A Domain-Specific Language for the Specification of Path Algebras

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

Path algebras are used to describe path problems in directed graphs. Constructing a new path algebra involves defining a carrier set, several operations, and proving that many algebraic properties hold. We describe work-in-progress on the development of a domain-specific language for specifying path algebras where implementations and proofs are *automatically* constructed in a bottom-up fashion. Our initial motivation came from the development of Internet routing protocols, but we believe that the approach could have much wider applications. We have implemented the language using the Coq theorem prover.

1 Introduction

Finding shortest paths in a graph is one of the fundamental problems in computer science. Algorithms for solving shortest path problems, such as Dijkstra's or Bellman-Ford, are widely used. Generalizations of shortest paths to semirings and related structures has been an active area of research for over forty years (see [2] for a survey). Recently, this approach has been further extended with exotic algebras that lack distributivity but can still be used to find *locally optimal* paths [7, 8]. We refer to such algebras, with or without distributivity, as *path algebras*.

Constructing a new path algebra involves defining a carrier set, several operations, and proving that many algebraic properties hold. With complex constructions such proofs can be quite challenging. In one example that we look at closely in Section 3, J. Monnot and O. Spanjaar [6] define a *bottleneck semiring* for solving certain combinatorial problems. The proofs that show the constructed algebra is in fact a semiring are non-trivial.

This paper describes a domain-specific language for the specification of path algebras. The goal is automating the implementation and verification of specified algebras. The language is designed so that the specification writer simply supplies a high-level expression comprised of names of *built-in* algebras and algebraic *constructors* (direct product, lexicographic product, etc.). The specified algebra's implementation and proofs (or refutations) for all covered properties are then derived in a bottom-up syntax-directed manner. Details of our approach are described in Section 2.

We have implemented[1] our language using the Coq theorem prover [1]. However, in our approach all interactive theorem proving is performed at language *design time*. Specification writers use a tool implemented in OCaml code that has been automatically extracted [5] from our Coq implementation. That is, specification writers are not required to do any interactive theorem proving.

Our initial motivation came from the development of Internet routing protocols, where the intended specification writers are network operators or protocols designers who are not well versed in the art of proving theorems [4]. However, we feel that this approach may have applications in other areas such as Operations Research. In particular, our implementation allows algebra designers to quickly explore a large space of specifications. By way of illustration, in

[1] The implementation is available at `http://www.cl.cam.ac.uk/~tgg22/metarouting`.

Section 3 we specify the bottleneck semiring mentioned above, and show how required properties for a semiring can be automatically derived.

This is very much a work-in-progress, and we discuss some of our ongoing efforts in Section 4.

2 Language Engineering

Our framework consists of a collection $\mathcal{L}$ of constructions of algebras (such as semirings), and a fixed finite set of properties $\mathbb{P}$ for these algebraic structures. The goal is for every algebra defined by composing constructions in $\mathcal{L}$ to decide for every property in $\mathbb{P}$ if it is true or false. We call such $\mathcal{L}$ to be *closed* w.r.t. $\mathbb{P}$. To achieve this, for every construction `c` in $\mathcal{L}$ and every $p \in \mathbb{P}$ we aim to have a rule of the following shape[2]

$$p(\texttt{c}(a_1, \ldots, a_n)) \Leftrightarrow \beta_{c,p}(a_1, \ldots, a_n) \tag{1}$$

where $\beta_{c,p}$ stands for some boolean expression over properties in $\mathbb{P}$ of algebras $a_1, \ldots, a_n$. Let us call such rules — *iff-rules*. Notice that, if c takes no arguments, the right hand side is either true or false. Now given an algebra defined by constructions, we can use iff-rules to infer properties in bottom-up way.

Insisting on iff-rules may require adding new properties to $\mathbb{P}$. Consider the selectivity property $(\forall xy.\ x \oplus y = x \vee x \oplus y = y)$ for the direct product of semigroups $(A, \oplus_A)$ and $(B, \oplus_B)$. Instantiating the selectivity property with the direct product construction gives us

$$\forall x_1 y_1 \in A.\ \forall x_2 y_2 \in B.\ (x_1 \oplus_A y_1 = x_1 \wedge x_2 \oplus_B y_2 = x_2) \vee (x_1 \oplus_A y_1 = y_1 \wedge x_2 \oplus_B y_2 = y_2) \tag{2}$$

To get the iff-rule, we need to simplify (2), so that it becomes a boolean expression of properties of only A or only B. Consequently, we get

$$\begin{aligned} &((\forall x_1 y_1 \in A.\ x_1 \oplus_A y_1 = x_1) \wedge (\forall x_2 y_2 \in B.\ x_2 \oplus_B y_2 = x_2)) \\ \vee\ &((\forall x_1 y_1 \in A.\ x_1 \oplus_A y_1 = y_1) \wedge (\forall x_2 y_2 \in B.\ x_2 \oplus_B y_2 = y_2)) \\ \vee\ &((\forall x_1 y_1 \in A.\ x_1 \oplus_A y_1 = x_1 \vee x_1 \oplus_A y_1 = y_1) \wedge (\forall x_2 y_2 \in B.\ x_2 = y_2)) \\ \vee\ &((\forall x_2 y_2 \in B.\ x_2 \oplus_B y_2 = x_2 \vee x_2 \oplus_B y_2 = y_2) \wedge (\forall x_1 y_1 \in A.\ x_1 = y_1)) \end{aligned} \tag{3}$$

If we do not have properties $\forall xy.\ x = y$, $\forall xy.\ x \oplus y = x$, and $\forall xy.\ x \oplus y = y$ in $\mathbb{P}$, we need to add them to get the iff-rule. As system develops we need to add more and more auxiliary properties. Which properties the final system will contain becomes an empirical observation, which is hard to predict beforehand. Tables in Appendix B list all iff-rules we currently know. The formula in (3) corresponds to the iff-rule in Table 3:

$$\begin{aligned} \mathrm{SL}(\texttt{sProduct}(S, S')) \Leftrightarrow\ &(\mathrm{L}(S) \wedge \mathrm{L}(S')) \vee (\mathrm{R}(S) \wedge \mathrm{R}(S')) \vee \\ &(\mathrm{SL}(S) \wedge \mathrm{SG}(S')) \vee (\mathrm{SL}(S') \wedge \mathrm{SG}(S)) \end{aligned}$$

We consider five different signatures shown in Fig. 1. All signatures have non-empty carrier. Additionally, they have axioms that $\oplus$ and $\otimes$ are associative binary operators, and $\leq$ is a preorder (reflexive and transitive) relation. Fig. 3 gives definitions of constructions for sets, semigroups and preorders. Some constructions take arguments of signatures from Fig. 1 together with additional axioms (we call them *preconditions*), e.g. a commutative and idempotent semigroup.

[2] We use `this` font for constructions of algebras.

ID	Name	Signature	Axioms
D	Sets	(S)	NE
S	Semigroups	$(S, \oplus)$	NE, ASSOC
P	Preorders	$(S, \leq)$	NE, REFL, TRANS
O	Order semigroups	$(S, \leq, \oplus)$	NE, REFL, TRANS, ASSOC
B	Bisemigroups	$(S, \oplus, \otimes)$	NE, $\text{ASSOC}_{\oplus}$, $\text{ASSOC}_{\otimes}$

Figure 1: A table of signatures with axioms. Notice that we have forgetful projections between signatures. Bisemigroups have two projection to semigroups as we can drop either $\oplus$ or $\otimes$.

ID	Name	Formula
NE	Non-empty	$\exists x.True$
ASSOC	Associative	$\forall xyz.x \oplus (y \oplus z) = (x \oplus y) \oplus z$
REFL	Reflexive	$\forall x.x \leq x$
TRANS	Transitive	$\forall xyz.x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$

Figure 2: Definitions of axioms used in signatures.

Note that a construction not only has to define the appropriate operations and relations, but also to make sure that these operations satisfy required axioms.

To define bisemigroups and order semigroups it is enough to define their projections (Fig. 4). The last two constructions in Fig. 4 add special elements. `bAddOne`(B) and `bAddZero`(B) are equivalent to bisemigroups $(B \uplus \{\overline{1}\}, \oplus_B, \otimes_B)$ and $(B \uplus \{\overline{0}\}, \oplus_B, \otimes_B)$ respectively, where $\overline{1}$ is annihilator for $\oplus_B$ and identity for $\otimes_B$, and $\overline{0}$ is identity for $\oplus_B$ and annihilator for $\otimes_B$.

2.1 Witnesses

Another goal we have is to give witnesses to properties in $\mathbb{P}$ with existential quantifiers. We can split iff-rule (1) into two implication.

$$p(\mathtt{c}(a_1, \ldots, a_n)) \Leftarrow \beta_{c,p}(a_1, \ldots, a_n) \tag{4}$$
$$\neg p(\mathtt{c}(a_1, \ldots, a_n)) \Leftarrow \neg \beta_{c,p}(a_1, \ldots, a_n) \tag{5}$$

By $\neg$ we mean that the negation in front of p and $\beta_{c,p}$ is pushed through quantifiers to relation symbols. One of the implication is responsible for proving a property with existential quantifier. Say it is (5). If it is proved constructively, the proof says how to construct a witness for existential quantifier in $\neg p$ from witnesses of existential quantifiers in $\neg\beta_{c,p}$. Consequently, we can construct witnesses in bottom-up way as well as infer properties.

Some iff-rules like $\text{LD}(\mathtt{bAddOne}(B)) \Leftrightarrow \text{LD}(B) \wedge \text{IDM}(B_{\oplus}) \wedge (\text{RI}(B) \vee \neg\text{IDM}(B_{\oplus}))$ may look unnecessarily complex, as classically it can be simplified to $\text{LD}(\mathtt{bAddOne}(B)) \Leftrightarrow \text{LD}(B) \wedge \text{RI}(B)$. Consider the negative form (5) of this rule

$$\neg\text{LD}(\mathtt{bAddOne}(B)) \Leftarrow \neg\text{LD}(B) \vee \neg\text{IDM}(B_{\oplus}) \vee (\neg\text{RI}(B) \wedge \neg\text{IDM}(B_{\oplus}))$$

Remember that the negation is pushed inside, e.g. $\neg$LD is $\exists xyz.z \otimes (x \oplus y) \neq (z \otimes x) \oplus (z \otimes y)$. To prove the implication, we construct three different counterexamples corresponding to three cases separated by disjunction. The rule explicitly says that we need to make a case split if

dUnit is a singleton set $\mathbb{I}$.
dNat is a set $\mathbb{N}$ of natural numbers.
dProduct takes two sets A and B and constructs their direct product $A \times B$.
dUnion takes two sets A and B and constructs their disjoint union $A \uplus B$.
dFSets takes a sets A and constructs a set of all finite subsets of A, denoted by $\mathcal{P}(A)$.
dFMinSets takes a preorder $(A, \leq)$ and constructs a set of all minimal finite subsets of A, denoted by $\mathcal{P}_{\leq}(A)$. A subset X is minimal if $X = \min_{\leq}(X)$ where $\min_{\leq}(X) = \{x \in X \mid \forall y \in X. y \not< x\}$.
sUnit is a semigroup $(\mathbb{I}, K)$ where K is the constant binary operation.
sNatPlus is a semigroup $(\mathbb{N}, +)$.
sNatMin is a semigroup $(\mathbb{N}, \min)$.
sNatMax is a semigroup $(\mathbb{N}, \max)$.
sProduct takes two semigroups $(A, \oplus)$, $(B, \oplus')$ and constructs a semigroup $(A \times B, \oplus_{\times})$ where $(a_1, b_1) \oplus_{\times} (a_2, b_2) = (a_1 \oplus a_2, b_1 \oplus' b_2)$.
sLeftSum takes two semigroups $(A, \oplus)$, $(B, \oplus')$ and constructs a semigroup $(A \uplus B, \oplus_{A \uplus B})$ where

$$x \oplus_{A \uplus B} y = \begin{cases} x \oplus y & \text{if } x, y \in A \\ x & \text{if } x \in A, y \in B \\ y & \text{if } x \in B, y \in A \\ x \oplus' y & \text{if } x, y \in B \end{cases}$$

sRightSum takes two semigroups $(A, \oplus)$, $(B, \oplus')$ and constructs a semigroup $(B \uplus A, \oplus_{A \uplus B})$.
sLex takes two semigroups $(A, \oplus)$ and $(B, \oplus')$, s.t. $\oplus$ is commutative and idempotent, and $\oplus'$ has an identity elements $\overline{0}_B$. The resulting semigroup is $(A \times B, \vec{\oplus})$ where $(x_1, x_2) \vec{\oplus} (y_1, y_2) =$

$$\begin{cases} (x_1, x_2 \oplus' y_2) & \text{if } x_1 = y_1 \\ (x_1, x_2) & \text{if } x_1 = (x_1 \oplus y_1) \neq y_1 \\ (y_1, y_2) & \text{if } x_1 \neq (x_1 \oplus y_1) = y_1 \\ (x_1 \oplus y_1, \overline{0}_B) & \text{if } x_1 \neq (x_1 \oplus y_1) \neq y_1 \end{cases}$$

$\vec{\oplus}$ first chooses according to $\oplus$, only if $x_1 = y_1$ it chooses according to $\oplus'$. The fourth case is used when x_1 and y_1 are incomparable.
sSelLex is similar to sLex. It does not require $\oplus'$ to have the identity, but instead the $\oplus$ has to be selective. Consequently, the fourth case in $\vec{\oplus}$ can never happen.
sFSetsUnion takes a set A and constructs a semigroup $(\mathcal{P}(A), \cup)$.
sFSetsOp takes a semigroup $(A, \oplus)$ and constructs a semigroup $(\mathcal{P}(A), \hat{\oplus})$ where

$$X \hat{\oplus} Y = \{x \oplus y \mid x \in X, y \in Y\}$$

sFMinSetsUnion takes a preorder $(A, \leq)$, s.t. $\leq$ is antisymmetric, and constructs a semigroup $(\mathcal{P}_{\leq}(A), \cup_{\leq})$ where $\cup_{\leq} = \min_{\leq} \circ \cup$.
sFMinSetsOp takes an order semigroup $(A, \leq, \oplus)$, s.t. $\leq$ is antisymmetric and $\oplus$ is monotone, and constructs a semigroup $(\mathcal{P}_{\leq}(A), \hat{\oplus}_{\leq})$ where $\hat{\oplus}_{\leq} = \min_{\leq} \circ \hat{\oplus}$.
pLeftNaturalOrder takes a semigroup $(A, \oplus)$, s.t. $\oplus$ is commutative and idempotent, and constructs a preorder $(A, \leq_L)$ where $x \leq_L y \Leftrightarrow x \oplus y = x$.
pRightNaturalOrder takes a semigroup $(A, \oplus)$, s.t. $\oplus$ is commutative and idempotent, and constructs a preorder $(A, \leq_R)$ where $x \leq_R y \Leftrightarrow x \oplus y = y$.
pDual takes a preorder $(A, \leq)$ and constructs a preorder $(A, \geq)$.

Figure 3: Constructions of sets, semigroups and preorders.

IDM($B_{\oplus}$) holds or not in order to construct the counterexample. Classically the case split is trivial, but constructively we cannot do it for arbitrary B.

Splitting iff-rules into (4) and (5) helps us in using the framework that is under development where we do not have iff-rules. If $\beta_{c,p}$ in (4) is different from $\beta_{c,p}$ in (5), we can still generate proofs and compute witnesses, but we may run into situations where we cannot decide if some property is true or false.

2.2 Key Properties

All properties in $\mathbb{P}$ are defined in Appendix A. In this section we explain reasons why some properties were initially added to $\mathbb{P}$. Let us denote the initial set of properties by $\mathbb{P}_0$.

One of the aims of the system is to build semirings. Bisemigroups only guarantee associa-

Constructor in $\mathcal{L}$	$\leq$ *projection*	$\oplus$ *projection*
Order semigroups		
`oLeftNaturalOrder`(S)	`pLeftNaturalOrder`(S)	S
`oRightNaturalOrder`(S)	`pRightNaturalOrder`(S)	S
Constructor in $\mathcal{L}$	$\oplus$ *projection*	$\otimes$ *projection*
Bisemigroups		
`bUnit`	`sUnit`	`sUnit`
`bNatMinPlus`	`sNatMin`	`sNatPlus`
`bNatMaxMin`	`sNatMax`	`sNatMin`
`bProduct`(B, B')	`sProduct`$(B_\oplus, B'_\oplus)$	`sProduct`$(B_\otimes, B'_\otimes)$
`bLex`(B, B')	`sLex`$(B_\oplus, B'_\oplus)$	`sProduct`$(B_\otimes, B'_\otimes)$
`bSelLex`(B, B')	`sSelLex`$(B_\oplus, B'_\oplus)$	`sProduct`$(B_\otimes, B'_\otimes)$
`bFSetsOp`(S)	`sFSetsUnion`(S_{Set})	`sFSetsOp`(S)
`bFMinSets`(O)	`sFMinSetsUnion`$(O_\leq)$	`sFMinSetsOp`(O)
`bFMinSetsOpUnion`(O)	`sFMinSetsOp`(O)	`sFMinSetsUnion`$(O_\leq)$
`bAddOne`(B)	`sLeftSum`$(\texttt{sUnit}, B_\oplus)$	`sRightSum`$(B_\otimes, \texttt{sUnit})$
`bAddZero`(B)	`sLeftSum`$(B_\oplus, \texttt{sUnit})$	`sRightSum`$(\texttt{sUnit}, B_\otimes)$

Figure 4: Constructions for order semigroups and bisemigroups. S ranges over semigroups, B, B' — over bisemigroups, O — over order semigroups. We use indexes $(Set, \oplus, \otimes, \leq)$ to denote projected algebras. In addition to axioms inherited from projections `bSelLex` requires $B'_\oplus$ to be commutative, and `bAddOne` requires $B_\oplus$ to be commutative.

tivity. Hence, we need other semiring axioms to know if a bisemigroup is actually a semiring.

Studies of BGP [3] protocol show that path algebras are interesting even when operations $\oplus$ and $\otimes$ do not form a semiring. In such case we are not looking for optimal paths, but for locally optimal paths, where each node has the best path depending on what their neighbours have chosen. An interesting property in this scenario is the increasing property $(\forall xy.x \oplus (y \otimes x) = x)$, which as shown by T.G. Griffin and J.L. Sobrinho [9] is needed for Dijkstra's algorithm to find locally optimal solutions.

Another key property is: the identity for $\oplus$ is also the annihilator for $\otimes$. It implies 0-stability property $(\forall x.\overline{1} \oplus x = \overline{1})$ used by M. Gondran and M. Minoux [2]. 0-stability guarantees the convergence of matrix multiplication shortest path algorithm in n steps, where n is the number of nodes in the graph.

Also properties like idempotency, selectivity or antisymmetry are in $\mathbb{P}_0$, because they are required for some constructions as preconditions.

2.3 Iff-Rules in Context

Some properties defined in Appendix A are only meaningful in the context where some other properties (say $p_1, \ldots, p_n$) are satisfied. We denote this by $p(p_1, \ldots, p_n)$. If a property is defined in a context or a construction has preconditions, the proofs of the iff-rule take the context and preconditions as assumption. For example, to proof the iff-rule for minimal set construction of bisemigroups and right increasing property we need to show

$$\begin{aligned}\text{CM}_\oplus(\texttt{bFMinSets}(O)) \Rightarrow \text{IDM}_\oplus(\texttt{bFMinSets}(O)) \Rightarrow \\ \text{LM}(O) \Rightarrow \text{RM}(O) \Rightarrow \text{ASM}_\leq(O) \Rightarrow (\text{RI}(\texttt{bFMinSets}(O)) \Leftrightarrow \text{LND}(O))\end{aligned}$$

2.4 Putting it all together

We defined various constructions of algebras and showed how iff-rules can be used to prove properties. To make an automatic tool out of iff-rules we need to reflect constructions in the collection $\mathcal{L}$ into a mutually inductive language (with a syntactic type for each signature). For each construction we have a corresponding syntactic constructor. To map back from terms in syntax to algebras, we have a semantics function that is mutually inductively defined on the syntax structure, e.g. for bisemigroups the semantics function has type

$$\text{BS} \to (S, \oplus, \otimes, \vec{\pi}, \vec{\rho}) + \text{error}$$

where BS is the syntactic category for bisemigroup specifications, $(S, \oplus, \otimes)$ is a bisemigroup, $\vec{\pi}$ contains proofs that this is actual a bisemigroup, i.e. S is not empty and $\oplus, \otimes$ are associative, $\vec{\rho}$ for each property in $\mathbb{P}$ contains a proof or a refutation. The semantics function fails and returns an error if some preconditions of constructors specified in the input do not hold.

We have implemented all definitions of constructions and proved the iff-rules in Coq. We also defined syntax and the semantic function in Coq. Using code extraction mechanism [5], we generate an OCaml implementation of the semantics function that can be invoked without invoking the Coq theorem prover. This gives us a tool for quickly defining algebras and getting their properties together with witnesses. Also since the semantics function provides implementations for the $\oplus$ and $\otimes$ operations, we can construct a concrete labelled graph and run a generalised shortest path algorithm.

3 Examples

3.1 Lexicographic Product

To show how the language can be used, let us consider a graph where each edge has a distance and a bandwidth. Say we want to find best paths according to these two metrics. Bisemigroups `bNatMinPlus` and `bNatMaxMin` can be used to represent distance and bandwidth respectively. We can make one metric more significant by ordering pairs of metrics lexicographicly. Here are the specifications of algebras in our syntax:

$$\texttt{bAddOne}(\texttt{bAddZero}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin}))) \tag{6}$$

$$\texttt{bAddOne}(\texttt{bAddZero}(\texttt{bSelLex}(\texttt{bNatMaxMin}, \texttt{bNatMinPlus}))) \tag{7}$$

However, the choice of which metric is more significant gives us algebras with significantly different properties. The bisemigroup specified by (6) is a semiring, but the one specified by (7) is not distributive. We illustrate how we can check these properties using iff-rules in Appendix B by proving left distributivity of the first bisemigroup.

$$\begin{aligned}
& \text{LD}(\texttt{bAddOne}(\texttt{bAddZero}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin})))) \\
\Leftrightarrow\ & \text{LD}(\texttt{bAddZero}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin}))) \wedge \\
& \text{IDM}_{\oplus}(\texttt{bAddZero}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin}))) \wedge \\
& (\text{RI}(\texttt{bAddZero}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin}))) \vee \\
& \neg\text{IDM}_{\oplus}(\texttt{bAddZero}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin})))) \\
\Leftrightarrow\ & \text{LD}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin})) \wedge \\
& \text{IDM}(\texttt{sLeftSum}(\texttt{sUnit}, \texttt{sSelLex}(\texttt{sNatMin}, \texttt{sNatMax}))) \wedge
\end{aligned}$$

$$
\begin{aligned}
& (\mathrm{RI}(\texttt{bSelLex}(\texttt{bNatMinPlus}, \texttt{bNatMaxMin})) \vee \\
& \neg\mathrm{IDM}(\texttt{sLeftSum}(\texttt{sUnit}, \texttt{sSelLex}(\texttt{sNatMin}, \texttt{sNatMax})))) \\
\Leftrightarrow\ & (\mathrm{LD}(\texttt{bNatMinPlus}) \wedge \mathrm{LD}(\texttt{bNatMaxMin}) \wedge (\mathrm{LC}(\texttt{sNatPlus}) \vee \mathrm{LCD}(\texttt{sNatMin}))) \wedge \\
& (\mathrm{IDM}(\texttt{sUnit}) \wedge \mathrm{IDM}(\texttt{sSelLex}(\texttt{sNatMin}, \texttt{sNatMax}))) \wedge \\
& ((\mathrm{RSI}(\texttt{bNatMinPlus}) \vee (\mathrm{RI}(\texttt{bNatMinPlus}) \wedge \mathrm{RI}(\texttt{bNatMaxMin}))) \vee \\
& \neg(\mathrm{IDM}(\texttt{sUnit}) \wedge \mathrm{IDM}(\texttt{sSelLex}(\texttt{sNatMin}, \texttt{sNatMax})))) \\
\Leftrightarrow\ & (\mathrm{LD}(\texttt{bNatMinPlus}) \wedge \mathrm{LD}(\texttt{bNatMaxMin}) \wedge (\mathrm{LC}(\texttt{sNatPlus}) \vee \mathrm{LCD}(\texttt{sNatMin}))) \wedge \\
& (\mathrm{IDM}(\texttt{sUnit}) \wedge \mathrm{IDM}(\texttt{sNatMax})) \wedge \\
& ((\mathrm{RSI}(\texttt{bNatMinPlus}) \vee (\mathrm{RI}(\texttt{bNatMinPlus}) \wedge \mathrm{RI}(\texttt{bNatMaxMin}))) \vee \\
& \neg(\mathrm{IDM}(\texttt{sUnit}) \wedge \mathrm{IDM}(\texttt{sNatMax}))) \\
\Leftrightarrow\ & (\mathit{True} \wedge \mathit{True} \wedge (\mathit{True} \vee \mathit{False})) \wedge (\mathit{True} \wedge \mathit{True}) \wedge \\
& ((\mathit{False} \vee (\mathit{True} \wedge \mathit{True})) \vee \neg(\mathit{True} \wedge \mathit{True})) \\
\Leftrightarrow\ & \mathit{True}
\end{aligned}
$$

Such derivations are done mechanically by our tool. In a similar way we can derive False for left distributivity of the second bisemigroup. The tool also generates a counterexample for left distributivity:

$$(0,1) \otimes_\times ((1,1) \vec{\oplus} (0,0)) = (0,2) \neq (0,1) = ((0,1) \otimes_\times (1,1)) \vec{\oplus} ((0,1) \otimes_\times (0,0))$$

3.2 The Bottleneck Semiring

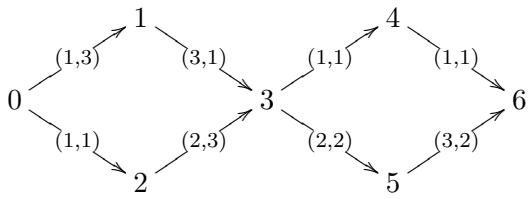
As a second example, consider the semiring defined by J. Monnot and O. Spanjaar [6] for finding best paths according to their bottleneck. Say we have a graph where edges are labelled by two independent metrics (two natural numbers). Values assigned to edges are partially ordered by comparing metrics pointwise. The weight of a path consists of a set of worst edges in the path. A path x is more preferred to another path y if for each edge in x there is a worse edge in y. We aim to have a semiring that calculates a set of such best paths between every pair of nodes in the graph.

In [6] the semiring is explicitly defined together with non-trivial proofs of semiring axioms. By reverse engineering definitions of the semiring operations, we can specify it in our language in the following way. We can represent pairs of metrics by `E = sProduct(sNatMin, sNatMin)`. Weights of paths are represented by `P = sFMinSetsUnion(pRightNaturalOrder(E))`. Finally, the bisemigroup that can be used to compute sets of best paths is

$$\texttt{B} = \texttt{bFMinSets}(\texttt{oRightNaturalOrder}(\texttt{P})) \tag{8}$$

Most importantly, in order to come up with the specification, we do not need to look to the proofs given in [6] — these can be generated automatically using the iff-rules as in the previous example.

If we take acyclic graphs as in [6], we can compute shortest paths according to the semiring using Bellman-Ford algorithm (Fig. 5). However, the semiring is not selective and it cannot be used with Dijkstra's algorithm. From the tool we get witnesses ($\{\{(1,0)\}\}$ and $\{\{(0,1)\}\}$) explaining where selectivity fails.



$0-1$	$\{\{(1,3)\}\}$	$1-4$	$\{\{(3,1)\}\}$	$3-4$	$\{\{(1,1)\}\}$
$0-2$	$\{\{(1,1)\}\}$	$1-5$	$\{\{(3,1);(2,2)\}\}$	$3-5$	$\{\{(2,2)\}\}$
$0-3$	$\{\{(1,3),(3,1)\},\{(2,3)\}\}$	$1-6$	$\{\{(3,1)\}\}$	$3-6$	$\{\{(1,1)\}\}$
$0-4$	$\{\{(1,3),(3,1)\},\{(2,3)\}\}$	$2-3$	$\{\{(2,3)\}\}$	$4-6$	$\{\{(1,1)\}\}$
$0-5$	$\{\{(1,3),(3,1),(2,2)\},\{2,3\}\}$	$2-4$	$\{\{(2,3)\}\}$	$5-6$	$\{\{(3,2)\}\}$
$0-6$	$\{\{(1,3),(3,1)\},\{(2,3)\}\}$	$2-5$	$\{\{(2,3)\}\}$		
$1-3$	$\{\{(3,1)\}\}$	$2-6$	$\{\{(2,3)\}\}$		

Figure 5: At the top is the example graph taken from [6]. At the bottom are the results computed by the tool using the bisemigroup defined in (8). For each pair of nodes we have a set of best paths. Each path is valued by a set of worst edges on that path.

4 Discussion, Future Work

Our approach to language design requires many iff-rules and it would be hard to ensure correctness without using a formal theorem prover. We have chosen the Coq theorem prover as it seems to meet our requirements quite well. Dependent types allow defining signatures for algebras as dependent records. Constructive proofs in Coq fit our need to associate witnesses to proofs of existential quantifiers. We haven't yet used some recent Coq features, such as type classes [10], which may simplify some of the infrastructure for our proofs.

A large part of our current effort is directed at "closing" the iff-rules listed in Appendix B. There is of course a conflict between the goal of closure and the goal of increased expressive power, and so various trade-offs have to be assessed at language design time.

Not all natural path problems can be expressed using bisemigroups. Many Internet routing protocols are best modelled by attaching functions to arcs in a graph. We are currently extending our system to encompass such algebraic structures.

References

[1] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer-Verlag, 2004.

[2] M. Gondran and M. Minoux. *Graphs, Dioids and Semirings: New Models and Algorithms*. Springer, 2008.

[3] T. G. Griffin, F. B. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking (TON)*, 10(2):232–243, 2002.

[4] T. G. Griffin and J. L. Sobrinho. Metarouting. *SIGCOMM*, 35:1–12, August 2005.

[5] P. Letouzey. A new extraction for coq. *Types for proofs and programs*, pages 617–617, 2003.

[6] J. Monnot and O. Spanjaard. Bottleneck shortest paths on a partially ordered scale. *4OR: A Quarterly Journal of Operations Research*, 1(3):225–241, 2003.

[7] J. Sobrinho and T. G. Griffin. Routing in equilibrium. In *19th International Symposium on Mathematical Theory of Networks and Systems (MTNS 2010)*, 2010.

[8] J. L. Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Transactions on Networking*, 13(5):1160–1173, October 2005.

[9] J. L. Sobrinho and T. G. Griffin. Routing in equilibrium. *Mathematical Theory of Networks and System*, 2010.

[10] B. Spitters and E Van Der Weegen. Developing the algebraic hierarchy with type classes in coq. *ITP 2010. International Conference on Interactive Theorem Proving*, 2010.

A Property Set $\mathbb{P}$

We use shorthands: $(x\#y) \Leftrightarrow (x \not\leq y \wedge y \not\leq x)$ and $(x \lesseqgtr y) \Leftrightarrow \neg(x\#y)$. For bisemigroups $\leq$ denotes the left natural order.

$\mathbb{P}_0$	*ID(Context)*	*Description*	*Formula*
	DecSetoid		
	SG	Singleton	$\exists c.\forall x.x = c$
	TE	Has exactly two elements	$\exists ab.\forall x.a \neq b \wedge (x = a \vee x = b)$
	FT	Finite	$\exists l : list\ S.\forall x.x \in l$
	Semigroup		
*	HI	Has identity	$\exists i.\forall x.(i \oplus x = x) \wedge (x \oplus i = x)$
*	HA	Has annihilator	$\exists w.\forall x.(w \oplus x = w) \wedge (x \oplus w = w)$
*	SL	Selective	$\forall xy.(x \oplus y = x) \vee (x \oplus y = y)$
*	CM	Commutative	$\forall ab.a \oplus b = b \oplus a$
*	IDM	Idempotent	$\forall x.x \oplus x = x$
	L	Always returns the left argument	$\forall ab.a \oplus b = a$
	R	Always returns the right argument	$\forall ab.a \oplus b = b$
	LCD	Left condensed	$\forall abc.a \oplus b = a \oplus c$
	RCD	Right condensed	$\forall abc.b \oplus a = c \oplus a$
	LC	Left cancelative	$\forall xyz.z \oplus x = z \oplus y \Rightarrow x = y$
	RC	Right cancelative	$\forall xyz.x \oplus z = y \oplus z \Rightarrow x = y$
	AL	Anti-left	$\forall xy.x \oplus y \neq x$
	AR	Anti-right	$\forall xy.x \oplus y \neq y$
	TG(CM, IDM)		$\forall xyz.x \oplus y \oplus z = x \oplus z \vee x \oplus y \oplus z = y \oplus z$
	Preorder		
*	TT	Total	$\forall xy.x \leq y \vee y \leq x$
*	ASM	Antisymmetric	$\forall xy.x \leq y \wedge y \leq x \Rightarrow x = y$
	OrderSemigroup		
*	LM	Left monotonic	$\forall axy.x \leq y \Rightarrow (a \oplus x \leq a \oplus y)$
*	RM	Right monotonic	$\forall xya.x \leq y \Rightarrow x \oplus a \leq y \oplus a$
	LND	Left non-decreasing	$\forall xy.x \leq x \oplus y$
	RND	Right non-decreasing	$\forall xy.x \leq y \oplus x$
	SND(IDM, ASM)	Selective non-decreasing	$\forall xy.x \leq x \oplus y \vee y \leq x \oplus y$
	IAUS(LM, RM, ASM, SL)		$\forall xyz.x\#y \Rightarrow x \oplus y = y \Rightarrow y \oplus x = y \Rightarrow z\#y \Rightarrow x = z$
	IAF(LM, RM, ASM, SL)		$\forall xyz.x\#y \Rightarrow x \oplus y = y \Rightarrow y \oplus x = y \Rightarrow x \oplus z = z \Rightarrow z \oplus x = z \Rightarrow x \neq z \Rightarrow (y \oplus z = z \wedge z \oplus y = z) \vee z \leq y$
	RT	Right total	$\forall abc.(b \oplus a) \leq (c \oplus a) \vee (c \oplus a) \leq (b \oplus a)$
	LT	Left total	$\forall abc.(a \oplus b) \leq (a \oplus c) \vee (a \oplus c) \leq (a \oplus b)$
	RMCC(RT)		$\forall xyzw.(x \oplus z) < (y \oplus z) \Rightarrow (y \oplus w) < (x \oplus w) \Rightarrow (x \oplus z) \leq (y \oplus w) \vee (y \oplus w) \leq (x \oplus z)$
	LMCC(LT)		$\forall xyzw.(z \oplus x) < (z \oplus y) \Rightarrow (w \oplus y) < (w \oplus x) \Rightarrow (z \oplus x) \leq (w \oplus y) \vee (w \oplus y) \leq (z \oplus x)$
	Bisemigroup		
*	LD	Left distributive	$\forall abc.c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$
*	RD	Right distributive	$\forall abc.(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
*	PITA($HI_\oplus$, $HA_\otimes$)	Plus identity is times annihilator	$\alpha_\oplus = \omega_\otimes$
*	PATI($HA_\oplus$, $HI_\otimes$)	Plus annihilator is times identity	$\omega_\oplus = \alpha_\otimes$
*	RSI($CM_\oplus$, $IDM_\oplus$)	Right strict increasing	$\forall xy.x < x \otimes y$
*	LSI($CM_\oplus$, $IDM_\oplus$)	Left strict increasing	$\forall xy.x < y \otimes x$
	RI($CM_\oplus$, $IDM_\oplus$)	Right increasing	$\forall xy.x \oplus (x \otimes y) = x$
	LI($CM_\oplus$, $IDM_\oplus$)	Left increasing	$\forall xy.x \oplus (y \otimes x) = x$
	RSS($CM_\oplus$, $IDM_\oplus$)		$\forall abc.(a < b \Leftrightarrow a \otimes c < b \otimes c)$
	LSS($CM_\oplus$, $IDM_\oplus$)		$\forall abc.(a < b \Leftrightarrow c \otimes a < c \otimes b)$
	RCEC($CM_\oplus$, $IDM_\oplus$)		$\forall xyz.x \otimes z = y \otimes z \Rightarrow (x \lesseqgtr y)$
	LCEC($CM_\oplus$, $IDM_\oplus$)		$\forall xyz.z \otimes x = z \otimes y \Rightarrow (x \lesseqgtr y)$

$\mathbb{P}_0$	*ID(Context)*	*Description*	*Formula*
	RCC($CM_\oplus$, $IDM_\oplus$)		$\forall xyz.x \otimes z \# y \otimes z \Rightarrow (x \lessapprox y)$
	LCC($CM_\oplus$, $IDM_\oplus$)		$\forall xyz.z \otimes x \# z \otimes y \Rightarrow (x \lessapprox y)$
	LDT($CM_\oplus$, $IDM_\oplus$)	Left discrete	$\forall xyz.\neg(z \otimes x < z \otimes y)$
	RDT($CM_\oplus$, $IDM_\oplus$)	Right discrete	$\forall xyz.\neg(x \otimes z < y \otimes z)$
	LCP($CM_\oplus$, $IDM_\oplus$)	Left comparable	$\forall xyz.z \otimes x \lessapprox z \otimes y$
	RCP($CM_\oplus$, $IDM_\oplus$)	Right comparable	$\forall xyz.x \otimes z \lessapprox y \otimes z$
	RTID($HI_\oplus$)		$\forall xyz.(x \otimes z) \oplus (y \otimes z) = \alpha_\oplus \otimes z$
	LTID($HI_\oplus$)		$\forall xyz.(z \otimes x) \oplus (z \otimes y) = z \otimes \alpha_\oplus$
	PITLA($HI_\oplus$)	Plus identity is times left annihilator	$\forall x.\alpha_\oplus \otimes x = \alpha_\oplus$
	RITRA($HI_\oplus$)	Plus identity is times right annihilator	$\forall x.x \otimes \alpha_\oplus = \alpha_\oplus$

B Iff-Rules

	dProduct(D,D')	dUnion(D,D')	dFSets(D)	dFMinSets(P)
Prec.				
SG	SG(D) ∧ SG(D')	False	False	False
TE	(SG(D) ∧ TE(D')) ∨ (SG(D') ∧ TE(D))	SG(D) ∧ SG(D')	SG(D)	SG(P)
FT	FT(D) ∧ FT(D')	FT(D) ∧ FT(D')	FT(D)	FT(P)

Table 2: DecSetoid rules

	sProduct(S,S')	sLex(S,S')	sSelLex(S,S')	sLeftSum(S,S')
Prec.		CM(S), IDM(S), HI(S')	CM(S), SL(S)	
HI	HI(S) ∧ HI(S')	HI(S)	HI(S) ∧ HI(S')	HI(S')
HA	HA(S) ∧ HA(S')	HA(S) ∧ HA(S')	HA(S) ∧ HA(S')	HA(S)
SL	(L(S) ∧ L(S')) ∨ (R(S) ∧ R(S')) ∨ (SL(S) ∧ SG(S')) ∨ (SL(S') ∧ SG(S))	SL(S) ∧ SL(S')	SL(S')	SL(S) ∧ SL(S')
CM	CM(S) ∧ CM(S')	CM(S')	CM(S')	CM(S) ∧ CM(S')
IDM	IDM(S) ∧ IDM(S')	IDM(S')	IDM(S')	IDM(S) ∧ IDM(S')
L	L(S) ∧ L(S')	SG(S) ∧ L(S')	SG(S) ∧ L(S')	False
R	R(S) ∧ R(S')	SG(S) ∧ R(S')	SG(S) ∧ R(S')	False
LCD	LCD(S) ∧ LCD(S')	SG(S) ∧ SG(S')	SG(S) ∧ LCD(S')	False
RCD	RCD(S) ∧ RCD(S')	SG(S) ∧ SG(S')	SG(S) ∧ RCD(S')	False
LC	LC(S) ∧ LC(S')	SG(S) ∧ LC(S')	(SG(S) ∨ (¬SG(S') ∧ SG(S'))) ∧ LC(S')	LC(S) ∧ AL(S) ∧ SG(S')
RC	RC(S) ∧ RC(S')	SG(S) ∧ RC(S')	(SG(S) ∨ (¬SG(S') ∧ SG(S'))) ∧ RC(S')	RC(S) ∧ AR(S) ∧ SG(S')
AL	AL(S) ∨ AL(S')	SG(S) ∧ AL(S')	SG(S) ∧ AL(S')	False
AR	AR(S) ∨ AR(S')	SG(S) ∧ AR(S')	SG(S) ∧ AR(S')	False
TG	(TG(S) ∧ SG(S')) ∨ (TG(S') ∧ SG(S))	*work-in-progress*	*work-in-progress*	SL(S) ∧ TG(S')

Table 3: Semigroup rules

	sFSetsUnion(D)	sFSetsOp(S)	sFMinSetsUnion(P)	sFMinSetsOp(O)
Prec.			ASM(P)	ASM(O)
HI	True	HI(S)	True	*pos* : HI(O)
HA	FT(D)	True	*pos* : FT(P)	True
SL	SG(D)	L(S) ∨ R(S) ∨ (TE(S) ∧ IDM(S))	TT(P)	SL(O) ∧ (¬SL(O) ∨ (IAUS(O) ∧ IAF(O)))
CM	True	CM(S)	True	CM(O)
IDM	True	SL(S)	True	IDM(O) ∧ (¬IDM(O) ∨ SND(O))
L	False	False	False	False
R	False	False	False	False
LCD	False	False	False	False
RCD	False	False	False	False
LC	False	False	False	False
RC	False	False	False	False
AL	False	False	False	False
AR	False	False	False	False

	sFSetsUnion(D)	sFSetsOp(S)	sFMinSetsUnion(P)	sFMinSetsOp(O)
TG	SG(D)	*work-in-progress*	*work-in-progress*	*work-in-progress*

Table 4: Semigroup rules. Rules prefixed with *pos* : mean that we know only positive implication (4)

	pLeftNaturalOrder(S)	pRightNaturalOrder(S)	pDual(P)
Prec.	CM(S), IDM(S)	CM(S), IDM(S)	
TT	SL(S)	SL(S)	TT(P)
ASM	True	True	ASM(P)

Table 5: Preorder rules

	oLeftNaturalOrder(S)	oRightNaturalOrder(S)
Prec.	CM(S), IDM(S)	CM(S), IDM(S)
LM	True	True
RM	True	True
LND	SG(S)	True
RND	SG(S)	True
SND	SL(S)	True
IAUS	True	True
IAF	True	True
RT	TG(S)	TG(S)
LT	TG(S)	TG(S)
RMCC	True	SL(S)
LMCC	True	SL(S)

Table 6: OrderSemigroup rules

	bProduct(B,B')	bLex(B,B')	bSelLex(B,B')	bFSetsOp(S)
Prec.		$CM(B_{\oplus})$, $IDM(B_{\oplus})$, $HI(B'_{\oplus})$	$CM(B_{\oplus})$, $SL(B_{\oplus})$, $CM(B'_{\oplus})$	
LD	LD(B) ∧ LD(B')	LD(B) ∧ LD(B') ∧ (LSS(B) ∨ $LCD(B'_{\otimes})$) ∧ (LCEC(B) ∨ LTID(B')) ∧ (LCC(B) ∨ RITRA(B'))	LD(B) ∧ LD(B') ∧ ($LC(B_{\otimes})$ ∨ $LCD(B'_{\otimes})$)	True
RD	RD(B) ∧ RD(B')	RD(B) ∧ RD(B') ∧ (RSS(B) ∨ $RCD(B'_{\otimes})$) ∧ (RCEC(B) ∨ RTID(B')) ∧ (RCC(B) ∨ PITLA(B'))	RD(B) ∧ RD(B') ∧ ($RC(B_{\otimes})$ ∨ $RCD(B'_{\otimes})$)	True
PITA	PITA(B) ∧ PITA(B')	PITA(B) ∧ PITA(B')	PITA(B) ∧ PITA(B')	True
PATI	PATI(B) ∧ PATI(B')	PATI(B) ∧ PATI(B')	PATI(B) ∧ PATI(B')	SG(S)
RSS	RSS(B) ∧ RSS(B') ∧ (RDT(B) ∨ RCEC(B')) ∧ (RDT(B') ∨ RCEC(B))	RSS(B) ∧ RSS(B') ∧ (RCEC(B) ∨ RDT(B'))	RSS(B) ∧ RSS(B')	False
LSS	LSS(B) ∧ LSS(B') ∧ (LDT(B) ∨ LCEC(B')) ∧ (LDT(B') ∨ LCEC(B))	LSS(B) ∧ LSS(B') ∧ (LCEC(B) ∨ LDT(B'))	LSS(B) ∧ LSS(B')	False
RCEC	RCEC(B) ∧ RCEC(B') ∧ ($RC(B_{\otimes})$ ∨ $RC(B'_{\otimes})$)	RCEC(B) ∧ RCEC(B')	RCEC(B) ∧ RCEC(B')	SG(S)
LCEC	LCEC(B) ∧ LCEC(B') ∧ ($LC(B_{\otimes})$ ∨ $LC(B'_{\otimes})$)	LCEC(B) ∧ LCEC(B')	LCEC(B) ∧ LCEC(B')	SG(S)
RCC	*work-in-progress*	(RCEC(B) ∨ RCP(B')) ∧ RCC(B) ∧ RCC(B')	RCC(B) ∧ RCC(B')	RCD(S)
LCC	*work-in-progress*	(LCEC(B) ∨ LCP(B')) ∧ LCC(B) ∧ LCC(B')	LCC(B) ∧ LCC(B')	LCD(S)
LDT	LDT(B) ∧ LDT(B')	LDT(B) ∧ LDT(B')	LDT(B) ∧ LDT(B')	False
RDT	RDT(B) ∧ RDT(B')	RDT(B) ∧ RDT(B')	RDT(B) ∧ RDT(B')	False
LCP	LCP(B) ∧ LCP(B') ∧ (LDT(B) ∨ LDT(B'))	LCP(B) ∧ LCP(B')	LCP(B) ∧ LCP(B')	LCD(S)
RCP	RCP(B) ∧ RCP(B') ∧ (RDT(B) ∨ RDT(B'))	RCP(B) ∧ RCP(B')	RCP(B) ∧ RCP(B')	RCD(S)
RI	RI(B) ∧ RI(B')	RSI(B) ∨ (RI(B) ∧ RI(B'))	RSI(B) ∨ (RI(B) ∧ RI(B'))	L(S)
LI	LI(B) ∧ LI(B')	LSI(B) ∨ (LI(B) ∧ LI(B'))	LSI(B) ∨ (LI(B) ∧ LI(B'))	R(S)
RSI	(RI(B) ∧ RSI(B')) ∨ (RSI(B) ∧ RI(B'))	RSI(B) ∨ (RI(B) ∧ RSI(B'))	RSI(B) ∨ (RI(B) ∧ RSI(B'))	False

	bProduct(B,B')	bLex(B,B')	bSelLex(B,B')	bFSetsOp(S)
LSI	(LI(B) ∧ LSI(B')) ∨ (LSI(B) ∧ LI(B'))	LSI(B) ∨ (LI(B) ∧ LSI(B'))	LSI(B) ∨ (LI(B) ∧ LSI(B'))	False
RTID	RTID(B) ∧ RTID(B')	RTID(B) ∧ RTID(B') ∧ (RDT(B) ∨ RCD(B'$_{\otimes}$)) ∧ (RCP(B) ∨ PITLA(B'))	RTID(B) ∧ RTID(B') ∧ (RDT(B) ∨ RCD(B'$_{\otimes}$))	False
LTID	LTID(B) ∧ LTID(B')	LTID(B) ∧ LTID(B') ∧ (LDT(B) ∨ LCD(B'$_{\otimes}$)) ∧ (LCP(B) ∨ RITRA(B'))	LTID(B) ∧ LTID(B') ∧ (LDT(B) ∨ LCD(B'$_{\otimes}$))	False
PITLA	PITLA(B) ∧ PITLA(B')	PITLA(B) ∧ PITLA(B')	PITLA(B) ∧ PITLA(B')	True
RITRA	RITRA(B) ∧ RITRA(B')	RITRA(B) ∧ RITRA(B')	RITRA(B) ∧ RITRA(B')	True

Table 7: Bisemigroup rules

	bFMinSets(O)	bFMinSetsOpUnion(O)	bAddOne(B)	bAddZero(B)
Prec.	LM(O), RM(O), AMS(O)	LM(O), RM(O), AMS(O)	CM(B$_{\oplus}$)	
LD	True	IDM(O) ∧ LND(O) ∧ RND(O)	LD(B) ∧ IDM(B$_{\oplus}$) ∧ (RI(B) ∨ ¬IDM(B$_{\oplus}$))	LD(B)
RD	True	IDM(O) ∧ LND(O) ∧ RND(O)	RD(B) ∧ IDM(B$_{\oplus}$) ∧ (LI(B) ∨ ¬IDM(B$_{\oplus}$))	RD(B)
PITA	True	*work-in-progress*	PITA(B)	True
PATI	*work-in-progress*	True	True	PATI(B)
RSS	False	False	RSS(B) ∧ LSI(B)	False
LSS	False	False	LSS(B) ∧ RSI(B)	False
RCEC	TT(O)	SL(O) ∧ (¬SL(O) ∨ (IAUS(O) ∧ IAF(O)))	RCEC(B)	SL(B$_{\oplus}$)
LCEC	TT(O)	SL(O) ∧ (¬SL(O) ∨ (IAUS(O) ∧ IAF(O)))	LCEC(B)	SL(B$_{\oplus}$)
RCC	RT(O) ∧ (¬RT(O) ∨ RMCC(O))	SL(O) ∧ (¬SL(O) ∨ (IAUS(O) ∧ IAF(O)))	SL(B$_{\oplus}$)	RCC(B)
LCC	LT(O) ∧ (¬LT(O) ∨ LMCC(O))	SL(O) ∧ (¬SL(O) ∨ (IAUS(O) ∧ IAF(O)))	SL(B$_{\oplus}$)	LCC(B)
LDT	False	False	False	False
RDT	False	False	False	False
LCP	LT(O) ∧ (¬LT(O) ∨ LMCC(O))	SL(O) ∧ (¬SL(O) ∨ (IAUS(O) ∧ IAF(O)))	SL(B$_{\oplus}$)	LCP(B)
RCP	RT(O) ∧ (¬RT(O) ∨ RMCC(O))	SL(O) ∧ (¬SL(O) ∨ (IAUS(O) ∧ IAF(O)))	SL(B$_{\oplus}$)	RCP(B)
RI	LND(O)	RND(O)	RI(B)	RI(B)
LI	RND(O)	RND(O)	LI(B)	LI(B)
RSI	False	False	False	False
LSI	False	False	False	False
RTID	False	False	False	False
LTID	False	False	False	False
PITLA	True	*work-in-progress*	PITLA(B)	True
RITRA	True	*work-in-progress*	RITRA(B)	True

Table 8: Bisemigroup rules