the property of **Identity** for the additive component of this structure:

$$\exists i \in \mathbb{N}, \forall x \in \mathbb{N}, i \max x = x = x \max i.$$

This has the witness 0, which is displayed next to the `TRUE` indicating that **Identity** holds. The property **Left strict absorbing** for this structure is

$$\forall x\ y \in \mathbb{N}, x \max (y \min x) = x \wedge x \neq y \min x,$$

which is FALSE. MrE has converted the negation of this property into the existentially quantified formula

$$\exists x\ y \in \mathbb{N}, x \max (y \min x) \neq x \vee x = y \min x,$$

and the displayed pair $0,\ 0$ are witnesses for this fact. That is, whenever a property holds or is refuted, and the corresponding formula can be expressed with a sequence of existential quantifiers, then MrE will construct witnesses.

Finally, the **Consistency** of $(\min, \max)$, that is $\mathsf{con}(S, \otimes, \oplus)$ for this structure, is *irrelevant* since the context of this property does not hold.

Next, the expression `bNatMinPlus` has semantics

$$[\![\texttt{bNatMinPlus}]\!] = (\mathbb{N},\ \min,\ +).$$

Let us check its properties.

```
MrE> min_plus <- bNatMinPlus

------- additive properties -------------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           FALSE : -
Annihilator        TRUE  : 0
------- multiplicative properties --------
Commutativity      TRUE  : -
Idempotence        FALSE : 1
Selectivity        FALSE : 1, 1
Identity           TRUE  : 0
Annihilator        FALSE : -
------- bisemigroup properties -----------
Consistency(+,*)   IRRELEVANT
Consistency(*,+)   TRUE  : -
L absorbing        TRUE  : -
L strict absorbing FALSE : 0, 0
L distributivity   TRUE  : -
R distributivity   TRUE  : -
```

Before we can directly use either of these algebras with one of our algorithms we need to add some identities and annihilators. If $\circ$ is a binary operator over set S, then $\mathrm{id}\ c \circ$ is the binary operator over $S \uplus \{c\}$ defined as

$$\begin{aligned} \mathrm{inr}(c) \bullet x &= x, \\ x \bullet \mathrm{inr}(c) &= x, \\ \mathrm{inl}(s_1) \bullet \mathrm{inl}(s_2) &= \mathrm{inl}(s_1 \circ s_2). \end{aligned}$$

where $\bullet = \mathrm{id}\ c \circ$. Here the notation $S \uplus T$ represents the set $\{\mathrm{inl}(s) \mid s \in S\} \cup \{\mathrm{inr}(t) \mid t \in T\}$ — that is, a concrete implementation of the set $S \cup T$ where each element is tagged to indicate from which set it came.

In a similar way, $\mathrm{ann}\ c \circ$ adds an annihilator, and is defined as

$$\begin{aligned} \mathrm{inr}(c) \star x &= \mathrm{inr}(c), \\ x \star \mathrm{inr}(c) &= \mathrm{inr}(c), \\ \mathrm{inl}(s_1) \star \mathrm{inl}(s_2) &= \mathrm{inl}(s_1 \circ s_2). \end{aligned}$$

where $\star = \mathrm{ann}\ c \circ$.

An MrE user does not have direct access to these constructions. Rather, they are "packaged" in combinators that operate on bisemigroups:

$$\begin{aligned} [\![\texttt{bAddOne}\ c\ E]\!] &= (S \uplus \{c\},\ \mathrm{ann}\ c \oplus_S,\ \mathrm{id}\ c \otimes_S) \\ &\quad \text{where } [\![E]\!] = (S,\ \oplus_S,\ \otimes_S) \\ [\![\texttt{bAddZero}\ c\ E]\!] &= (S \uplus \{c\},\ \mathrm{id}\ c \oplus_S,\ \mathrm{ann}\ c \otimes_S) \\ &\quad \text{where } [\![E]\!] = (S,\ \oplus_S,\ \otimes_S) \end{aligned}$$

Using these definitions, we see that we need to apply `bAddOne` to `max_min` and `bAddZero` to `min_plus`.

```
MrE> bw <- bAddOne INF max_min

------- additive properties -------------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           TRUE  : inl 0
Annihilator        TRUE  : inr INF
------- multiplicative properties --------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           TRUE  : inr INF
Annihilator        TRUE  : inl 0
------- bisemigroup properties -----------
Consistency(+,*)   TRUE  : -
Consistency(*,+)   TRUE  : -
L absorbing        TRUE  : -
L strict absorbing FALSE : inr INF, inr INF
L distributivity   TRUE  : -
R distributivity   TRUE  : -

MrE> sp <- bAddZero INF min_plus

------- additive properties -------------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           TRUE  : inr INF
Annihilator        TRUE  : inl 0
------- multiplicative properties --------
Commutativity      TRUE  : -
Idempotence        FALSE : inl 1
```

```
Selectivity         FALSE : inl 1, inl 1
Identity            TRUE  : inl 0
Annihilator         TRUE  : inr INF
------- bisemigroup properties -----------
Consistency(+,*)    TRUE  : -
Consistency(*,+)    TRUE  : -
L absorbing         TRUE  : -
L strict absorbing FALSE : inr INF, inr INF
L distributivity    TRUE  : -
R distributivity    TRUE  : -
```

Either of these structures can now be used to label graphs and solve "shortest path" problems using an algorithm such as Dijkstra's. Due to lack of space we do not show here the details of creating graphs or running algorithms in MrE.

Now suppose we want to construct an algebra with path metrics of the form (d, b), where d is from the `sp` algebra and b is from the `bw`. In addition, we want distance d to be more significant than bandwidth — only using better bandwidth as a tie breaker for paths of the same length. This brings us to the lexicographic product.

We describe this construction in full generality. Assume $\circ$ is a binary operator over set S, and $\diamond$ is a binary operator over set T. Their *direct product*, $\circ \times \diamond$, is defined to be the binary operator over $S \times T$

$$(s_1, t_1) \bullet (s_2, t_2) = (s_1 \circ s_2, t_1 \diamond t_2).$$

where $\bullet = \circ \times \diamond$.

The *lexicographic product*, $\circ \vec{\times} \diamond$, is defined as follows.

$$(s_1, t_1) \bullet (s_2, t_2) = \begin{cases} (s_1, t_1 \diamond t_2), & \text{if } s_1 = s_2 \\ (s_1, t_1), & \text{if } s_1 = (s_1 \circ s_2) \neq s_2 \\ (s_2, t_2), & \text{if } s_1 \neq (s_1 \circ s_2) = s_2 \\ (s_1 \circ s_2, 1_\diamond), & \text{if } s_1 \neq (s_1 \circ s_2) \neq s_2 \end{cases}$$

where $\bullet = \circ \vec{\times} \diamond$. For the lexicographic product, $1_\diamond \in T$ denotes an identity for T, if it exists, and this will be enforced by the *wellformedness* rules for this constructor.

$$\begin{aligned} [\![\texttt{bLex}\ E\ E']\!] \quad = \quad & (S \times T,\ \oplus_S \vec{\times} \oplus_T,\ \otimes_S \times \otimes_T) \\ & \text{where } [\![E]\!] = (S,\ \oplus_S,\ \otimes_S) \\ & \quad \text{and } [\![E']\!] = (T,\ \oplus_T,\ \otimes_T) \end{aligned}$$

Let us now combine `sp` with `bw` using the `bLex` combinator:

```
MrE> lex_sp_bw <- bLex sp bw

------- additive properties -------------
Commutativity       TRUE  : -
Idempotence         TRUE  : -
Selectivity         TRUE  : -
Identity            TRUE  : (inr INF, inl 0)
Annihilator         TRUE  : (inl 0, inr INF)
------- multiplicative properties --------
Commutativity       TRUE  : -
Idempotence         FALSE : (inl 1, inr INF)
Selectivity         FALSE : (inl 1, inr INF),
                            (inl 1, inr INF)
Identity            TRUE  : (inl 0, inr INF)
Annihilator         TRUE  : (inr INF, inl 0)
------- bisemigroup properties -----------
Consistency(+,*)    TRUE  : -
Consistency(*,+)    TRUE  : -
L absorbing         TRUE  : -
L strict absorbing FALSE :
                            (inr INF, inr INF),
                            (inr INF, inr INF)
L distributivity    FALSE :
                            (inl 0, inl 0),
                            (inr INF, inr INF),
                            (inr INF, inr INF)
R distributivity    FALSE :
                            (inl 0, inl 0),
                            (inr INF, inr INF),
                            (inr INF, inr INF)
```

Note that the distributivity properties *fail* for this algebra! Looking at **Left distributivity**, we see three witnesses for the negation of this property. Writing this in a mathematical notation, the negated property has the form

$$\begin{aligned} & \exists (x_1, x_2)\ (y_1, y_2)\ (z_1, z_2) \in (\mathbb{N} \uplus \{\text{INF}\}) \times (\mathbb{N} \uplus \{\text{INF}\}), \\ & \quad (x_1, x_2) \otimes ((y_1, y_2) \oplus (z_1, z_2)) \\ & \quad \neq ((x_1, x_2) \otimes (y_1, y_2)) \oplus ((x_1, x_2) \otimes (z_1, z_2)) \end{aligned}$$

where

$$\begin{aligned} \oplus \quad &= \quad (\text{id INF min}) \vec{\times} (\text{ann INF max}) \\ \otimes \quad &= \quad (\text{ann INF} +) \times (\text{id INF min}). \end{aligned}$$

With a bit of patience we can check the counterexamples provided (writing 0 for $\text{inl}(0)$ and ∞ for $\text{inr}(\text{INF})$):

$$(\infty, \infty) \otimes ((0, 0) \oplus (\infty, \infty)) = (\infty, \infty) \otimes (0, 0) = (\infty, 0),$$

and

$$\begin{aligned} & ((\infty, \infty) \otimes (0, 0)) \oplus ((\infty, \infty) \otimes (\infty, \infty)) \\ & = (\infty, 0) \oplus (\infty, \infty) = (\infty, \infty). \end{aligned}$$

There is something else "wrong" with this algebra. If we want to interpret ∞ as the length of a non-existing path, then elements like (∞, b) are not useful. Perhaps we have invoked `bAddOne` and `bAddZero` too early in the construction of our algebra. Let us try combining the basic algebras first.

```
MrE> lex_min_plus_max_min
                    <- bLex min_plus max_min

------- additive properties -------------
Commutativity       TRUE  : -
Idempotence         TRUE  : -
Selectivity         TRUE  : -
Identity            FALSE : -
Annihilator         FALSE : -
------- multiplicative properties --------
```

```
Commutativity      TRUE  : -
Idempotence        FALSE : (1, 0)
Selectivity        FALSE : (1, 0) , (1, 0)
Identity           FALSE : -
Annihilator        FALSE : -
------- bisemigroup properties -----------
Consistency(+,*)   IRRELEVANT
Consistency(*,+)   IRRELEVANT
L absorbing        TRUE  : -
L strict absorbing FALSE : (0, 0) , (0, 0)
L distributivity   TRUE  : -
R distributivity   TRUE  : -
```

Now that we have distributivity we only need to add identities and annihilators!

```
MrE> lex_sp_bw_v2 <- bAddOne NIL
       (bAddZero INF lex_min_plus_max_min)

------- additive properties -------------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           TRUE  : inl inr INF
Annihilator        TRUE  : inr NIL
------- multiplicative properties --------
Commutativity      TRUE  : -
Idempotence        FALSE : inl inl (1, 0)
Selectivity        FALSE : inl inl (1, 0),
                           inl inl (1, 0)
Identity           TRUE  : inr NIL
Annihilator        TRUE  : inl inr INF
------- bisemigroup properties -----------
Consistency(+,*)   TRUE  : -
Consistency(*,+)   TRUE  : -
L absorbing        TRUE  : -
L strict absorbing FALSE : inr NIL, inr NIL
L distributivity   TRUE  : -
R distributivity   TRUE  : -
```

We have now defined an algebraic structure that allows us to solve for globally optimal paths algorithms such as matrix iteration or Dijkstra's algorithm.

Let us see what happens if we switch the order of the algebras `min_plus` and `max_min`.

```
MrE> lex_max_min_min_plus <-
                   bLex max_min min_plus

  Error : min_plus does not have a
          multiplicative identity.
```

This error indicates the expression is not well formed because `bLex` expects the second argument to have a multiplicative identity. We could have used another form of lexicographic product, `bSelLex`, which has the same semantics as `bLex` but requres its first argument to be selective (and so the fourth clause in the definition of the lexicographic operation is never used).

```
MrE> slex_max_min_min_plus <-
                bSelLex max_min min_plus

------- additive properties -------------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           FALSE : -
Annihilator        FALSE : -
------- multiplicative properties --------
Commutativity      TRUE  : -
Idempotence        FALSE : (0, 1)
Selectivity        FALSE : (0, 1), (0, 1)
Identity           FALSE : -
Annihilator        FALSE : -
------- bisemigroup properties -----------
Consistency(+,*)   IRRELEVANT
Consistency(*,+)   IRRELEVANT
L absorbing        TRUE  : -
L strict absorbing FALSE : (0, 0), (0, 0)
L distributivity   FALSE : (1, 1),
                           (0, 0),
                           (0, 1)
R distributivity   FALSE : (1, 1),
                           (0, 0),
                           (0, 1)
```

Now we are ready to apply `bAddOne` and `bAddZero`.

```
MrE> slex_bw_sp <- bAddOne NIL
          (bAddZero INF slex_max_min_min_plus)

------- additive properties -------------
Commutativity      TRUE  : -
Idempotence        TRUE  : -
Selectivity        TRUE  : -
Identity           TRUE  : inl inr INF
Annihilator        TRUE  : inr NIL
------- multiplicative properties --------
Commutativity      TRUE  : -
Idempotence        FALSE : inl inl (0, 1)
Selectivity        FALSE : inl inl (0, 1),
                           inl inl (0, 1)
Identity           TRUE  : inr NIL
Annihilator        TRUE  : inl inr INF
------- bisemigroup properties -----------
Consistency(+,*)   TRUE  : -
Consistency(*,+)   TRUE  : -
L absorbing        TRUE  : -
L strict absorbing FALSE : inr NIL,
                           inr NIL
L distributivity   FALSE : inl inl (1, 1),
```

```
                                inl inl (0, 0),
                                inl inl (0, 1)
R distributivity    FALSE :     inl inl (1, 1),
                                inl inl (0, 0),
                                inl inl (0, 1)
```

Since this algebra is absorbing we can use it with a version of Dijkstra's algorithm [13] to find locally optimal paths.

A. Advanced constructions

Besides bisemigroups, our language of combinators includes other types of algebraic structures:

name	signature	prefix
Sets	(S)	d
Semigroups	$(S, \oplus)$	s
Preorders	$(S, \leq)$	p
Transforms	$(S, L, \triangleright)$	t
Order semigroups	$(S, \leq, \oplus)$	o
Bisemigroups	$(S, \oplus, \otimes)$	b
Order transforms	$(S, L, \leq, \triangleright)$	ot
Semigroup transforms	$(S, L, \oplus, \triangleright)$	st

Here sets are represented by S and L, $\oplus$ and $\otimes$ are associative binary operations, $\leq$ is a preorder (reflexive and transitive relation), and $\triangleright$ is an operation with type $L \rightarrow S \rightarrow S$. This last kind of operation is used instead of $\otimes$ when the path weights (in S) are distinct from arc labels (in L). The interested reader can consult [8] for examples.

To give a hint of the expressive power of the CAS, we are able to express a semiring defined by Martelli [10] that calculates all minimal cutsets between any two nodes in a directed graph. For more details, see [12].

IV. Coq-based implementation

We have implemented the core of MrE using the Coq theorem prover [2]. Coq is an interactive theorem prover with a large user community that has used it for the development of formal mathematics [4], [6] and formally verified programs [11], [3]. Coq uses a constructive logic, so many proofs can be given a direct computational interpretation.

Our use of Coq centers around proving theorems that allow us to verify or refute algebraic properties such as those of Figure 1. Suppose C is an n-ary combinator (such as `bLex`, a 2-ary combinator) and $\vec{a} = a_1, \ldots, a_n$ represent arguments to C. For each property P (such as **Left distributivity**) we attempt to formulate a result of the form

$$\pi_{P,C}(\vec{a}) \Rightarrow (P(C(\vec{a})) \Leftrightarrow \beta_{P,C}(\vec{a}))$$

where $\beta_{P,C}$ and $\pi_{P,C}$ are boolean expressions of properties over the algebras of $\vec{a}$. The formula $\pi_{P,C}$ determines when it is legal to use the C combinator, and under that condition $\beta_{P,C}$ provides necessary and sufficient conditions for asserting P for the construction $C(\vec{a})$. In fact, we reformulate each result as two Coq theorems of the form

$$\begin{aligned} \pi_{P,C}(\vec{a}) \wedge \beta_{P,C}(\vec{a}) &\Rightarrow P(C(\vec{a})) \\ \pi_{P,C}(\vec{a}) \wedge \neg\beta_{P,C}(\vec{a}) &\Rightarrow \neg P(C(\vec{a})), \end{aligned}$$

where we have rewritten the form of both $\neg\beta_{P,C}$ and $\neg P$ to expose their constructive content. These two theorems then give us a way of extracting P or $\neg P$ in a bottom-up manner for any expression constructed with C. Often the formulas $\beta_{P,C}$ and P require the addition of a new property Q, which is then added to our "working set" of properties. This then requires futher rules for reasoning about Q for all constructors. This process has led to a large working set of properties (currently around 150), and several thousand theorems proved in Coq. Again, see [12] for details.

Fortunately, the MrE user does not need to know about this underlying complexity! One of the main goals of the design of CAS and MrE is to automate the tedium of this kind of theorem proving — we have done the theorem proving at *design time* of CAS. A user of MrE is actually running code that has been *extracted* from our Coq libraries using the features of [9]. In this way MrE users do not run Coq directly or need to know any of the Coq-related implementation details.

Acknowledgment

The first author thanks the Engineering and Physical Sciences Research Council (EPSRC) for a Doctoral Training Grant, and the second author thanks EPSRC for support under grant EP/F002718/1. Both authors would like to thank Arthur Azevedo de Amorim, Alexander Gurney, Philip Taylor, and the W-RiPE reviewers for their helpful comments.

References

[1] John S. Baras and George Theodorakopoulos. *Path problems in networks*. Morgan & Claypool, 2010.
[2] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer-Verlag New York Inc, 2004.
[3] S. Blazy and X. Leroy. Mechanized semantics for the Clight subset of the C language. *Journal of Automated Reasoning*, 43(3):263–288, 2009.
[4] L. Cruz-Filipe, H. Geuvers, and F. Wiedijk. C-corn, the constructive coq repository at nijmegen. In *Mathematical Knowledge Management*, pages 88–103. Springer, 2004.
[5] M. Gondran and M. Minoux. Graphs, Dioids and Semirings: New Models and Algorithms (Operations Research/Computer Science Interfaces Series). 2008.
[6] G. Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.
[7] T.G. Griffin and J.L. Sobrinho. Metarouting. *ACM SIGCOMM Computer Communication Review*, 35(4):1–12, 2005.
[8] A.J.T. Gurney and T.G. Griffin. Lexicographic products in metarouting. In *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, pages 113–122. IEEE, 2007.
[9] P. Letouzey. A new extraction for coq. *Types for proofs and programs*, pages 617–617, 2003.
[10] A. Martelli. *An application of regular algebra to the enumeration of cut sets in a graph*. 1974.
[11] A. Nanevski, G. Morrisett, A. Shinnar, P. Govereau, and L. Birkedal. Ynot: Reasoning with the awkward squad. In *ACM SIGPLAN International Conference on Functional Programming*. Citeseer, 2008.
[12] Vilius Naudžiūnas. Formal specification language for path algebras. PhD dissertation, University of Cambridge, forthcoming.
[13] J.L. Sobrinho and T.G. Griffin. Routing in equilibrium. *Mathematical Theory of Networks and System*, 2010.
[14] Joao Luis Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Transactions on Networking*, 13(5):1160–1173, October 2005.