Galois Theory of Algorithms

Noson S. Yanofsky[*]

November 2, 2010

Abstract

Many different programs are the implementation of the same algorithm. This makes the collection of algorithms a quotient of the collection of programs. Similarly, there are many different algorithms that implement the same computable function. This makes the collection of computable functions into a quotient of the collection of algorithms. Algorithms are intermediate between programs and functions:

Programs $\twoheadrightarrow$ Algorithms $\twoheadrightarrow$ Functions.

Galois theory investigates the way that a subobject sits inside an object. We investigate how a quotient object sits inside an object. By looking at the Galois group of programs, we study the the intermediate types of algorithms possible. Along the way, we formalize the intuition that one program can be substituted for another if they are the same algorithms.

1 Introduction

In this paper we continue the work in [17] where we began a study of formal definitions of algorithms. (Knowledge of that paper is not necessary for this paper.) The previous paper generated some interest in the community. Yuri I. Manin looked at the structure of programs and algorithms from the operad/PROP point of view [9] (Chapter 9). See also [10, 11] where it is discussed in context of renormalization. The work is currently being extended from primitive recursive functions to all recursive functions in [12]. Ximo Diaz Boils has looked at these constructions in relations to earlier papers such as [2, 8, 16]. The paper has also been criticized in [1] (which will be discussed below.)

Figure 1 motivates the formal definition of algorithms.

On the bottom is the set of computable functions. Two examples of computable functions are given: the sorting function and the find max function. On top of the diagram is the set of programs. To each computable function on the bottom, a cone shows the corresponding set of programs that implement that function. Four such programs that implement the sorting function have

[*]Department of Computer and Information Science, Brooklyn College CUNY, Brooklyn, N.Y. 11210. And Computer Science Department, The Graduate Center CUNY, New York, N.Y. 10016. e-mail: noson@sci.brooklyn.cuny.edu

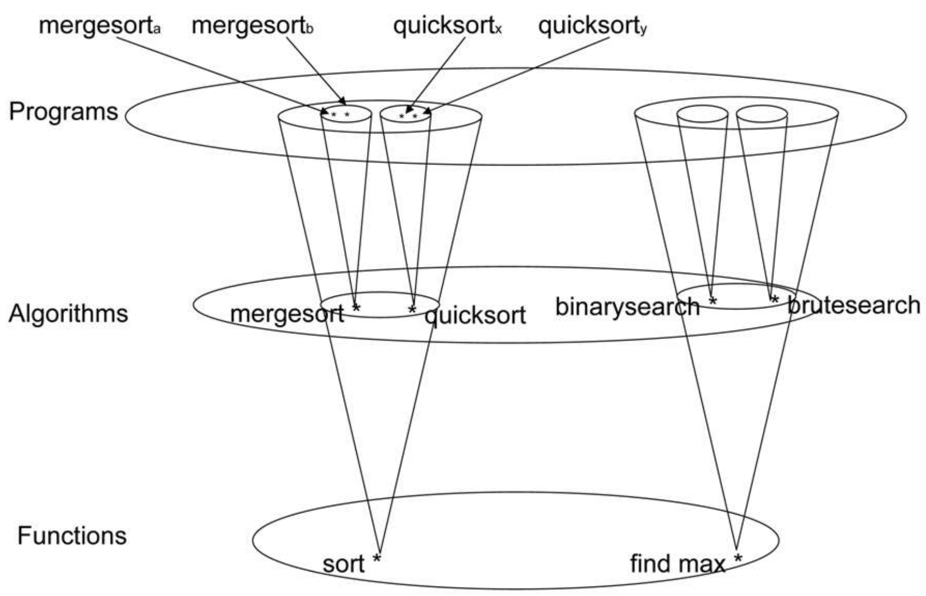


Figure 1: Programs, Algorithms and Functions.

been highlighted: $\texttt{mergesort}_a$, $\texttt{mergesort}_b$, $\texttt{quicksort}_x$ and $\texttt{quicksort}_y$. One can think of $\texttt{mergesort}_a$ and $\texttt{mergesort}_b$ as two implementations of the mergesort algorithm written by Ann and Bob respectively. These two programs are obviously similar but they are not the same. In the same way, $\texttt{quicksort}_x$ and $\texttt{quicksort}_y$ are two different implementations of the quicksort algorithm. These two programs are similar but not the same. We shall discuss in what sense they are "similar." Nevertheless programs that implement the mergesort algorithm are different than programs that implement the quicksort algorithm. This leads us to having algorithms as the middle level of Figure 1. An algorithm is to be thought of as an equivalence class of programs that implement the same function. The mergesort algorithm is the set of all programs that implement mergesort. Similarly, the quicksort algorithm is the set of all programs that implement quicksort. The set of all programs are partitioned into equivalence classes and each equivalence class corresponds to an algorithm. This gives a surjective map from the set of programs to the set of algorithms.

One can similarly partition the set of algorithms into equivalence classes. Two algorithms are deemed equivalent if they perform the same computable function. This gives a surjective function from the set of algorithms to the set of programs.

This paper is employing the fact that equivalence classes of programs have more manageable structure than the original set of programs. We will find that the set of programs does not have much structure at all. In contrast, types of algorithms have better structure and the set of computable functions have a very strict structure.

The obvious question is, what are the equivalence relation that say when two programs are "similar?" In [17] a single tentative answer was given to this question. Certain relations were described that seem universally agreeable. Using these equivalence relations, the set of algorithms have the structure of a category (composition) with a product (bracket) and a natural number object (a categorical way of describing recursion.) Furthermore, we showed that with these equivalence relations, the set of algorithms has universal properties. See [17] for more details.

Some of the relations that describe when two programs are "similar" were:

- One program might perform $Process_1$ first and then perform an unrelated $Process_2$ after. The other program might perform the two unrelated processes in the opposite order.

- One program will perform a certain process in a loop n times and the other program will "unwind the loop" and perform it $n-1$ times and then perform the the process again outside the loop.

- One program might perform two unrelated processes in one loop, and the other program might perform each of these two processes in its own loops.

A criticism of [17] was given in [1]. In that paper, the subjectivity of the question as to when two programs are considered equivalent was emphasized. While writing [17], we were aware that the answer to this question is a subjective decision (hence the word "Towards" in the title), we nevertheless described the structure of algorithms in that particular case. In this paper we answer that criticism by looking at the *many* different sets of equivalence relations that one can have. It is shown that with every set of equivalence relations we get a certain structure.

The main point of this paper is to explore the set of intermediate structures between programs and computable functions using the techniques of Galois theory. In Galois theory, intermediate fields are studied by looking at automorphism of fields. Here we study intermediate algorithmic structures by looking at automorphism of programs.

A short one paragraph review of Galois theory is in order. Given a polynomial with coefficients in a field F, we can ask if there is a solution to the polynomial in an extension field E. One examines the group of automorphisms of E that fix F, i.e., automorphisms $\phi : E \longrightarrow E$ such that for all $f \in F$ we have $\phi(f) = f$.

$$\begin{tikzcd}
E \arrow[rrrr, "\phi"] & & & & E \\
& F_H \arrow[ul, hook] \arrow[urrr, hook] \arrow[rr, "\psi"] & & F_H & \\
& & F \arrow[uull, hook] \arrow[ul] \arrow[ur] \arrow[uurr, hook] & &
\end{tikzcd} \tag{1}$$

This group is denoted $Aut(E/F)$. For every normal subgroup H of $Aut(E/F)$ there is an intermediate field $F \subseteq F_H \subseteq E$. And similarly for every intermediate field $F \subseteq K \subseteq E$ there is a subgroup $H_K = Aut(K/F)$ of $Aut(E/F)$. This correspondence is the essence of the Fundamental Theorem of Galois Theory which says that the lattice of subgroups of $Aut(E/F)$ is isomorphic to the duel lattice of intermediate fields between F and E. The properties of $Aut(E/F)$ mimic the properties of the fields. The group is "solvable" if and only if the polynomial is "solvable."

In order to understand intermediate algorithmic structures we study automorphisms of programs. Consider all automorphisms of programs that respect functionality. Such automorphisms can be thought of as ways of swapping programs for other programs that perform the same function.

$$\begin{tikzcd}
\text{Progs} \arrow[rrrr, "\phi"] \arrow[dr, two heads, "R"] \arrow[ddrr, two heads, "Q"'] & & & & \text{Progs} \arrow[dl, two heads, "R"'] \arrow[ddll, two heads, "Q"] \\
& \text{Algorithms} \arrow[rr, "\psi"] \arrow[dr, two heads, "S"] & & \text{Algorithms} \arrow[dl, two heads, "S"'] & \\
& & \text{Functs} & &
\end{tikzcd} \tag{2}$$

A subgroup of the group of all automorphisms is going to correspond to an intermediate structure. And in the other direction, an intermediate algorithmic structure will correspond to a subgroup. This will be the essence of the fundamental theorem of Galois theory of algorithms. This theorem formalizes the intuitive notion that two programs can be switched for one another if they are considered to be the same algorithm. If you consider every program to be its own algorithm than there is no swapping different programs. The other extreme is if you consider two programs to be equivalent when they perform the same function. In this case you can swap many programs for other programs. We study all the intermediate possibilities.

Notice that the vertical arrows in Diagram (2) go the opposite direction than the arrows in Diagram (1) and are surjections rather than injections. In fact the proofs in this paper are similar to the ones in classical Galois theory as long as you stand on your head. We resist the urge to call this work "co-Galois theory."

All this is somewhat abstract. What type of programs are we talking about? What type of algorithmic structures are we dealing with? How will our descriptions be specified? Rather than choosing one programming language to the exclusion of others, we look at a language of descriptions of primitive recursive functions. We choose this language because of its beauty, its simplicity of presentation, and the fact that most readers are familiar with this language. The language of descriptions of primitive recursive functions has only three operations: Composition, Bracket, and Recursion. We are limiting ourselves to the set of primitive recursive functions as opposed to all computable functions for ease. By limiting ourselves, we are going to get a proper subset of all algorithms. Even though we are, for the present time, restricting ourselves, we feel that the results obtained are interesting in their own right. There is an ongoing project to extend this work to all recursive functions [12].

Another way of looking at this work is from the homotopy point of view. We can think of the set of programs as a graph enriched over groupoids. In detail, the 0-cells are the powers of the natural number (types), the 1-cells are the programs from a power of natural numbers to a power of natural numbers. There is a 2-cell from one program two another program if and only if they are "essentially the same". That is the 2-cells are the equivalence relations. By the symmetry of the equivalence relations, the 2-cells are a groupoid. Now we take the quotient, or fraction category where we identify the programs at the end of the equivalences. This is the graph or category of algorithms. From this perspective we can promote the extremely sexy mantra:

"Algorithms are the homotopy category of programs."

This is a step towards

"Semantics is the homotopy category of syntax."

Much work remains to be done.

Another way of viewing this work is about composition. Compositionality has for many decades been recognized as one of the most valuable tools of software engineering.

There are different levels of abstractions that we use when we teach computation or work in building computers, networks, and search engines. There are programs, algorithms, and functions. Not all levels of abstraction of computation admit useful structure. If we take programs to be the finest level then we may find it hard to compose programs suitably. But if we then pass to the abstract functions they compute, again we run into trouble. In between these two extremes —extreme concreteness and extreme abstractness— there can be many levels of abstraction that admit useful composition operations unavailable at either extreme.

It is our goal here to study the many different levels of algorithms and to understand the concomitant different possibilities of composition. We feel that this work can have great potential value for software engineering.

Yet another way of viewing this work is an application and a variation of some ideas from universal algebra and model theory. In the literature, there is some discussion of Galois theory for arbitrary universal algebraic structures ([3] section II.6) and and model-theoretic structures ([5, 4, 6, 13].) In broad philosophical terms, following the work of Galois and Klein's *erlangen* program, an object can be defined by looking at its symmetries. Primitive recursive programs are here considered as a universal algebraic structure where the generators of the structure are the initial functions while composition, bracket and recursion are the operations. This work examines the symmetries of such programs and types of structures that can be defined from those symmetries.

Section 2 reviews primitive recursive programs and the basic structure that they have. In Section 3 we define an algorithmic universe as the minimal structure that a set of algorithms can have. Many examples are given . The main theorems in this paper are found in Section 4 where we prove the Fundamental Theorem of Galois Theory. We conclude with a list of possible ways that this work might progress in the future.

Acknowledgment. I thank Ximo Diaz Boils, Leon Ehrenpreis (of blessed memory), Roman Kossak, Florian Lengyel, Dustin Mulcahey, Vaughan Pratt, and Lou Thrall for helpful discussions.

2 Programs

Consider the structure of all descriptions of primitive recursive functions. Throughout this paper we shall use the words "description" and "program" interchangeably. The descriptions shall form a graph denoted **PRdesc**. The objects of the graph are powers of natural numbers $\mathbb{N}^0, \mathbb{N}^1, \mathbb{N}^2, \ldots, \mathbb{N}^n, \ldots$ and the morphisms are descriptions of primitive recursive functions. In particular, there exists descriptions of initial functions: the null function $z : \mathbb{N} \longrightarrow \mathbb{N}$, the successor function $s : \mathbb{N} \longrightarrow \mathbb{N}$, and the projection functions, i.e., for all $n \in \mathbb{N}$ and for all $1 \leq i \leq n$ there are distinguished descriptions $\pi_i^n : \mathbb{N}^n \longrightarrow \mathbb{N}$.

There will be three ways of composing edges in this graph:

- Composition: For $f : \mathbb{N}^a \longrightarrow \mathbb{N}^b$ and $f : \mathbb{N}^b \longrightarrow \mathbb{N}^c$, there is a
$$(g \circ f) : \mathbb{N}^a \longrightarrow \mathbb{N}^c.$$
Notice that this composition need not be associative. There is also no reason to assume that this composition has a unit.

- Recursion: For $f : \mathbb{N}^a \longrightarrow \mathbb{N}^b$ and $g : \mathbb{N}^{a+b} \longrightarrow \mathbb{N}^b$, there is a
$$(f \sharp g) : \mathbb{N}^{a+1} \longrightarrow \mathbb{N}^b.$$
There is no reason to think that this operation satisfies any universal properties or that it respects the composition or the bracket.

- Bracket: For $f : \mathbb{N}^a \longrightarrow \mathbb{N}^b$ and $g : \mathbb{N}^a \longrightarrow \mathbb{N}^c$, there is a
$$\langle f, g \rangle : \mathbb{N}^a \longrightarrow \mathbb{N}^{b+c}.$$
There is no reason to think that this bracket is a functorial (that is respects the composition) or is in any way coherent.

At times we shall use trees to specify the descriptions. The leaves of the trees will have initial functions and the internal nodes will be marked with **C**, **R** or **B** for composition, recursion and bracket as follows:

$$\begin{tikzcd}
 & g \circ f : \mathbb{N}^a \to \mathbb{N}^c & \\
 & \boxed{\mathbf{C}} \arrow[dl, dash] \arrow[dr, dash] & \\
f : \mathbb{N}^a \to \mathbb{N}^b & & g : \mathbb{N}^b \to \mathbb{N}^c
\end{tikzcd}
\quad
\begin{tikzcd}
 & h = f \sharp g : \mathbb{N}^a \times \mathbb{N} \to \mathbb{N}^b & \\
 & \boxed{\mathbf{R}} \arrow[dl, dash] \arrow[dr, dash] & \\
f : \mathbb{N}^a \to \mathbb{N}^b & & g : \mathbb{N}^a \times \mathbb{N}^b \to \mathbb{N}^b
\end{tikzcd}
\quad
\begin{tikzcd}
 & \langle f, g \rangle : \mathbb{N}^a \to \mathbb{N}^b \times \mathbb{N}^c & \\
 & \boxed{\mathbf{B}} \arrow[dl, dash] \arrow[dr, dash] & \\
f : \mathbb{N}^a \to \mathbb{N}^b & & g : \mathbb{N}^a \to \mathbb{N}^c
\end{tikzcd}$$

Just to highlight the distinction between programs and functions, it is important to realize that the following are all legitimate descriptions of the null function:

- $z : \mathbb{N} \longrightarrow \mathbb{N}$
- $(z \circ s \circ s \circ s \circ s \circ z \circ s \circ s \circ s \circ s \circ s \circ s \circ s \circ s \circ s \circ s) : \mathbb{N} \longrightarrow \mathbb{N}$
- $(z \circ (\pi_1^2 \circ \langle s, s \rangle)) : \mathbb{N} \longrightarrow \mathbb{N}$

- etc.

There are, in fact, an infinite number of descriptions of the null function.

In this paper we will need "macros", that is, certain combinations of operations to get commonly used descriptions.

There is a need to generalize the notion of a projection. The π_i^n accept n inputs and outputs one number. A multiple projection takes n inputs and outputs m outputs. Consider $\mathbb{N}^n$ and the sequence $X = \langle x_1, x_2, \ldots, x_m\rangle$ where each x_i is in $\{1, 2, \ldots, n\}$. For every X there exists $\pi^X : \mathbb{N}^n \to \mathbb{N}^m$ as

$$\pi^X = \langle \pi^n_{x_1}, \langle \pi^n_{x_2}, \langle \ldots, \langle \pi^n_{x_{m-1}}, \pi^n_{x_m}\rangle\rangle \ldots\rangle.$$

In other words, π^X outputs the proper numbers in the order described by X. In particular

- If $I = \langle 1, 2, 3, \ldots, n\rangle$ then $\pi^I : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ will be a description of the identity function.
- If $X = \langle 1, 2, 3, \ldots, n, 1, 2, 3, \ldots, n\rangle$ then $\pi^X = \triangle = \pi^a_{a+a} : \mathbb{N}^n \longrightarrow \mathbb{N}^{n+n}$ is the diagonal map.
- For $a \le b \in \mathbb{N}$
$$X = \langle b, b+1, b+2, \ldots b+a, 1, 2, 3, \ldots b-1\rangle,$$
then π^X will be the twist operator which swaps the first a elements with the second b elements. Then by abuse of notation, we shall write
$$\pi^X = tw = \pi^{a+b}_{b+a} : \mathbb{N}^{a+b} \longrightarrow \mathbb{N}^{a+b}.$$

Whenever possible, we shall be ambiguous with superscripts and subscripts.

Concomitant with the bracket operation is the product operation. A product of two maps is defined for a given $f : \mathbb{N}^a \to \mathbb{N}^b$ and $g : \mathbb{N}^c \to \mathbb{N}^d$ as

$$f \times g : \mathbb{N}^a \times \mathbb{N}^c \to \mathbb{N}^b \times \mathbb{N}^d.$$

The product can be defined using the bracket as

$$f \times g = \langle f \circ \pi^{a+c}_a, g \circ \pi^{a+c}_c\rangle.$$

Given the product and the diagonal, $\triangle = \pi^a_{a+a}$, we can define the bracket as

$$\begin{tikzcd}
\mathbb{N}^a \arrow[rr, "\langle f,g\rangle"] \arrow[dr, "\triangle"'] & & \mathbb{N}^b \times \mathbb{N}^c \\
& \mathbb{N}^a \times \mathbb{N}^a. \arrow[ur, "f\times g"'] &
\end{tikzcd}$$

Since the product and the bracket are derivable from each other, we shall use them interchangeably.

That is enough about the graph of descriptions.

Related to descriptions of primitive recursive functions is the set of primitive recursive *functions*. The set of functions has a lot more structure than **PRdesc**.

PRfunc is the category of primitive recursive functions. The objects of this category are powers of natural numbers $\mathbb{N}^0, \mathbb{N}^1, \mathbb{N}^2, \ldots, \mathbb{N}^n, \ldots$ and the morphisms are primitive recursive functions. In particular, there are specific maps $z : \mathbb{N} \longrightarrow \mathbb{N}$, $s : \mathbb{N} \longrightarrow \mathbb{N}$ and for all $n \in \mathbb{N}$ and for all $1 \leq i \leq n$ there are projection maps $\pi^n_i : \mathbb{N}^n \longrightarrow \mathbb{N}$. Since composition of primitive recursive functions is associative and the identity functions $id = \pi^n_n : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ are primitive recursive and act as units for composition, **PRfunc** is a genuine category. **PRfunc** has a categorically coherent Cartesian product $\times$. Furthermore, **PRfunc** has a strong natural number object. That is, for every $f : \mathbb{N}^a \longrightarrow \mathbb{N}^b$ and $g : \mathbb{N}^a \times \mathbb{N}^b \longrightarrow \mathbb{N}^b$ there exists a *unique* $h = f\sharp g : \mathbb{N}^a \times \mathbb{N} \longrightarrow \mathbb{N}^b$ that satisfies the the following two commutative diagrams

$$\begin{tikzcd}
\mathbb{N}^a \times \mathbb{N} \arrow[r, "id \times z"] \arrow[d, "\pi^{a+1}_a"'] & \mathbb{N}^a \times \mathbb{N} \arrow[d, "h"] & \mathbb{N}^a \times \mathbb{N} \arrow[r, "id \times s"] \arrow[d, "\langle \pi^{a+1}_a, h \rangle"'] & \mathbb{N}^a \times \mathbb{N} \arrow[d, "h"] \\
\mathbb{N}^a \arrow[r, "f"'] & \mathbb{N}^b & \mathbb{N}^a \times \mathbb{N}^b \arrow[r, "g"'] & \mathbb{N}^b
\end{tikzcd} \qquad (3)$$

This category of primitive recursive functions was studied extensively by many people including [2, 8, 16, 14, 17]. It is known to be the initial object in the 2-category of categories, with products and strict natural number objects. Other categories in that 2-category will be primitive recursive functions with oracles. One can think of the oracles as functions put on the leaves of the trees besides the initial functions.

There is a surjective graph morphism $Q : \mathbf{PRdesc} \longrightarrow \mathbf{PRfunc}$ that takes $\mathbb{N}^n$ to $\mathbb{N}^n$, i.e., is identity on objects. Q takes descriptions of primitive recursive functions in **PRdesc** to the functions they describe in **PRfunc**. Since there are, in general, many descriptions of a primitive recursive function, Q is surjective on morphisms. Another way to say this is that **PRfunc** is a quotient of **PRdesc**.

Algorithms will be graphs that are "between" **PRdesc** and **PRfunc**.

3 Algorithms

In the last section we saw the type of structure the set of programs form. In this section we look at the weakest type of structures a set of algorithms can have.

Definition 1 *A* **primitive recursive (P.R.) algorithmic universe***,* **PRalg***, is a graph whose objects are the powers of natural numbers* $\mathbb{N}^0, \mathbb{N}^1, \mathbb{N}^2, \ldots, \mathbb{N}^n, \ldots$. *We furthermore require that there exist graph morphisms R and S that are the identity on objects and that make the following diagram of graphs commute:*

$$\begin{tikzcd}
\mathbf{PRdesc} \arrow[dr, "R"] \arrow[dd, "Q"'] & & . \\
& \mathbf{PRalg} \arrow[dl, "S"] & \\
\mathbf{PRfunc}. & &
\end{tikzcd} \tag{4}$$

The image of the initial functions under R will be distinguished objects in **PRalg***:*

- $z : \mathbb{N} \longrightarrow \mathbb{N}$
- $s : \mathbb{N} \longrightarrow \mathbb{N}$ *and*
- *for all $n \in \mathbb{N}$ and for all $1 \leq i \leq n$ there are projection maps $\pi_i^n : \mathbb{N}^n \longrightarrow \mathbb{N}$*

A P.R. algorithmic universe *might* have the following operations: (Warning: these are not functors because we are not dealing with categories.)

- Composition: For $f : \mathbb{N}^a \longrightarrow \mathbb{N}^b$ and $g : \mathbb{N}^b \longrightarrow \mathbb{N}^c$, there is a
$$(g \circ f) : \mathbb{N}^a \longrightarrow \mathbb{N}^c.$$
- Recursion: For $f : \mathbb{N}^a \longrightarrow \mathbb{N}^b$ and $g : \mathbb{N}^{a+b} \longrightarrow \mathbb{N}^b$, there is a
$$(f \sharp g)\rangle : \mathbb{N}^{a+1} \longrightarrow \mathbb{N}^b$$
- Bracket: For $f : \mathbb{N}^a \longrightarrow \mathbb{N}^b$ and $g : \mathbb{N}^a \longrightarrow \mathbb{N}^c$, there is a
$$\langle f, g\rangle : \mathbb{N}^a \longrightarrow \mathbb{N}^{b+c}$$

These operations are well defined for programs but need not be well defined for equivalence classes of programs. There was never an insistence that our equivalence relations be congruences (i.e. respect the operations). We shall study when these operations exist.

Notice that although the Q graph morphism preserves the composition, bracket and recursion operators, we do not insist that R and S preserve them. We shall see that this is too strict of a requirement.

Definition 2 *Let* **PRalg** *be a P.R. algorithmic universe. A* **P.R. quotient algorithmic universe** *is a P.R. algorithmic universe* **PRalg′** *and an identity on objects, surjection on edges graph map T that makes all of the the following squares and triangles commute*

$$\begin{tikzcd}
\mathbf{PRdesc} \arrow[r, two heads, "R"] \arrow[d, two heads, "Q"'] \arrow[dr, two heads, "R'" near start] & \mathbf{PRalg} \arrow[d, two heads, "T"] \arrow[dl, two heads, "S"' near start] \\
\mathbf{PRfunc} & \mathbf{PRalg'}. \arrow[l, two heads, "S'"]
\end{tikzcd} \tag{5}$$

Examples of P.R. algorithmic universe abound:
Example: PRdesc is the primary example. In fact, all our examples will be quotients of this algorithmic universes. Here $R = id$ and $S = Q$. □

Example: PRfunc is another example of an algorithmic universe. Here $R = Q$ and $S = id$. □

Example: PRalgC is a quotient of **PRdesc**. This is constructed by adding the following relation:
For any three composable maps f, g and h, we have

$$h \circ (g \circ f) \sim (h \circ g) \circ f. \tag{6}$$

In terms of trees, we say that the following trees are equivalent:

$$\begin{tikzcd}[column sep=small]
& \begin{array}{c} h\circ(g\circ f):\mathbb{N}^a\to\mathbb{N}^d \\ \boxed{\mathbf{C}} \end{array} \arrow[dl, dash] \arrow[dr, dash] & & \sim & & & \begin{array}{c} (h\circ g)\circ f:\mathbb{N}^a\to\mathbb{N}^d \\ \boxed{\mathbf{C}} \end{array} \arrow[dll, dash] \arrow[dr, dash] & & \\
\begin{array}{c} g\circ f:\mathbb{N}^a\to\mathbb{N}^c \\ \boxed{\mathbf{C}} \end{array} \arrow[d, dash] \arrow[dr, dash] & & h:\mathbb{N}^c\to\mathbb{N}^d & & f:\mathbb{N}^a\to\mathbb{N}^b & & & \begin{array}{c} h\circ g:\mathbb{N}^b\to\mathbb{N}^d \\ \boxed{\mathbf{C}} \end{array} \arrow[dl, dash] \arrow[dr, dash] & \\
f:\mathbb{N}^a\to\mathbb{N}^b & g:\mathbb{N}^b\to\mathbb{N}^c & & & & & g:\mathbb{N}^b\to\mathbb{N}^c & & h:\mathbb{N}^c\to\mathbb{N}^d
\end{tikzcd}$$

It is obvious that the composition map in **PRalgC** is associative.
□

Example: PRalgI is also a quotient of **PRdesc** that is constructed by adding in the relations that says that the projections π^Is act like identity maps. That means for any $f : \mathbb{N}^a \to \mathbb{N}^b$, we have

$$f \circ \pi^a_a \sim f \sim \pi^b_b \circ f. \tag{7}$$

In terms of trees:

$$\begin{tikzcd}
 & f \circ \pi^a_a : \mathbb{N}^a \to \mathbb{N}^b & & \sim & f : \mathbb{N}^a \to \mathbb{N}^b & \sim & & \pi^b_b \circ f : \mathbb{N}^a \to \mathbb{N}^b & \\
 & \boxed{\mathbf{C}} \arrow[dl, dash] \arrow[dr, dash] & & & & & & \boxed{\mathbf{C}} \arrow[dl, dash] \arrow[dr, dash] & \\
\pi^a_a : \mathbb{N}^a \to \mathbb{N}^a & & f : \mathbb{N}^a \to \mathbb{N}^b & & & & f : \mathbb{N}^a \to \mathbb{N}^b & & \pi^b_b : \mathbb{N}^b \to \mathbb{N}^b
\end{tikzcd}$$

□

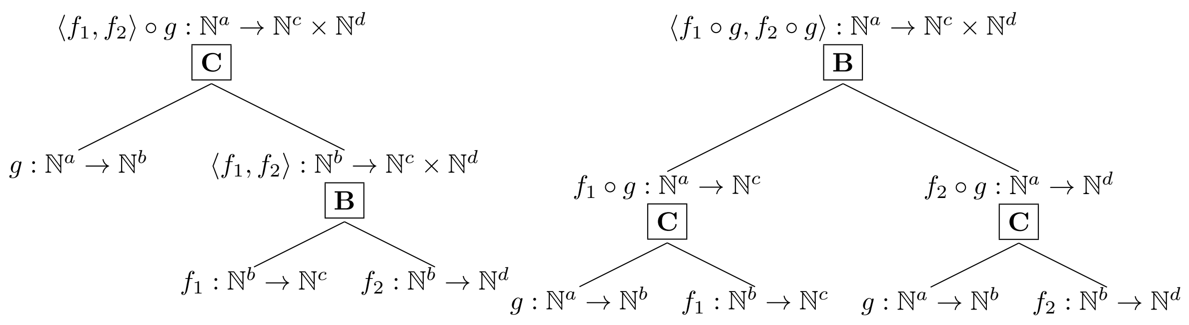
The composition map in **PRalgI** has a unit.

Example: PRalgCat is **PRdesc** with both relations (6) and (7). Notice that this ensures that **PRalgCat** is more than a graph and is, in fact, a full fledged category. □

Example: PRalgCatX is a quotient of **PRalgCat** which has a well-defined bracket/product function. We add the following relations to **PRalgCat**:

- The bracket is associative. For any three maps f, g, and h with the same domain, we have
$$\langle\langle f, g\rangle, h\rangle \sim \langle f, \langle g, h\rangle\rangle$$
In terms of trees, this amounts to
$$\begin{tikzcd}
 & & \langle\langle f, g\rangle h\rangle : \mathbb{N}^a \to \mathbb{N}^b \times \mathbb{N}^c \times \mathbb{N}^d & \sim & & \langle f, \langle g, h\rangle\rangle : \mathbb{N}^a \to \mathbb{N}^b \times \mathbb{N}^c \times \mathbb{N}^d & & \\
 & & \boxed{\mathbf{B}} \arrow[dl, dash] \arrow[dr, dash] & & & \boxed{\mathbf{B}} \arrow[dl, dash] \arrow[dr, dash] & & \\
 & \langle f, g\rangle : \mathbb{N}^a \to \mathbb{N}^b \times \mathbb{N}^c & & h : \mathbb{N}^a \to \mathbb{N}^d & f : \mathbb{N}^a \to \mathbb{N}^b & & \langle g, h\rangle : \mathbb{N}^b \to \mathbb{N}^c \times \mathbb{N}^d & \\
 & \boxed{\mathbf{B}} \arrow[dl, dash] \arrow[d, dash] & & & & & \boxed{\mathbf{B}} \arrow[d, dash] \arrow[dr, dash] & \\
f : \mathbb{N}^a \to \mathbb{N}^b & g : \mathbb{N}^a \to \mathbb{N}^c & & & & & g : \mathbb{N}^a \to \mathbb{N}^c & h : \mathbb{N}^a \to \mathbb{N}^d
\end{tikzcd}$$

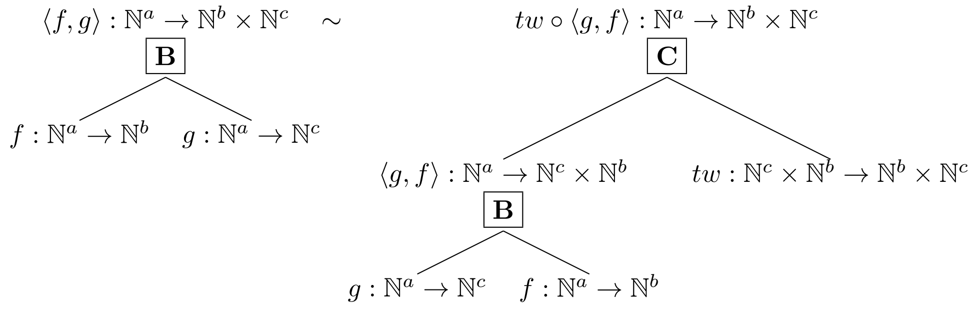
- Composition distributes over the bracket on the right. For $g : \mathbb{N}^a \to \mathbb{N}^b$, $f_1 : \mathbb{N}^b \to \mathbb{N}^c$ and $f_2 : \mathbb{N}^b \to \mathbb{N}^d$, we have
$$\langle f_1, f_2\rangle \circ g \sim \langle f_1 \circ g, f_2 \circ g\rangle. \tag{8}$$
In terms of trees, this amounts to saying that these trees are equivalent:


$\langle f_1, f_2\rangle \circ g : \mathbb{N}^a \to \mathbb{N}^c \times \mathbb{N}^d$
$\boxed{\mathbf{C}}$
$g : \mathbb{N}^a \to \mathbb{N}^b$
$\langle f_1, f_2\rangle : \mathbb{N}^b \to \mathbb{N}^c \times \mathbb{N}^d$
$\boxed{\mathbf{B}}$
$f_1 : \mathbb{N}^b \to \mathbb{N}^c$
$f_2 : \mathbb{N}^b \to \mathbb{N}^d$
$\langle f_1 \circ g, f_2 \circ g\rangle : \mathbb{N}^a \to \mathbb{N}^c \times \mathbb{N}^d$
$\boxed{\mathbf{B}}$
$f_1 \circ g : \mathbb{N}^a \to \mathbb{N}^c$
$\boxed{\mathbf{C}}$
$g : \mathbb{N}^a \to \mathbb{N}^b$
$f_1 : \mathbb{N}^b \to \mathbb{N}^c$
$f_2 \circ g : \mathbb{N}^a \to \mathbb{N}^d$
$\boxed{\mathbf{C}}$
$g : \mathbb{N}^a \to \mathbb{N}^b$
$f_2 : \mathbb{N}^b \to \mathbb{N}^d$

- The bracket is almost commutative. For any two maps f and g with the same domain,

$$\langle f, g\rangle \sim tw \circ \langle g, f\rangle.$$

In terms of trees, this amounts to


$\langle f, g\rangle : \mathbb{N}^a \to \mathbb{N}^b \times \mathbb{N}^c$
$\sim$
$\boxed{\mathbf{B}}$
$f : \mathbb{N}^a \to \mathbb{N}^b$
$g : \mathbb{N}^a \to \mathbb{N}^c$
$tw \circ \langle g, f\rangle : \mathbb{N}^a \to \mathbb{N}^b \times \mathbb{N}^c$
$\boxed{\mathbf{C}}$
$\langle g, f\rangle : \mathbb{N}^a \to \mathbb{N}^c \times \mathbb{N}^b$
$\boxed{\mathbf{B}}$
$g : \mathbb{N}^a \to \mathbb{N}^c$
$f : \mathbb{N}^a \to \mathbb{N}^b$
$tw : \mathbb{N}^c \times \mathbb{N}^b \to \mathbb{N}^b \times \mathbb{N}^c$

- Twist is idempotent.

$$tw_{\mathbb{N}^a,\mathbb{N}^b} \circ tw_{\mathbb{N}^a,\mathbb{N}^b} \sim id = \pi^{a+b}_{a+b} : \mathbb{N}^a \times \mathbb{N}^b \to \mathbb{N}^a \times \mathbb{N}^b.$$

- Twist is coherent. That is, the twist maps of three elements to get along with themselves.

$$(tw_{\mathbb{N}^b,\mathbb{N}^c} \times id) \circ (id \times tw_{\mathbb{N}^a,\mathbb{N}^c}) \circ (tw_{\mathbb{N}^a,\mathbb{N}^b} \times id) \sim (id \times tw_{\mathbb{N}^a,\mathbb{N}^b}) \circ (tw_{\mathbb{N}^a,\mathbb{N}^c} \times id) \circ (id \times tw_{\mathbb{N}^b,\mathbb{N}^c}).$$

This is called the hexagon law or the third Reidermeister move. Given the idempotence and hexagon laws, it is a theorem that there is a unique twist map made of smaller twist maps between any two products of elements ([7] Section XI.4).

□

Example: PRalgCatN is a category with a natural number object. It is **PRalgCat** with the following relations:

- Left square of Diagram (3). $(f\sharp g) \circ (id \times z) \quad \sim \quad (f \circ \pi^{a+1}_a)$.
- Right square of Diagram (3). $(f\sharp g) \circ (id \times s) \quad \sim \quad (f \circ \langle \pi^{a+1}_a, (f\sharp g)\rangle$.
- Natural number object and identity. If $g = \pi^{a+b}_b : \mathbb{N}^a \times \mathbb{N}^b \longrightarrow \mathbb{N}^b$ then
$$(f\sharp\pi^{a+b}_b) \quad \sim \quad (f \circ \pi^{a+1}_a).$$
- Natural number object and composition. This is explained in Section 3.5 of [17].
$$g_1 \ddot{\circ} (f\sharp(g_2 \ddot{\circ} g_1)) \quad \sim \quad (g_1 \ddot{\circ} f)\sharp(g_1 \ddot{\circ} g_2).$$

□

Example: PRalgCatXN is a category that has both a product and natural number object. It can be constructed by adding to **PRalgCat** all the relations of **PRalgCatX** and **PRalgCatN** as well as the following relations:

- Natural number object and bracket. This is explained in Section 3.4 of [17]
$$\langle f_1, f_2\rangle\sharp(g_1 \boxtimes g_2) \quad \sim \quad \langle f_1\sharp g_1, f_2\sharp g_2\rangle.$$

□

Putting all these examples together, we have the following diagram of P.R. algorithmic universes.

$$\begin{tikzcd}
& \mathbf{PRdesc} \arrow[dl, two heads] \arrow[dr, two heads] & \\
\mathbf{PRalgC} \arrow[dr, two heads] & & \mathbf{PRalgI} \arrow[dl, two heads] \\
& \mathbf{PRalgCat} \arrow[dl, two heads] \arrow[dr, two heads] & \\
\mathbf{PRalgCatX} \arrow[dr, two heads] & & \mathbf{PRalgCatN} \arrow[dl, two heads] \\
& \mathbf{PRalgCatXN} \arrow[d, two heads] & \\
& \mathbf{PRfunc} &
\end{tikzcd}$$

There is no reason to think that this is a complete list. One can come up with infinity many more examples of algorithmic universes. We can take other permutations and combinations of the relations given here as well as new ones. Each such relations will give a different algorithmic universe.

In [17], we mentioned other relations which deal with the relationship between the operations and the initial functions. We do not mention those relations here because our central focus is the existence of well defined operations.

A word about decidability. The question is, for a given P.R. algorithmic universe determine whether or not two programs in **PRdesc** are in the same equivalence class of that algorithmic universe.

- This is very easy in the algorithmic universe **PRdesc** since every equivalence class has only one element. Two descriptions are in the same equivalence relation iff they are *exactly* the same.

- The extreme opposite is in **PRfunc**. By a restriction of Rice's theorem, there is no way to tell when two different programs/descriptions are the same primitive recursive function. So **PRfunc** is not decidable.

- In between **PRdesc** and **PRfunc** things get a little hairy. Consider **PRalgC**, i.e., the graph with associative composition. This is decidable. All one has to do is change all the contiguous sequences of compositions to associate on the left. Do this for both descriptions and then see if the two modified programs are the same.

- One can perform a similar trick for **PRalgI**. Simply eliminate all the identities and see if the two modified programs are the same.

- For **PRalgCat** one can combine the tricks from **PRalgC** and **PRalgI** to show that it is also decidable.

- It is believed that **PRalgCatX** is also decidable because of the coherence of the product. Once again, any contiguous sequences of products can be associated to the left. Also, equivalence relation (8) insures the naturality of the product so that products and compositions can "slide across" each other. Again, each description can be put into a canonical form and then see if the modified programs are the same.

- Things are not so clear for **PRalgCatN** and **PRalgCatXN**. The equivalence relations that deal with the $\sharp$ operator are far more complicated. One can think of this as the dividing line between the decidable, syntactical structure of **PRdesc** and the undecidable, semantical structure of **PRfunc**. We leave it as an open question to determine if they are decidable or not.

4 Galois Theory

An automorphism ϕ of **PRdesc** is a graph isomorphism that is the identity on the vertices (i.e., $\phi(\mathbb{N}^n) = \mathbb{N}^n$). For every $a, b \in \mathbb{N}$ ϕ basically acts on the edges between $\mathbb{N}^a$ and $\mathbb{N}^b$. We are interested in automorphisms that preserve functionality. That is, automorphisms ϕ, such that for all programs p, we have that p and $\phi(p)$ perform the same function. In terms of Diagram (2) we demand that $Q(\phi(p)) = Q(p)$. It is not hard to see that the set of all automorphism of **PRdesc** that preserve functionality forms a group. We shall denote this group as $Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$.

Theorem 1 (Fundamental theorem of Galois theory) *The lattice of subgroups of* $Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$ *is isomorphic to the dual lattice of algorithmic universes between* **PRdesc** *and* **PRfunc***.*

Proof. We shall construct inverse functions between intermediate algorithmic universes and subgroups of $Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$.

For a given algorithmic universe **PRalg**, we construct the subgroup $H_{Aut(\mathbf{PRalg}/\mathbf{PRfunc})}$.

$$\mathbf{PRalg} \qquad \mapsto \qquad H_{Aut(\mathbf{PRdesc}/\mathbf{PRalg})} \subseteq Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$$

This is the set of all automorphisms of **PRdesc** that preserve that algorithmic universe, i.e., automorphisms ϕ such that for all programs p, we have p and $\phi(p)$ are in the same equivalence class in **PRalg**. In terms of Diagram (4), this means $R(\phi(p)) = R(p)$. In order to see that it is a subgroup of $Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$, notice that if ϕ is in $H_{Aut(\mathbf{PRdesc}/\mathbf{PRfunc})}$ then we have

$$R\phi = R \qquad \Rightarrow \qquad SR\phi = SR \qquad \Rightarrow \qquad Q\phi = Q$$

which means that ϕ is in $Aut(\mathbf{PRdesc}/\mathbf{PRalg})$. In general, this subgroup fails to be normal.

The other direction goes as follows. For $H \subseteq Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$, the graph $\mathbf{PRalg}_H$ is a quotient of **PRdesc**.

$$H \subseteq Aut(\mathbf{PRdesc}/\mathbf{PRfunc}) \qquad \mapsto \qquad \mathbf{PRdesc} \twoheadrightarrow \mathbf{PRalg}_H \twoheadrightarrow \mathbf{PRfunc}$$

The vertices of $\mathbf{PRalg}_H$ are powers of natural numbers. The edges will be equivalence classes of edges from **PRdesc**. The equivalence relation $\sim_H$ is defined as

$$p \sim_H p' \qquad \text{iff} \qquad \text{there exists a } \phi \in H \text{ such that } \phi(p) = p' \tag{9}$$

The fact that $\sim_H$ is an equivalence relation follows from the fact that H is a group. In detail

- Reflexivity comes from the fact that $id \in H$.
- Symmetry comes from the fact that if $\phi \in H$ then $\phi^{-1} \in H$.

- Transitivity comes from the fact that if $\phi \in H$ and $\psi \in H$ then $\phi\psi \in H$.

Now to make sure that these two functions respect the lattice structure. If $H \subseteq H' \subseteq Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$ then there will be a surjective map

$$\mathbf{PRalg}_H \twoheadrightarrow \mathbf{PRalg}_{H'}.$$

The way to see this is to realize that there are more ϕ in H' to make different programs equivalent as described in line (9).

Going the other way, if

$$T : \mathbf{PRalg} \twoheadrightarrow \mathbf{PRalg}'$$

is a quotient algorithmic universe as in Diagram (5) then

$$H_{\mathbf{PRalg}} \subseteq H_{\mathbf{PRalg}'} \subseteq Aut(\mathbf{PRdesc}/\mathbf{PRfunc}).$$

This is obvious if you look at $\phi \in H_{\mathbf{PRalg}}$ then we have that

$$R\phi = R \qquad \Rightarrow \qquad TR\phi = TR \qquad \Rightarrow \qquad R'\phi = R'$$

which means that ϕ is also in $H_{\mathbf{PRalg}'}$.

We must show that these two operations are inverse of each other. Let us start with a subgroup $H \subseteq Aut(\mathbf{PRdesc}/\mathbf{PRfunc})$. Performing both operations we have

$$H \subseteq Aut(\mathbf{PRdesc}/\mathbf{PRfunc}) \qquad \mapsto \qquad \mathbf{PRalg}_H \qquad \mapsto \qquad H_{\mathbf{PRalg}_H}$$

We must show that $H = H_{\mathbf{PRalg}_H}$. Let $\phi \in H$. Since $\phi : \mathbf{PRdesc} \longrightarrow \mathbf{PRdesc}$ takes p to p' we have that in $\mathbf{PRalg}_H$ the following equivalence holds $p \sim_H p'$. An automorphism of $\mathbf{PRdesc}$ that preserves $\mathbf{PRalg}_H$ will take p to something that it is equivalent to in $\mathbf{PRalg}_H$. ϕ is such an automorphism, i.e., $\phi \in H_{\mathbf{PRalg}_H}$.

Now let us start with a $\mathbf{PRalg}$. Performing both operations we have

$$\mathbf{PRalg} \qquad \mapsto \qquad H_{Aut(\mathbf{PRdesc}/\mathbf{PRalg})} \qquad \mapsto \qquad \mathbf{PRalg}_{H_{Aut(\mathbf{PRdesc}/\mathbf{PRalg})}}$$

Let us consider an equivalence class $[p]$ in $\mathbf{PRalg}_{H_{Aut(\mathbf{PRdesc}/\mathbf{PRalg})}}$.

$$[p] = \{p' | \text{ there exits a } \phi \in H_{Aut(\mathbf{PRdesc}/\mathbf{PRalg})} \text{ such that } \phi(p) = p'\}$$

$= \{p' | \exists \phi$ that preserves the equivalence relation of $\mathbf{PRalg}$ and $\phi(p) = p'\}$
$= [p]_{\mathbf{PRalg}}$. The theorem is proved.

Notice that the algorithmic universes that we dealt with in this theorem does not necessarily have well-defined extra structure/operations. We simply discussed the equivalence relations of **PRdesc** and did not discuss congruences of **PRdesc**. Without the congruence, the operations of composition, bracket and recursion might not be well-defined for the equivalence classes. This is very similar to classical Galois theory where we discuss a single weak structure (fields) and discuss all intermediate objects as fields even though they might have more structure. So too, here we stick to one weak structure.

However we can further. Our definition of algorithmic universes is not carved in stone. One can go on and define, say, a *composable* algorithmic universe. Then we can make the fundamental theorem of Galois theory for composable algorithmic universes by looking at automorphisms of **PRdesc** that preserve the composition operations. That is, automorphisms ϕ such that for all programs p and p' we have that

$$\phi(p \circ p') = \phi(p) \circ \phi(p').$$

Such automorphisms also form a group and one can look at subgroups as we did in the main theorem. On the algorithmic universe side, we will have to look at equivalence relations that are congruences. That is, $\sim$ such that if $p \sim p'$ and $p'' \sim p'''$ then

$$p \circ p'' \sim p' \circ p'''.$$

Such an analogous theorem can easily be proved.

Similarly, one can define a *recursive* and a *bracket* algorithmic universes. One can still go further and ask that an algorithmic universe has two well-defined operations. In that case the automorphism will have to preserve two operations. If H is a group of automorphism, then we can denote the subgroup of automorphisms that preserve composition as H_C. Furthermore, the subgroup that preserves composition and recursion will be denoted as H_{CR}, etc. The subgroups fit into the following lattice.

$$\begin{tikzcd}
 & H & \\
H_C \arrow[ur, hook] & H_R \arrow[u, hook] & H_B \arrow[ul, hook] \\
H_{CR} \arrow[u, hook] \arrow[ur, hook] & H_{CB} \arrow[ul, hook] \arrow[ur, hook] & H_{RB} \arrow[ul, hook] \arrow[u, hook] \\
 & H_{CRB} = \{e\}. \arrow[ul, hook] \arrow[u, hook] \arrow[ur, hook] &
\end{tikzcd} \tag{10}$$

It is important to realize that it is uninteresting to require that the algorithmic universe have all three operations. The only automorphism that preserves all the operations is the identity automorphism on **PRdesc**. One can see this by remembering that the automorphisms preserve all the initial functions and if we ask them to preserve all the operations, then it must be the identity automorphism. This is similar to looking at an automorphism of a group that preserves the generators and the group operation. That is not a very interesting automorphism.

One can ask of the automorphisms to preserve all three operations but not preserve the initial operations. Similarly, when discussing oracle computation, one can ask the automorphisms to preserve all three operations and the initial functions, but not the oracle functions. All these suggestions open up new vistas of study.

5 Future Directions

Extend to all computable functions. The first thing worthy of doing is to extend this work to all computable functions from primitive recursive functions. One need only add in the minimization operator and look at its relations with the other operations. The study of such programs from our point of view is already underway in [12]. However the Galois theory perspective is a little bit complicated because of the necessity to consider partial operations. A study of [15] will, no doubt, be helpful.

Continuing with Galois Theory There are many other classical Galois theory theorems that need to be proved for our context. We need the Zassenhaus lemma, the Schreier refinement theorem, and culminating in the Jordan-Hölder theorem. In the context of algorithms this would be some statement about decomposing a category of algorithms regardless of the order in which the equivalence relations are given. We might also attempt a form of the Krull-Schmidt theorem.

Impossibility results. The most interesting part of Galois theory is that it shows that there are certain contingencies that are impossible or not "solvable". What would the analogue for algorithms be?

Calculate some groups. Its not interesting just knowing that there are automorphism groups. It would be nice to actually give generators and relations for some of these groups. This will also give us a firmer grip for any impossibility results.

Decidability. Where exactly is the boundary between the decidable/syntactic side of programs and the undecidable/semantic side of functions?

Universal Algebra of Algorithms. In this paper we stressed looking at quotients of the structure of all programs. However there are many other aspects of the algorithms that we can look at from the universal algebraic perspective. **Subalgebras.** We considered all primitive recursive programs. But there are subclasses of programs that are of interest. We can for example restrict the number of recursions in our programs and get to subclasses like the Grzegorczyk's hierarchy. How does the subgroup lattice survive with this stratification? Other subclasses of primitive recursive functions such as elementary functions and EXPTIME functions can also be studied. **Superalgebras.** We can also look at larger classes of algorithms. As stated above, we can consider all computable functions by simply adding in a minimization operator. Also, oracle computation can be dealt with by looking at trees of descriptions that in addition to initial functions permit arbitrary functions on their leaves. Again we ask similar questions about the structure of the lattice of automorphisms and the related lattice of intermediate algorithms. **Homomorphisms.** What would correspond to a homomorphism between classes of computable algorithms? Compilers. They input programs and output programs. This opens up a whole new can of worms. What does it mean for a compiler to preserve algorithms? When are two compilers similar? What properties should a compiler preserve? How are the lattices of subgroups and intermediate algorithms preserved under homomorphisms/compilers? There is obviously much work to be done.

References

[1] A. Blass, N. Dershowitz, and Y. Gurevich. "When are two algorithms the same?". Available at http://arxiv.org/PS_cache/arxiv/pdf/0811/0811.0811v1.pdf. Downloaded Feburary 5, 2009.

[2] A. Burroni "Recursivite Graphique (1^e partie): Categories des fonctions recursives primitives formelles". Cahiers De Topologie Et Geometrie Differentielle Categoriques. Vol XXVII-1(1986).

[3] P.M. Cohen *Universal Algebra, 2nd Edition* D. Reidel Publishing Company, 1980.

[4] N.C.A. da Costa "Remarks on Abstract Galois Theory". On the web ftp://logica.cle.unicamp.br/pub/e-prints/vol.5,n.8,2005.pdf.

[5] N.C.A. Da Costa, A.A.M. Rodrigues. "Definability and Invariance", Studia Logica (2007)86 : 1-30.

[6] I. Fleischer. "The abstract Galois theory: a survey." In Proceedings of the international Conference on Cryptology on Algebraic Logic and Universal Algebra in Computer Science (Sydney, Australia). C. H. Bergman, R. D. Maddux, and D. L. Pigozzi, Eds. Springer-Verlag New York, New York, NY, pages 133-137.

[7] Saunders Mac Lane. *Categories for the Working Mathematician,* Second Edition. Springer, 1998.

[8] M.E. Maietti "Joyal's Arithmetic Universe via Type Theory". Electronic notes in Theoretical Computer Science. 69 (2003).

[9] Yu. I. Manin. *A course in Mathematical Logic for Mathematicians– Second Edition.* Springer. October 2009.

[10] Yu. I. Manin. "Renormalization and computation I: motivation and background" on the web http://arxiv4.library.cornell.edu/abs/0904.4921.

[11] Yu. I. Manin. "Renormalization and computation II: Time Cut-off and the Halting Problem" on the web http://arxiv4.library.cornell.edu/abs/0908.3430.

[12] Yu. I. Manin, and N.S. Yanofsky. "Notes on the Recursive Operad". Work in Progress.

[13] Reinhard Pöschel."A general Galois theory for operations and relations and concrete characterization of related algebraic structures". On the web at. http://www.math.tu-dresden.de/~poeschel/poePUBLICATIONSpdf/poeREPORT80.pdf

[14] L. Roman. “Cartesian Categories with Natural Numbers Object.” Journal of Pure and Applied Algebra. 58 (1989), 267-278.

[15] I.G. Rosenberg. “Galois theory for partial algebras”. Springer LNM 1004 (1981).

[16] M.-F. Thibault. “Prerecursive categories”, Journal of Pure and Applied Algebra,. vol. 24,(1982), 7993.

[17] N.S. Yanofsky “Towards a Definition of Algorithms” J Logic Computation exq016 first published online May 30, 2010 doi:10.1093/logcom/exq016.