Design of a Generic Linear Equation Solver and its Implementation

Hassan Aït-Kaci*

hak@acm.org

Abstract

This work exploits the convenience of object-orientation—as supported by, *e.g.*, C++ (*viz.*, multiple inheritance, template classes and functions, and operator overloading)—for designing a minimal set of generic classes implementing linear-equation solvers for a large variety of specific semi-ring structures. This illustrates using a simple relativistic paradigm to obtain, with a minimal set-up, a large collection of algorithms which can all be obtained as derived classes and instance objects of a single very abstract scheme. The resulting system is a truly generic solver which can single-handedly and efficiently solve left or right linear equational systems for optimization problems in number structures, but also in graphs and networks.

Keywords: Software Design Pattern; Object-Oriented Programming; Algebraic Structures; Fixed-Point Equations

*http://hassan-ait-kaci.net

Contents

1 Purpose of this work

This work means to illustrate how the we can exploit the convenience of object-orientation—as supported by, *e.g.*, C++ (*viz.*, multiple inheritance, template classes and functions, and operator overloading)—for designing a minimal set of generic classes implementing linear-equation solvers for a large variety of specific semi-ring structures. This will also illustrate a simple software development methodology based on a simple relativistic interpretation of object orientation that allows a minimal set-up to yield a large collection of algorithms which can all be obtained as derived classes and instance objects of a single very abstract scheme.[1]

Because algebraic structures were invented in mathematics for the precise same purpose and use as those of object-orientation in programming, it comes as no surprise that the two paradigms match quite harmoniously. The design specified in this paper is a proof of this in the domain of linear equation solving in a variety of algebraic structures.

This document is a specification of an Application Program Interface (API) in the form of a few generic classes for linear-equation solving in an abstract *semi-ring* structure. This specification is detailed below, along with all the mathematical background that is needed to understand it.

If implemented correctly, this API can solve a variety of linear equation-solving problems ranging from familar numerical equations, to regular expression equations, to graph path problems, including network flow optimization problems [2], Abstract Interpretation of programs [3, 1], and Program flow analysis [5].

2 One Equation and One Unknown

2.1 Inverses and quasi-inverses

The left linear *fixed-point* equation:

$$x = ax + b \tag{2.1}$$

is easily solved in a *ring* structure by:[2]

$$\begin{aligned} x &= ax + b \\ x - ax &= b \\ (1-a)x &= b \\ x &= (1-a)^{-1}b \end{aligned} \tag{2.2}$$

The right linear version of Equation (2.1) is:

$$x = xa + b \tag{2.3}$$

which is, too, solved by:

$$\begin{aligned} x &= xa + b \\ x - xa &= b \\ x(1-a) &= b \\ x &= b(1-a)^{-1} \end{aligned} \tag{2.4}$$

If the ring is a *commutative* ring—*i.e.*, $*$ is commutative as well—then both Equations (2.1), and (2.3) "collapse" into one, and so do solutions (2.2), and (2.4) "collapse" into:

$$x = \frac{b}{1-a}. \tag{2.5}$$

[1] See Section 6.

[2] Please refer to Section 5.2.3.

as the solution of Equation (2.1).

This is the most familar case, for most readers, of the *field of rationals* $\langle\mathbb{Q}, +, 0, *, 1\rangle$.[3]

Let us define x^*, the *quasi-inverse* of x, as the infinite sum:

$$x^* \stackrel{\text{def}}{=} \sum_{n\geq 0} x^n. \tag{2.6}$$

This sum is well known as the simplest of all Taylor series expansion:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots = \sum_{n=0}^{\infty} x^n = x^*. \tag{2.7}$$

It is then possible to rewrite the solution in (2.5) as either:

$$x = a^*b. \tag{2.8}$$

or:

$$x = ba^*. \tag{2.9}$$

Both can also be verified to be indeed *bona fide* solutions of Equations (2.1) and (2.3), respectively, by direct substitution:

$$\begin{aligned} a(a^*b) + b &= (aa^*)b + b \\ &= (a^* - 1)b + b \\ &= a^*b. \end{aligned}$$

and

$$\begin{aligned} (ba^*)a + b &= b(aa^*) + b \\ &= b(a^* - 1) + b \\ &= ba^*. \end{aligned}$$

The forms $x = a^*b$ and $x = ba^*$ of the solutions of Equations (2.1) and (2.3), are more general than the forms (2.2) and (2.4) in the sense that they involve only the additive operation $+$ and the multiplicative operation $\times$, whereas the forms (2.2) and (2.4) involve also both an additive and multiplicative *inverse* operations. This is a more general property because, after all, Equations (2.1) and (2.3) use only $+$ and $\times$, no inverses. Therefore, the forms (2.8) and (2.9) may be used to compute a solution to Equations (2.1) and (2.3) for different interpretations of $+$ and $\times$, when the sets wherein a, b, and x take their values do not possess sufficient algebraic structure for $+$ and $\times$ to provide all elements with inverses. The only requirement is that the quasi-inverse's *infinite* "Taylor" expansion (2.6) *converge* to a *limit*; *i.e.*, it must denote a finitely expressible element, or a finitely approximable element.

Indeed, for well-known structures with different interpretations of $+$ and $\times$, such as *multiplicative semi-lattice*[4] (also known as *path algebras* [2]),[5] these operations are also idempotent and therefore quasi-inverses exist. Then, using the solution's form (2.8) or (2.9) allows to solve systems of linear equations in a wider variety of algebraic structures, including *graphs*, *regular sets*, *distributive lattices*, as well as the familiar ring structures where the form (2.5) happens to be more easily expressible, as well as all the multi-dimensional variations of all these structures using matrix algebra.

[3] Strictly speaking, a *field* is not quite a commutative ring as required; *i.e.*, $\mathbb{Q}$ does not admit a multiplicative inverse for 0. Then, the solution described by Equation (2.5) exists in $\mathbb{Q}$ only under the condition that $a \neq 1$. In the case where $a = 1$, Equation (2.1) becomes *degenerate*. In $\mathbb{Q}$, a degenerate equation admits solutions iff $b = 0$, in which case any element of $\mathbb{Q}$ is a solution. In general semi-rings, existence of solutions for a degenerate equation will depend on the specific algebraic structure.

[4] Please refer to Section 5.1.5.

[5] Please refer to Section 5.2.2.

In *all* (!) these structures, a simple generic elimination algorithm such as, *e.g.*, the standard Gaussian elimination procedure, may be used to solve systems of linear fixed-point equations. Equation solving may be made more efficient in specific structures using the particular algebraic properties local to the specific structures. For example, the `Ring` class has both additive and multiplicative inverse methods; if it has as well exact precision, then an algorithm based on Equation (2.5), rather than on quasi-inverses, can be used.

2.2 Examples

$\langle \mathbb{Q}, +, 0, *, 1 \rangle$

This is the most familiar setting: the usual field of rational numbers arithmetic.[6]

This is how the structure $\langle \mathbb{Q}, +, 0, *, 1 \rangle$ is interpreted:

- it is an *field* on the set $\mathbb{Q}$ of rational numbers (*i.e.*, an Abelian ring without inverse for 0);
- the *additive* operation $+$ is the addition of rationals;
- the additive unit (or *zero*) is $0 \in \mathbb{Q}$;
- the additive inverse of a rational r is its *negative* $-r$;
- the *multiplicative* operation is the multiplication of rationals;
- the multiplicative unit (or *one*) is $1 \in \mathbb{Q}$;
- the multiplicative inverse of a rational r is its *reciprocal* $\frac{1}{r}$ (except for $r = 0$).

$\langle \mathbb{R}, +, 0, *, 1 \rangle$

$\langle \mathfrak{RE}_\Sigma, +, \emptyset, \cdot, \epsilon \rangle$

The set $\mathfrak{RE}_\Sigma$ is the set of all *regular sets* of finite strings of symbols of an alphabet Σ (*e.g.*, as denoted by *regular expressions* on Σ).

$\langle \{0, 1\}, \vee, 0, \wedge, 1 \rangle$

$\langle \mathbf{2}^S, \cap, \emptyset, \cup, S \rangle$

$\langle \mathbb{R}, \max, -\infty, \min, \infty \rangle$

3 Many Equations and Many Unknowns

A *system* of $m > 1$ left linear equations with $n > 1$ unknowns in fix-point form is shown in Figure 1. Luckily, this case can be reduced to the previous single equation and single unknown case.

3.1 Reduction to one equation and one unknown

There are two (equivalent) ways in which this reduction can be done. The first one is based on *Dynamic Programming*, and the second one is based on *Matrix Algebra*.

[6] Strictly speaking, the C++ types `float` and `double` are rational numbers because they use only a finite representation. The fact that *real* numbers can be approximated by finite rational number representations is the reason why such types are also used for computing with real numbers. The only important difference to keep in mind for the latter is that finite-representation types do rounding and/or truncating beyond the precision imposed by the finite representation. Such errors propagate and therefore, comparisons among `floats` and `doubles` must be done up to that precision. That is, rather than $x == y$, it is better to use $x - y < \varepsilon$, where ε is a small number (*e.g.*, $\varepsilon = 2^{-p}$, where p is any non-negative number of precision bits allowed by the representation). The lesser the *precision* p is, the slacker the approximation will be, but the faster will the convergence.

$$
\begin{array}{ccccccccccccc}
x_0 & = & a_{00}x_0 & + & \cdots & + & a_{0j}x_j & + & \cdots & + & a_{0(n-1)}x_{n-1} & + & b_0 \\
\vdots & & \vdots & & & \vdots & & \vdots & & \vdots & & \vdots & \\
x_i & = & a_{i0}x_0 & + & \cdots & + & a_{ij}x_j & + & \cdots & + & a_{i(n-1)}x_{n-1} & + & b_i \\
\vdots & & \vdots & & & \vdots & & \vdots & & \vdots & & \vdots & \\
x_{m-1} & = & a_{(m-1)0}x_0 & + & \cdots & + & a_{(m-1)j}x_j & + & \cdots & + & a_{(m-1)(n-1)}x_{n-1} & + & b_{m-1}
\end{array}
$$

Figure 1: System of m Left-Linear Fix-Point Equations With n Unknowns

Dynamic programming

The system of Figure 1 is expressed more concisely as:

$$S_0 = \left\{ x_i = \sum_{j=0}^{n-1} a_{ij}x_j + b_i \right\}_{i=0}^{m-1} \tag{3.1}$$

Expression (3.1) can be rewritten as:

$$S_0 = \{x_0 = \alpha_0 x_0 + \beta_0\} \cup S_1 \tag{3.2}$$

where:

$$\alpha_0 = a_{00}, \tag{3.3}$$

$$\beta_0 = b_0 + \sum_{j=1}^{n-1} a_{ij}x_j \tag{3.4}$$

and

$$S_1 = \left\{ x_i = \sum_{j=0}^{n-1} a_{ij}x_j + b_i \right\}_{i=1}^{m-1}. \tag{3.5}$$

Since β_0 is independent of x_0, the equation:

$$x_0 = \alpha_0 x_0 + \beta_0 \tag{3.6}$$

in Expression (3.2) is solved by:

$$x_0 = \alpha_0^* \beta_0. \tag{3.7}$$

Expression (3.7) gives x_0 only as a *parametric* solution in terms of the $n-1$ remaining *parametric* variables $x_1, \ldots, x_{n-1}$.

Substituting the value of x_0 given by Expression (3.7) in Expression (3.5), we get:

$$S_1 = \left\{ x_i = \sum_{j=1}^{n-1} a_{ij}^1 x_j + b_i^1 \right\}_{i=1}^{m-1} \tag{3.8}$$

where, for all $i = 1, \ldots, m-1$ and all $j = 1, \ldots, n-1$:

$$a_{ij}^1 = a_{ij} + a_{i0}a_{00}^* a_{0j}, \tag{3.9}$$

and for all $i = 1, \ldots, m-1$:

$$b_i^1 = b_i + a_{i0}a_{00}^*b_0. \tag{3.10}$$

Having proceeded thus, the new system obtained as Expression (3.8) is a system of $m-1$ equations and $n-1$ variables $(x_1, \ldots, x_{n-1})$. In other words, the system (3.8) contains one less variable (x_0 has been eliminated) and one less equation ($x_0 = \sum_{j=0}^{n-1} a_{ij}x_j$ has been eliminated).

Repeating this elimination process, it is straightforward to generalize the foregoing scheme by induction as follows. We start with the base case $(k = 0)$: for all $i = 0, \ldots, m-1$ and all $j = 0, \ldots, n-1$,

$$a_{ij}^0 = a_{ij} \tag{3.11}$$

and, for all $i = 0, \ldots, m-1$,

$$b_i^0 = b_i. \tag{3.12}$$

For all k, $k = 0, \ldots, m-1$, we have,

$$S_k = \left\{ x_i = \sum_{j=k}^{n-1} a_{ij}^k x_j + b_i^k \right\}_{i=k}^{m-1}. \tag{3.13}$$

Expression (3.13) can be rewritten as:

$$S_k = \{x_k = \alpha_k x_k + \beta_k\} \cup S_{k+1} \tag{3.14}$$

where, for $k = 0, \ldots, m-1$:

$$\alpha_k = a_{kk}^k, \tag{3.15}$$

$$\beta_k = b_k^k + \sum_{j=k+1}^{n-1} a_{ij}^k x_j. \tag{3.16}$$

such that, for all $i = k, \ldots, m-1$ and all $j = k+1, \ldots, n-1$:

$$a_{ij}^k = \begin{cases} a_{ij} & \text{if } k = 0, \\ a_{ij}^{k-1} + a_{i(k-1)}^{k-1}\alpha_{k-1}^* a_{(k-1)j}^{k-1} & \text{if } 0 < k < m; \end{cases} \tag{3.17}$$

and for all $i = 1, \ldots, m-1$:

$$b_i^k = \begin{cases} b_i & \text{if } k = 0, \\ b_i^{k-1} + a_{i(k-1)}^{k-1}\alpha_{k-1}^* b_{k-1}^{k-1} & \text{if } 0 < k < m. \end{cases} \tag{3.18}$$

Again, since β_k is independent of $x_0, \ldots, x_k$, the equation:

$$x_k = \alpha_k x_k + \beta_k \tag{3.19}$$

in Expression (3.14) is solved by:

$$x_k = \alpha_k^* \beta_k. \tag{3.20}$$

Thus, Expression (3.20) gives x_k as a *parametric* solution in terms of the $n-k-1$ remaining *parametric* variables $x_{k+1}, \ldots, x_{n-1}$.

Clearly, after *at most* m *steps*, this iterated parametric solving process halts. Indeed, substituting m for k in Expression (3.13), we obtain:

$$S_m = \quad \left\{ x_i = \textstyle\sum_{j=m}^{n-1} a_{ij}^m x_j + b_i^m \right\}_{i=m}^{m-1} \quad = \emptyset. \tag{3.21}$$

Therefore, the previous step's equational system S_{m-1} is independent of variables $x_0, \ldots, x_{m-1}$:

$$S_{m-1} = \{x_{m-1} = \alpha_{m-1} x_{m-1} + \beta_{m-1}\}. \tag{3.22}$$

where,

$$\alpha_{m-1} = a_{(m-1)(m-1)}^{m-1}, \tag{3.23}$$

$$\beta_{m-1} = b_{m-1}^{m-1} + \sum_{j=m}^{n-1} a_{ij}^{m-1} x_j. \tag{3.24}$$

Since β_{m-1} is independent of variables $x_0, \ldots, x_{m-1}$, the equation:

$$x_{m-1} = \alpha_{m-1} x_{m-1} + \beta_{m-1} \tag{3.25}$$

in Expression (3.22) is solved by:

$$x_{m-1} = \alpha_{m-1}^* \beta_{m-1}. \tag{3.26}$$

There are three situations to consider:

1. $m < n$: more variables than equations;
2. $m = n$: as many variables as equations;
3. $m > n$: more equations than variables.

This is what happens in each case:

1. $m < n$—*Underdefined system:* in this case, Equation (3.26) gives an expression of x_{m-1} in terms of the $n - m$ remaining variables $x_m, \ldots, x_{n-1}$. Therefore, since x_j, for $j = 0, \ldots, m-1$, depends on the $n - j - 1$ variables $x_{j+1}, \ldots, x_{n-1}$, all m variables $x_0, \ldots, x_{m-1}$ are expressed in terms of the $n - m$ remaining *parametric variables* $x_m, \ldots, x_{n-1}$.

2. $m = n$—*Well-defined system:* in this case, Equation (3.24) becomes $\beta_{m-1} = b_{m-1}^{m-1}$, and hence Equation (3.26) gives an expression of x_{m-1} independently of any variable. Therefore, since x_j, for $j = 0, \ldots, m-2$, depends on the $m - j - 1$ variables $x_{j+1}, \ldots, x_{m-1}$, all m variables $x_0, \ldots, x_{m-1}$ are expressed independently of any parametric variable. In this case the system is fully solved, and solutions are obtained by the propagation of values *from* x_{m-1} *back* to x_0.

3. $m > n$—*Overdefined system:* in this case, when we have a solution for $x_0, \ldots, x_{m-1}$ by back propagation of eliminated variables, there are still additional equations outstanding in the system. The only way the outstanding $m-n$ equations may be satisfied is if they are redundant with the m first equations; that is, if the solution $x_0, \ldots, x_{m-1}$ verifies the $m - n$ remaining equations.

If the structure R happens to be a *ring* $\langle D, +, \emptyset, \times, \mathbb{1}\rangle$, then the expressions solving the system in Figures 1 become, for all $k = 0, \ldots, m-1$, for all $i = k, \ldots, m-1$ and all $j = k+1, \ldots, n-1$:

$$a_{ij}^k = \begin{cases} a_{ij} \ \text{ if } k = 0, \\ \\ a_{ij}^{k-1} + a_{i(k-1)}^{k-1} \times \Big(\mathbb{1} + (-\alpha_{k-1})\Big)^{-1} \times a_{(k-1)j}^{k-1} \\ \quad \text{if } 0 < k < m; \end{cases} \tag{3.27}$$

and, for all $i = 1, \ldots, m-1$:

$$b_i^k = \begin{cases} b_i \ \text{ if } k = 0, \\ b_i^{k-1} + a_{i(k-1)}^{k-1} \times \Big(\mathbf{1} + (-\alpha_{k-1})\Big)^{-1} \times b_{k-1}^{k-1} \\ \quad \text{if } 0 < k < m. \end{cases} \tag{3.28}$$

The equation (3.19) is solved by:

$$x_k = \Big(\mathbf{1} + (-\alpha_k)\Big)^{-1} \times \beta_k \tag{3.29}$$

and the equation (3.25) is solved by:

$$x_{m-1} = \Big(\mathbf{1} + (-\alpha_{m-1})\Big)^{-1} \times \beta_{m-1}. \tag{3.30}$$

We leave expressions of the right version of the ring solutions as an exercise to the reader.

Matrix algebra

In the case where $m = n$, the systems of Figures 1 and 10 can be respectively rewritten, using matrix notation, as:[7]

$$X = AX + B \tag{3.31}$$

where $X \in D^{n1}$, $A \in D^{nn}$ and $B \in D^{n1}$, and:

$$X = XA + B \tag{3.32}$$

where $X \in D^{1n}$, $A \in D^{nn}$ and $B \in D^{1n}$.

Therefore, by Theorem 4,[8] it comes that the solutions of Equations (3.31) and (3.32) are, respectively:

$$X = A^* B \tag{3.33}$$

and:

$$X = BA^*. \tag{3.34}$$

3.2 Examples

$\langle \mathbb{Q}, +, 0, *, 1 \rangle$

This structure is a *commutative* ring. Therefore, the two systems in Figures 1 and 10 are identical, and the left and right solutions collapse into one solution. Namely, for all $k = 0, \ldots, m-1$, for all $i = k, \ldots, m-1$ and all $j = k+1, \ldots, n-1$:

$$a_{ij}^k = {a'}_{ij}^k = \begin{cases} a_{ij} & \text{if } k = 0, \\ a_{ij}^{k-1} + \dfrac{a_{i(k-1)}^{k-1} a_{(k-1)j}^{k-1}}{1 - \alpha_{k-1}} & \text{if } 0 < k < m; \end{cases} \tag{3.35}$$

[7] See Section 5.2.7.
[8] See Page 16.

and for all $i = 1, \ldots, m-1$:

$$b_i^k = {b'}_i^k = \begin{cases} b_i & \text{if } k = 0, \\ b_i^{k-1} + \dfrac{a_{i(k-1)}^{k-1} b_{k-1}^{k-1}}{1 - \alpha_{k-1}} & \text{if } 0 < k < m. \end{cases} \tag{3.36}$$

Finally, Equation (3.19) is solved in $\langle \mathbb{Q}, +, 0, *, 1 \rangle$ by:

$$x_k = \frac{\beta_k}{1 - \alpha_k} \tag{3.37}$$

and Equation (3.25) is solved by:

$$x_{m-1} = \frac{\beta_{m-1}}{1 - \alpha_{m-1}} \tag{3.38}$$

where, for $k = 0, \ldots, m-1$:

$$\alpha_k = \alpha'_k = a_{kk}^k, \tag{3.39}$$

$$\beta_k = \beta'_k = b_k^k + \sum_{j=k+1}^{n-1} a_{ij}^k x_j. \tag{3.40}$$

$\langle \mathbb{R}, +, 0, *, 1 \rangle$

$\langle \mathfrak{RE}_\Sigma, +, \emptyset, \cdot, \epsilon \rangle$

$\langle \{0, 1\}, \vee, 0, \wedge, 1 \rangle$

$\langle \mathbb{R}, \max, -\infty, \min, \infty \rangle$

4 Three Solving Schemes

4.1 Structures with inverses and exact precision

$\langle \mathbb{Q}, +, 0, *, 1 \rangle$

$\langle \mathbb{R}, \max, -\infty, \min, \infty \rangle$

4.2 Structures without inverses, but with stationary points

$\langle \mathfrak{RE}_\Sigma, +, \emptyset, \cdot, \epsilon \rangle$

$\langle \{0, 1\}, \vee, 0, \wedge, 1 \rangle$

4.3 Structures with inverses, but no exact precision

$\langle \mathbb{R}, +, 0, *, 1 \rangle$

5 A definitional ontology of algebraic structures

5.1 Monoidal structures

DEFINITION 1 (MONOIDAL STRUCTURE) *A monoidal structure $\langle D, \star \rangle$ consists of a set D of elements—the domain—with an internal binary operation:*

$$\star : D \times D \to D. \tag{5.1}$$

In a monoidal structure, the operation $\star$ has an associated *prefix relation* defined for all $x, y \in D$ as:

$$x \prec_\star y \;\text{ iff }\; \exists z \in D,\; x \star z = y. \tag{5.2}$$

5.1.1 Semigroup

DEFINITION 2 (SEMIGROUP) *A semigroup $\langle D, \star\rangle$ is a monoidal structure with domain D whose operation $\star$ (5.1) is associative. That is, for all $x, y, z \in D$:*

$$x \star (y \star z) = (x \star y) \star z. \tag{5.3}$$

Note that in a semigroup $\langle D, \star\rangle$, the prefix relation $\prec_\star$ is always transitive (by virtue of associativity of $\star$). However, but it is not necessarily reflexive.

5.1.2 Monoid

DEFINITION 3 (MONOID) *A monoid $\langle D, \star, \epsilon\rangle$ is a semigroup $\langle D, \star\rangle$ with a special element $\epsilon \in D$, called a unit, such that, for all $x \in D$:*

$$x \star \epsilon = \epsilon \star x = x. \tag{5.4}$$

Note that in a monoid $\langle D, \star, \epsilon\rangle$, the prefix relation $\prec_\star$ is also reflexive (by virtue of the unit element). Therefore, it is a preorder, and is sometimes called the monoid's *prefix approximation*.

5.1.3 Group

DEFINITION 4 (GROUP) *A group $\langle D, \star, \epsilon\rangle$ is a monoid such that any element x has an inverse. That is, for any $x \in D$, there exists a (necessarily unique) $x^{-1} \in D$ such that:*

$$x \star x^{-1} = x^{-1} \star x = \epsilon. \tag{5.5}$$

5.1.4 Abelian structure

DEFINITION 5 (ABELIAN STRUCTURE) *An Abelian structure is any of the foregoing monoidal structures whose operation $\star$ (5.1) is commutative. That is, for all $x, y \in D$:*

$$x \star y = y \star x. \tag{5.6}$$

Thus, we speak of an *Abelian* operation, an *Abelian* semigroup, an *Abelian* monoid, an *Abelian* group, *etc.*, ... Alternatively, the more suggestive adjective "commutative" is sometimes preferred to "Abelian."

5.1.5 Semilattice

DEFINITION 6 (SEMILATTICE) *A semilattice $\langle D, \star\rangle$ is a commutative semigroup such that $\star$ is idempotent; i.e., for all $x \in D$:*

$$x \star x = x. \tag{5.7}$$

A natural partner to the $\star$ operation is the relation defined as $\leq_\star$ on D by:

$$\forall x, y \in D,\; x \leq_\star y \;\text{ iff }\; x \star y = y. \tag{5.8}$$

The relation $\leq_\star$ is called the *semilattice ordering* and indeed defines a partial order on D. Namely, $\leq_\star$ is reflexive (by idempotence of $\star$), anti-symmetric (by commutativity of $\star$) and transitive (by associativity of $\star$).

In a semilattice $\langle D, \star\rangle$, the prefix relation $\prec_\star$ is also an ordering and furthermore it coincides with the semilattice ordering. Namely:

THEOREM 1 (ALGEBRAIC APPROXIMATION ORDERING) *$\forall x, y \in D,\ x \prec_\star y$ iff $x \leq_\star y$.*

PROOF Assume that $x \leq_\star y$. By definition, this means that $x \star y = y$. Thus, it is clear that $\exists z,\ x \star z = y$ (taking $z = y$). Therefore, $x \prec_\star y$.

Now assume that $x \prec_\star y$. Then, by definition, $x \star z_{xy} = y$ for some $z_{xy} \in D$. Hence,

$$\begin{aligned} x \star y &= x \star (x \star z_{xy}) && \text{(replacing } y \text{ by its value)} \\ &= (x \star x) \star z_{xy} && \text{(associativity)} \\ &= x \star z_{xy} && \text{(idempotence)} \\ &= y \end{aligned}$$

and so, $x \leq_\star y$. ■

Note that, $\star$ is automatically a *supremum* operation for its semilattice ordering; namely:

THEOREM 2 (ALGEBRAIC APPROXIMATION SUPREMUM) *For all $x, y, z \in D$:*

$$\text{if } y \leq_\star x \text{ and } z \leq_\star x \text{ then } y \star z \leq_\star x. \tag{5.9}$$

PROOF Assume that $y \leq_\star x$ and $z \leq_\star x$; then,

$$\begin{array}{lll} y \star x = x & \text{by (5.8)} & (a) \\ z \star x = x & \text{by (5.8)} & (b) \\ (y \star x) \star (z \star x) = x \star x & \text{by } (a) \text{ and } (b) & \\ (y \star x) \star (z \star x) = x & \text{by (5.7)} & \\ (y \star z) \star (x \star x) = x & \text{by (5.3) and (5.6)} & \\ (y \star z) \star x = x & \text{by (5.7)} & \\ y \star z \leq_\star x & \text{by (5.8).} & \end{array}$$

■

Finally, note that if a semilattice $\langle D, \star \rangle$ is also a monoid $\langle D, \star, \epsilon \rangle$, Equation (5.8) entails that ϵ is the (necessarily unique) *least element* of D for $\leq_\star$. Then, it is sometimes written as $\perp$ (and called *bottom*). Thus, a semilattice with bottom can also be described as an idempotent Abelian monoid.

5.2 Binoidal structures

DEFINITION 7 (BINOIDAL STRUCTURE) *A binoidal structure $\langle D, \star, * \rangle$ consists of two monoidal structures $\langle D, \star \rangle$ and $\langle D, * \rangle$*

In the notation used for an abstract structure, the particular symbols that denote the operation and unit element (if it is a monoid), are, of course, generic. Thus, in our definitions so far, we have used $\star$ for the operation, and ϵ for the unit element. Clearly, however, other symbols could be used instead—what matters is that the chosen symbols substituted for $\star$ and ϵ obey the appropriate equations. This being said, the familiar arithmetic operation symbols $+$ and $\times$, with associated unit symbols $\emptyset$ and $\imath$, respectively, are sometimes used as generic symbols, despite their conventional arithmetic meaning. Generally, this is to suggest that the structure at hand will behave as, or mostly as, in familiar arithmetic. The adjective *additive* (resp., *multiplicative*) is then used to designate properties of a structure whose operation is written $+$ (resp., $\times$).

Many common binoidal structures combine an additive and an multiplicative operation thanks to *distributivity*.

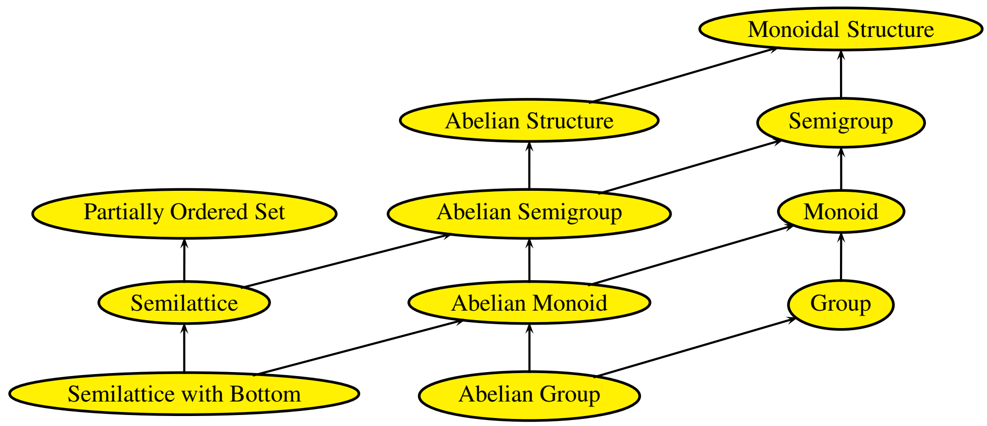

FIGURE 2: TAXONOMY OF MONOIDAL ALGEBRAIC STRUCTURES

5.2.1 Semiring

DEFINITION 8 (SEMIRING) *A semiring $\langle D, +, \emptyset, \times, \imath \rangle$ is a binoidal (additive and multiplicative) structure on a single set D such that:*

- *$\langle D, +, \emptyset \rangle$ is a commutative monoid;*
- *$\langle D, \times, \imath \rangle$ is a monoid;*
- *$\times$ is distributive over $+$; that is, for all $x, y, z \in D$:*

$$x \times (y + z) = (x \times y) + (x \times z) \tag{5.10}$$

and

$$(x + y) \times z = (x \times z) + (y \times z). \tag{5.11}$$

To distinguish between the two operations's unit elements in a semiring, the additive unit $\emptyset$ is referred to as the *zero* element, and the multiplicative unit $\imath$ as the *unit* element.

A semiring is an *Abelian* (or *commutative*) semiring if its multiplicative operation $\times$ is commutative (*i.e.*, if $\langle D, \times, \imath \rangle$ is a commutative monoid).

5.2.2 Path algebra

A *path algebra* is *the* appropriate algebraic structure to handle path problems in graphs and networks [2].

DEFINITION 9 (PATH ALGEBRA) *A path algebra $\langle D, +, \emptyset, \times, \imath \rangle$ is a semiring such that:*

- *$+$ is idempotent; i.e., for all $x \in D$:*

$$x + x = x; \tag{5.12}$$

- $\times$ *is idempotent; i.e., for all* $x \in D$*:*

$$x \times x = x; \tag{5.13}$$

- $\emptyset$ *is absorptive for* $\times$*; i.e., for all* $x \in D$*:*

$$x \times \emptyset = \emptyset \times x = \emptyset; \tag{5.14}$$

In other words, a path algebra is a semiring which is also an additive semilattice and a $\emptyset$-absorptive idempotent multiplicative semigroup.

5.2.3 Ring

DEFINITION 10 (RING) *A ring is a special case of a semiring. In fact, a ring structure is to a group what a semiring structure is to a monoid. Indeed, a ring* $\langle D, +, \emptyset, \times, \imath \rangle$ *is a binoidal (additive and multiplicative) structure on a set* D *such that:*

- $\langle D, +, \emptyset \rangle$ *is a commutative group;*
- $\langle D, \times, \imath \rangle$ *is a group;*
- *the multiplicative operation* $\times$ *is distributive over the additive operation* $+$*; that is, Equations (5.10) and (5.11) hold for all* $x, y, z \in D$*.*

A ring is an Abelian (or commutative) ring if its multiplicative operation $\times$ is commutative (*i.e.*, if $\langle D, \times, \imath \rangle$ is a commutative group).

5.2.4 Lattice

DEFINITION 11 (LATTICE) *A lattice* $\langle D, +, \times \rangle$ *is a binoidal structure such that:*

- $\langle D, + \rangle$ *is a semilattice (called its additive semilattice);*
- $\langle D, \times \rangle$ *is a semilattice (called its multiplicative semilattice);*
- *its two operations are mutually absorptive; i.e., for all* $x, y \in D$*:*

$$x + (x \times y) = x = x \times (x + y). \tag{5.15}$$

Thus, the structure of a lattice is symmetric with respect to its two operations in the sense that $\langle D, +, \times \rangle$ is a lattice iff $\langle D, \times, + \rangle$ is a lattice. This important property is called *duality*. It makes a statement equally valid when changing every additive part into its multiplicative counterpart, and vice versa.

Note that a lattice is partially ordered both as an additive semilattice and as a mutiplicative semilattice. In fact, it is easy to see that the two partial orders are mutual inverses. That is,

$$\leq_+ \;=\; \leq_\times^{-1}, \tag{5.16}$$

and thus also, by duality:

$$\leq_\times \;=\; \leq_+^{-1}. \tag{5.17}$$

By convention, because the additive and multiplicative orderings of a lattice are mutual inverses, we write simply $\leq$ for $\leq_+$ and $\geq$ for $\leq_\times$. Thus, if a lattice is an additive (resp., multiplicative) monoid, $\emptyset$ is the least (resp., $\mathtt{1}$ is the greatest) element for $\leq$ and is often referred to as "bottom" (resp., "top") and sometimes written "$\perp$" (resp., "$\top$").

Note also that if a lattice $\langle D, +, \times \rangle$ is an additive monoid $\langle D, +, \emptyset \rangle$, then $\emptyset$ is necessarily absorptive for $\times$; *i.e.*, Equation (5.14) holds for all $x \in D$. Dually, if a lattice $\langle D, +, \times \rangle$ is a multiplicative monoid $\langle D, \times, \mathtt{1} \rangle$, then $\mathtt{1}$ is necessarily absorptive for $+$; *i.e.*, Equation (5.18) holds for all $x \in D$:

$$x + \mathtt{1} = \mathtt{1} + x = \mathtt{1}. \tag{5.18}$$

It is important to realize that a lattice is neither an instance of, nor is it more general than, a semiring (it lacks distributivity). However, it is easy to show that the following "sub-distributive" inequality holds in a lattice:

THEOREM 3 (SUBDISTRIBUTIVITY) *Let $\mathcal{L} = \langle D, +, \times \rangle$ be a lattice. Then, for all x, y and z in D:*

$$x \times (y + z) \geq (x \times y) + (x \times z) \tag{5.19}$$

and, dually:

$$x + (y \times z) \leq (x + y) \times (x + z). \tag{5.20}$$

PROOF We need only establish Inequality (5.20); the proof of Inequality (5.19) is the same up to duality.

Let x, y, and z be arbitrary elements of a lattice $\langle D, +, \times \rangle$. Clearly, we have:

$$x \leq x + y \tag{5.21}$$

(since $x + (x + y) = x + y$). Similarly,

$$x \leq x + z. \tag{5.22}$$

Since $\times$ is an infimum operation for $\leq$, it follows from Inequalities (5.21) and (5.22) that:

$$x \leq (x + y) \times (x + z). \tag{5.23}$$

On the other hand, we also have:

$$y \times z \leq y \leq x + y \tag{5.24}$$

and

$$y \times z \leq z \leq x + z. \tag{5.25}$$

Again, because $\times$ is an infimum operation for $\leq$, Inequalities (5.24) and (5.25) imply that:

$$y \times z \leq (x + y) \times (x + z). \tag{5.26}$$

Finally, because $+$ is a supremum operation for $\leq$, Inequalities (5.23) and (5.26) imply Inequality (5.20).
■

A *distributive lattice* is a lattice in which equality, rather than $\leq$, holds in (5.19) for all x, y and z (or equivalently, by duality, if equality, rather than $\geq$, holds in (5.20)). Thus, a distributive lattice with top and bottom is both an additive and a multiplicative commutative semiring. That is, Equations (5.10) holds for all $x, y, z \in D$ (and so does (5.11), by commutativity of $\times$).[9]

Finally, note that a distributive lattice with top and bottom is simultaneously an additive and a multiplicative path algebra.

5.2.5 Boolean ring

DEFINITION 12 (BOOLEAN RING) *A boolean ring is a ring in which any element admits a (necessarily unique) complement with respect to the additive and multiplicative operations. That is, for any $x \in D$, there exists a unique $\bar{x} \in D$ such that:*

$$x + \bar{x} = \bar{x} + x = \mathbb{1}, \tag{5.29}$$

and

$$x \times \bar{x} = \bar{x} \times x = \emptyset. \tag{5.30}$$

5.2.6 Boolean lattice

DEFINITION 13 (BOOLEAN LATTICE) *A boolean lattice is a lattice which is also a boolean ring; i.e., it is a distributive complemented lattice.*

5.2.7 Matrix liftings

DEFINITION 14 (MATRIX LIFTINGS) *Given a semiring structure $\mathcal{R} = \langle D, +, \emptyset, \times, \mathbb{1} \rangle$ and two positive natural numbers m and n, we can construct its $m \times n$ matrix lifting:*

$$\mathfrak{M}^{mn}(\mathcal{R}) \;=\; \langle D^{mn}, +^{mn}, \emptyset^{mn}, \times^{mn}, \mathbb{1}^{mn} \rangle. \tag{5.31}$$

as shown as follows in Equations (5.32)–(5.36).

The domain of $m \times n$ matrices over R is defined as:

$$D^{mn} \;\stackrel{\text{def}}{=}\; \{\; d \in D^{m\times n} \mid d = \{d_{ij} \in D\}_{i=0,\;\; j=0}^{m-1, n-1} \;\} \tag{5.32}$$

Matrix addition is defined as follows. If $a \in D^{mn}$ and $b \in D^{mn}$ are two $m \times n$ matrices, then $\forall i, j,\; 0 \leq i \leq m-1, 0 \leq j \leq n-1$,

$$(a +^{mn} b)_{ij} \;\stackrel{\text{def}}{=}\; a_{ij} + b_{ij}; \tag{5.33}$$

The null $m \times n$ matrix is defined as follows: $\forall i, j,\; 0 \leq i \leq m-1, 0 \leq j \leq n-1$,

$$\emptyset^{mn}_{ij} \;\stackrel{\text{def}}{=}\; \emptyset \in D; \tag{5.34}$$

[9] This is equivalent, by duality, to the additive operation $+$ being also distributive over the multiplicative operation $\times$; that is:

$$x + (y \times z) = (x + y) \times (x + z) \tag{5.27}$$

or, by commutativity of $+$:

$$(x \times y) + z = (x + z) \times (y + z). \tag{5.28}$$

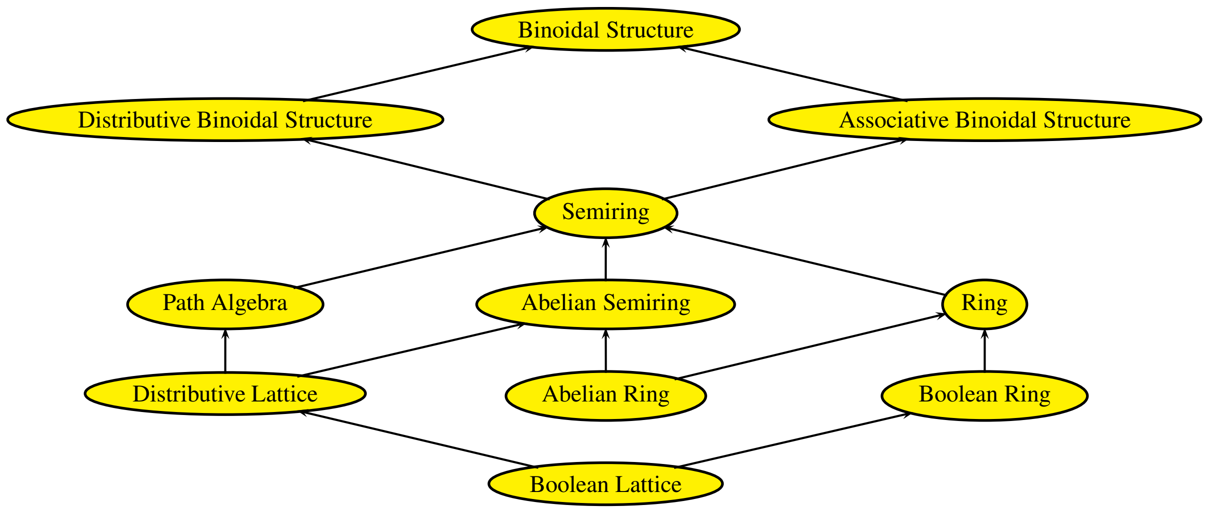

FIGURE 3: TAXONOMY OF BINOIDAL ALGEBRAIC STRUCTURES

Matrix multiplication is defined as follows. If a is an $m \times n$ matrix in D^{mn}, and b is an $n \times p$ matrix in D^{np}, then $\forall i, j,\ 0 \leq i \leq m-1, 0 \leq j \leq p-1$,

$$(a \times^{mp} b)_{ij} \stackrel{\text{def}}{=} \sum_{k=0}^{n-1} a_{ik} \times b_{kj}; \tag{5.35}$$

The unit $m \times n$ matrix is defined as follows: $\forall i, j,\ 0 \leq i \leq m-1, 0 \leq j \leq n-1$,

$$\mathtt{1}^{mn}_{ij} = \begin{cases} \mathtt{1} \in D & \text{if } i = j, \\ \emptyset \in D & \text{otherwise.} \end{cases} \tag{5.36}$$

We can simplify the $\_^{mn}$ notation in Equations (5.33)–(5.36) by dropping the dimension superscripts, with the dimension constraints implicit. Hence, Equations (5.33)–(5.36) become Equations (5.37)–(5.40):

$$(a + b)_{ij} \stackrel{\text{def}}{=} a_{ij} + b_{ij}; \tag{5.37}$$

$$\emptyset_{ij} \stackrel{\text{def}}{=} \emptyset \in D; \tag{5.38}$$

$$(a \times b)_{ij} \stackrel{\text{def}}{=} \sum_{k=0}^{n-1} a_{ik} \times b_{kj}; \tag{5.39}$$

$$\mathtt{1}_{ij} = \begin{cases} \mathtt{1} \in D & \text{if } i = j, \\ \emptyset \in D & \text{otherwise.} \end{cases} \tag{5.40}$$

An element $d = \{d_{ij} \in D\}_{i=0,\ j=0}^{m-1,n-1}$ of D^{mn} is written:

$$\begin{bmatrix} d_{00} & \cdots & d_{0j} & \cdots & d_{0(n-1)} \\ \vdots & \ddots & \vdots & & \vdots \\ d_{i0} & \cdots & d_{ij} & \cdots & d_{i(n-1)} \\ \vdots & & \vdots & \ddots & \vdots \\ d_{(m-1)0} & \cdots & d_{(m-1)j} & \cdots & d_{(m-1)(n-1)} \end{bmatrix} \tag{5.41}$$

Given a *matrix* $d \in D^{mn}$, an element d^{ij} of d is referred to as the *entry* of d at *row* i and *column* j. The ordered pair mn is called the *dimension* of the matrix: m is called the row dimension and n is called the column dimension. Note that the multiplicative matrix operation is *not* an internal function, but can only be applied if the first matrix' column dimension is equal to the second matrix' row dimension. However,

THEOREM 4 *Given a semiring R, its matrix lifting $\mathfrak{M}^{n^2}(\mathcal{R})$ for n fixed, is also a semiring.*

Note however that, even if R is a ring, $\mathfrak{M}^{n^2}(\mathcal{R})$ is still only a semiring.

6 A Relativistic View of Object Orientation

The essence of object-orientation coincides with that of Einstein's Special and General Relativity theories [4].

Einstein's Special Relativity Theory (SRT) is all based on the observation that there is a mathematical duality between being at rest on one hand, and being in motion on the other hand: all motion is relative to a set of reference. Hence, it is mathematically irrelevant whether I sit in a train moving along with it at some speed with respect to the scenery, or whether I sit in a motionless train while the scenery moves by in the opposite direction at the same speed.

Similarly, Einstein's General Relativity Theory (GRT) is all based on the observation that there is a mathematical duality between free falling frictionless in a straight line on one hand, and the texture of space being warped by massive bodies on the other hand: the curvature of all trajectory of motion is relative to space's own curvature. Hence, it is mathematically irrelevant whether the Earth is orbiting the Sun elliptically is a closed curve, or whether it free-falls frictionless indefinitely in a straight line, while space in which it moves is itself curved by the same opposite factor into the (hyper) elliptical (hyper) "eddy" created by the Sun's gravity.[10] Thus is GRT the key to explaining the mystery of "action at a distance" of gravity.

Similarly as well, object-orientation (OO) is based on the observation that there is a mathematical duality between an object being acted upon by a function on one hand, and a function being acted upon by an object on the other hand: the *orientation* of $f(x)$ is relative to the structure of interpretation of the object or the function. Hence, it is mathematically irrelevant whether the function f is applied to the object x, or whether the object x is *sent* the *message* f. In the first case (the conventional view), the function f *knows* what to do with an object of the type of x and performs it on x; in the second case (the *object-oriented view*), the object x *knows* what to do when it is asked to respond to the message sent to it as f, and performs it. Thus is OO the key to a new *decentralizing* view of computation which allows *distributed* computation and code modularity: whereas the conventional view's *centralizing* computation in functions made them huge, inefficient, and quickly impractical to maintain, the (mathematically equivalent) OO view now delegates computation to objects by making them react to messages sent to them by using methods specified for them by their class definitions.

Thus, object-orientation may simply be construed as exploiting a mathematical relativity principle. This relativistic view can be used as a systematic object-oriented software design methodology.

[10] "Hyper" because space is at least 3-dimensional...

```
class Object
  {
    virtual Object *method (Context *context);
  }
```

Figure 4: Object Class Skeleton

```
class Context
  {
    Object *method (Object *object) { return object.method(this); }
  }
```

Figure 5: Context Class Skeleton

To be precise, the change of perspective, when orienting computation with *reference* to an object rather than a function, is expressed mathematically by the set isomorphism:

$$A \to (B \to C) \simeq B \to (A \to C). \tag{6.1}$$

This equation essentially captures the dual *relativity* of computation alluded to above.

This article is an example of the general case that can be expressed as follows:

$$\begin{array}{c} \mathit{method} : \mathit{Context} \to (\mathit{Object} \to \mathit{Object}) \\ \simeq \\ \mathit{method} : \mathit{Object} \to (\mathit{Context} \to \mathit{Object}). \end{array} \tag{6.2}$$

Therefore, we can define two class structures, `Object` and `Context`, which *always* respectively declare a method (here called `method`) as shown in Figures 4 and 5.[11] Some examples are given in Figure 6.

Context	**Object**	**method**
`Name_Value_Environment`	`Expression`	`evaluate`
`Name_Type_Environment`	`Expression`	`typecheck`
`Run_Time_Environment`	`Instruction`	`execute`
`Algebraic_Structure`	`Equation`	`solve`
`Logical_Theory`	`Theorem`	`prove`
`Constraint_Structure`	`Constraint`	`resolve`
	⋮	

Figure 6: Some Use Cases for the Context/Object Relativity Principle

[11] Using C++ syntax:.

7 Implementation

A simple linear-equation solver over an algebraic dual structure (the parameter class `Structure`) should provide:

- a class to substitute for `Structure`, the type of elements in the structure's domain. This is the type of the coefficients `a`, and `b`, and that of the unknown `x` as well. This algebraic structure class will have:
  - a *private* member `rep` whose type is an adequate representation of the structure's domain elements.
  - a *public* `friend` method `operator+` that takes two arguments of type `const &Structure` and returns a result of type `&Structure`;[12]
  - a *public* `friend` method `operator-` that takes *one* argument of type `const &Structure` and returns a result of type `&Structure`;
  - a *public* `friend` method `operator-` that takes *two* arguments of type `const &Structure` and returns a result of type `&Structure`;
  - a *public* `const Structure zero;`
  - a *public* `friend` method `operator*` that takes two arguments of type `const &Structure` and returns a result of type `&Structure`;
  - a *public* `friend` method `operator/` that takes two arguments of type `const &Structure` and returns a result of type `&Structure`;
  - a *public* `const Structure one;`
  - a *public* `friend` method `operator==` that takes two arguments of type `const &Structure` and returns a result of type `bool`;
- a class `Equation` representing a linear fix-point equation on the `Structure`s; this class must have a *private* member `structure` of type `*Structure`;

We must also provide the *methods* `solve` for both `Structure` and `Equation<Structure>` classes following the design scheme of Section 6.

For example, solving over rational numbers should provide:

- a class `Rational` representing a rational number; this can be represented by pairs of integers, or decimal doubles, or whatever other equivalent representation of a rational number one may decide;[13]
  - a *private* member `rep` of, say, type `double`;
  - a *public* `friend` method `operator+` that takes two arguments of type `const &Rational` and returns a result of type `&Rational`;
  - a *public* `friend` method `operator-` that takes *one* argument of type `const &Rational` and returns a result of type `&Rational`;
  - a *public* `friend` method `operator-` that takes *two* arguments of type `const &Rational` and returns a result of type `&Rational`;

[12] The `&` return type may appear odd; however, keep in mind that the *generic* design eventually will actually allow the overloading of operators on *very big* structures such as, *e.g.*, matrices (*i.e.*, multidimensional arrays), and therefore saving the return copy space/time is worth saving. Be that as it may, we are free to choose to return a `Rational` instead of `&Rational` if we so wish. The `&` return type version has the advantage of generic uniformity for inheritance if doing the complete generic API.

[13] Recall that a rational number $r \in \mathbb{Q}$ is a pair of integers written $r = \frac{n}{d}$, where $n \in \mathbb{N}$ is the *numerator* and $d \in \mathbb{N}$ is the *denominator*, or equivalently as a number in decimal "dot" notation written $r = i.d$, where $i \in \mathbb{N}$ is the *integer part* and $d \in \mathbb{N}$ is the *decimal part*.

- a *public* `const Rational zero(0.0);`
- a *public* `friend` method `operator*` that takes two arguments of type `const &Rational` and returns a result of type `&Rational`;
- a *public* `friend` method `operator/` that takes two arguments of type `const &Rational` and returns a result of type `&Rational`;
- a *public* `const Rational one(1.0);`
- a *public* `friend` method `operator==` that takes two arguments of type `const &Rational` and returns a result of type `bool`;

- a class `Equation` representing a linear fix-point equation on the `Rationals`; this class must have a *private* member `structure` of type `*Rational`;

We must also provide the *methods* `solve` for both `Rational` and `Equation<Rational>` classes.

8 Discussion

8.1 Purpose of `structure`

The class `Equation` representing a linear fix-point equation on the `Rationals` actually does not need to have the `structure` member of type `*Rational`. In fact, this member comes in handy only when carrying out the implementation for arbitrary semi-rings.

If one does not wish carry out the implementation for arbitrary semi-rings, this member should be inherited by the instance `Equation<Rational>` from a generic class `Equation<Semi_Ring>`. In the latter, the `structure` member is of type `Semi_Ring`, the type parameter.

The taxonomy of algebraic structures of semi-rings, or special cases of semi-rings, can easily be encoded as a class taxonomy deriving from a base class `Dual_Structure`:[14]

Abstract class hierarchies can now easily be defined following the formal specification given by the mathematical ontologies in Section 5. There are two kinds of classes: one for the monoidal algebras shown in Figure 5.1.5, and the other for the binoidal algebras Figure 5.2.6.

In the generic linear-equation-solver software architecture we are defining, each of these classes is parameterized by the type variable `Domain`, of its (private) representation. For example, the class `Rational` can be defined as a subclass of `Abelian_Ring<double>`.[15]

Figures 7–9 show a skeleton for a C++ implementation of a solver using dynamic programming.[16]

8.2 Purpose of `Rational::solve(Equation)`

We would not need to worry about invoking `Rational::solve(Equation)` unless the system also means to allow the scheduling of the simultaneous resolution of several systems from the context of a given semi-ring structure. Only then is this method needed.

8.3 Testing the design

Since $\mathbb{Q}$ is not quite a ring (because 0 has no multiplicative inverse), we must test whether `a[0][0]` is equal to `structure.one`. If so, the equation $x = x + b$ is solvable only if the structure is an additive semi-

[14] We do not have to include all these classes, of course, unless we actually want to implement a complete API library...

[15] Assuming that we use a `double` to represent a rational number in $\mathbb{Q}$.

[16] Please note the "informal" C++ syntax. . . This is just a program skeleton, not a complete solution.

```
// FILE. . . . . lineq.h

#ifndef LINEQ_H
#define LINEQ_H

template <class Structure>
class System;

template <class Structure>
class Equation
{
  Structure *a;
  Structure *b,
  Structure *x;

  bool left;

public:

  Structure *a () { return a; }
  Structure *b () { return b; }
  Structure *x () { return x; }

  bool isLeft () { return left; }

  Equation (Structure &a, Structure &b, bool left=true)
    : a      (a)
    , b      (b)
    , left (left)
    {
      solve();
    }

  Equation (System<Structure> s, bool left=true)
    {
      this = s.solve(left);
    }

  Equation *solve ()
    {
      x = isLeft() ? a().quasi_inverse() * b()
                   : b() * a().quasi_inverse();

      return this;
    }
};
```

Figure 7: Header File—Part I

```
const int DefaultNumberOfEquations = 2;
const int DefaultNumberOfUnknowns  = 2;

template <class Structure>
class System
{
  int m = DefaultNumberOfEquations;
  int n = DefaultNumberOfUnknowns;

  Structure* a[m][n];
  Structure* b[m];
  Structure* x[n];

  bool left = true;

public:

  int numberOfEquations () { return m; }
  int numberOfUnknowns  () { return n; }

  Structure* a[][] () { return a; }
  Structure* b[]   () { return b; }
  Structure* x     () { return x; }

  bool isLeft () { return left; }

  System (Structure* a[], Structure* b[], bool left)
    : m      (sizeof(b))
    , n      (sizeof(a)/m)
    , a      (a)
    , b      (b)
    , left (left)
    {
      if (m == 0 | m != n) exit(1);
    }

  Equation<Structure> *solve ();

};

#endif
```

Figure 8: Header File—Part II

```
#include "lineq.h"

template <class Structure>
Equation<Structure> *System<Structure>::solve ()
{
  Structure* newa[][], newb[];
  Equation<Structure> *eq;
  int i,j;

  if (m == 1) return new Equation<Structure>(a[0][0],b[0],left);

  newa = new Structure[m-1][n-1];
  newb = new Structure[m-1];

  Structure qi = a[0][0].quasi_inverse();

  if (isLeft())
    for (i=1;i<=m-1;i++)
      {
        for (j=1;j<=n-1;j++) newa[i-1][j-1] = a[i][j] + a[i][0] * qi * a[0][j];
        newb[i-1] = b[i] + a[i][0] * qi * b[0];
      }
  else
    for (i=1;i<=m-1;i++)
      {
        for (j=1;j<=n-1;j++) newa[i-1][j-1] = a[i][j] + a[i][0] * a[0][j] * qi;
        newb[i-1] = b[i] + a[i][0] * b[0] * qi;
      }

  eq = new Equation<Structure>(new System<Structure>(newa,newb,left),left);

  return new Equation<Structure>(a[0][0],
                                 b[0] + (left ? a[0][1] * eq->x
                                              : eq->x * a[0][1]),
                                 left);
}
```

Figure 9: Implementation File

$$\begin{array}{ccccccccccccc}
x_0 & = & x_0a_{00} & + & \cdots & + & x_ja_{0j} & + & \cdots & + & x_{n-1}a_{0(n-1)} & + & b_0 \\
 & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\
x_i & = & x_0a_{i0} & + & \cdots & + & x_ja_{ij} & + & \cdots & + & x_{n-1}a_{i(n-1)} & + & b_i \\
 & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\
x_{m-1} & = & x_0a_{(m-1)0} & + & \cdots & + & x_ja_{(m-1)j} & + & \cdots & + & x_{n-1}a_{(m-1)(n-1)} & + & b_{m-1}
\end{array}$$

Figure 10: System of m Right-Linear Fix-Point Equations with n Unknowns

lattice—that is, has an idempotent plus (*i.e.*, such that $x + x = x$). In this case, any x such that $b \leq_+ x$ is a solution. The alternatives are:

- abort solving;
- if underdefined, give a parameterized solution;
- if overdefined:
  - solve for the square subsystem, and check whether or not the partial solution satisfies the outstanding equations;
  - solve for the *least-squares*.[17]

Finally, let us note that the skeleton given above can solve only for *well-defined* systems, and aborts otherwise. One should use exceptions for a more graceful control.

Appendix

A Right Linear Equations

A system of m right linear equations with n unknowns in fix-point form is shown in Figure 10.

The right version of all that was done for the *left* system in Figure 1 is of course valid for the *right* system in Figure 10. Namely, the base case $(k = 0)$: for all $i = 0, \ldots, m-1$ and all $j = 0, \ldots, n-1$,

$$a'^0_{ij} = a_{ij} \tag{A.1}$$

and, for all $i = 0, \ldots, m-1$,

$$b'^0_i = b_i. \tag{A.2}$$

For all k, $k = 0, \ldots, m-1$, we have,

$$S'_k = \left\{ x_i = \sum_{j=k}^{n-1} x_j a^k_{ij} + b^k_i \right\}_{i=k}^{m-1}. \tag{A.3}$$

[17]The *least-square* approximant of a system in canonical form $Ax + b = 0$ that is overdefined is the well-defined system $A^tAx + A^tb = 0$. This system is always square and can therefore be solved. Its solution x_{lq} is such that its "distance" from any solution x of the overdefined system $Ax + b = 0$—i.e., the inner product $(x - x_{lq})^t(x - x_{lq})$—is minimal; that is, $\forall x, \emptyset \leq_+ x - x_{lq}$.

Expression (A.3) can be rewritten as:

$$S'_k = \{x_k = x_k\alpha'_k + \beta'_k\} \cup S'_{k+1} \tag{A.4}$$

where, for $k = 0, \ldots, m-1$:

$$\alpha'_k = {a'}^k_{kk}, \tag{A.5}$$

$$\beta'_k = {b'}^k_k + \sum_{j=k+1}^{n-1} x_j {a'}^k_{ij}. \tag{A.6}$$

such that, for all $i = k, \ldots, m-1$ and all $j = k+1, \ldots, n-1$:

$${a'}^k_{ij} = \begin{cases} {a'}_{ij} & \text{if } k = 0, \\ {a'}^{k-1}_{ij} + {a'}^{k-1}_{i(k-1)}{a'}^{k-1}_{(k-1)j}{\alpha'}^*_{k-1} & \text{if } 0 < k < m; \end{cases} \tag{A.7}$$

and for all $i = 1, \ldots, m-1$:

$${b'}^k_i = \begin{cases} b_i & \text{if } k = 0, \\ {b'}^{k-1}_i + {a'}^{k-1}_{i(k-1)}{b'}^{k-1}_{k-1}{\alpha'}^*_{k-1} & \text{if } 0 < k < m. \end{cases} \tag{A.8}$$

Hence, the equation:

$$x_k = x_k\alpha'_k + \beta'_k \tag{A.9}$$

in Expression (A.4) is solved by:

$$x_k = \beta'_k {\alpha'}^*_k. \tag{A.10}$$

After $m-1$ steps, we obtain:

$$S'_{m-1} = \{x_{m-1} = x_{m-1}\alpha'_{m-1} + \beta'_{m-1}\}. \tag{A.11}$$

where,

$$\alpha'_{m-1} = {a'}^{m-1}_{(m-1)(m-1)}, \tag{A.12}$$

$$\beta'_{m-1} = {b'}^{m-1}_{m-1} + \sum_{j=m}^{n-1} x_j {a'}^{m-1}_{ij}. \tag{A.13}$$

The equation:

$$x_{m-1} = x_{m-1}\alpha'_{m-1} + \beta'_{m-1} \tag{A.14}$$

in Expression (A.11) is solved by:

$$x_{m-1} = \beta'_{m-1}{\alpha'}^*_{m-1}. \tag{A.15}$$

References

[1] BANCILHON, F., MAIER, D., SAGIV, Y., AND ULLMAN, J. Magic sets and other strange ways to implement logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems* (1986), pp. 1–15.

[2] CARRÉ, B. *Graphs and Networks.* Oxford Applied Mathematics and Computing Science Series. Oxford University Press, Clarendon Press, Oxford, UK, 1979.

[3] COUSOT, P., AND COUSOT, R. Abstract interpretation and application to logic programs. *Journal of Logic Pogramming 13*, 2 & 3 (1992), 103–179.

[4] EINSTEIN, A. *Relativity—The Special and the General Theory.* Crown Publishers, Inc., New York, NY, USA, 1961.

[5] MUCHNICK, S. S., AND JONES, N. D. *Program Flow Analysis: Theory and Application.* Prentice Hall, 1981.