Towards Automated Derivation in the Theory of Allegories

Joel Glanfield, MSc.

MSc in Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Faculty of Math and Science, Brock University
St. Catharines, Ontario

Abstract

We provide an algorithm that automatically derives many provable theorems in the equational theory of allegories. This was accomplished by noticing properties of an existing decision algorithm that could be extended to provide a derivation in addition to a decision certificate. We also suggest improvements and corrections to previous research in order to motivate further work on a complete derivation mechanism. The results presented here are significant for those interested in relational theories, since we essentially have a subtheory where automatic proof-generation is possible. This is also relevant to program verification since relations are well-suited to describe the behaviour of computer programs. It is likely that extensions of the theory of allegories are also decidable and possibly suitable for further expansions of the algorithm presented here.

Acknowledgements

The author gratefully acknowledges support from the Natural Sciences and Engineering Research Council of Canada.

Appreciation is also expressed to Dr. Michael Winter for many hours of relentless question and answer periods.

Contents

List of Figures

Chapter 1

Introduction

The equational theory of allegories (ALL) is a decidable fragment of the theory of relations (see [7], [10]). It is considered a fragment (or sub-theory) simply because its signature is made up of operations corresponding to the theory of relations. A brief overview of the theory will be given in the next chapter.

Past research has not only demonstrated the decidability of ALL, but has also produced a decision algorithm [10]. Other extensions of ALL have been shown to be decidable with the decision algorithm being implemented in software [3]. The question of interest at this point is as follows:

- using previous work on decidability as a basis, can we automatically generate a derivation for any provable theorem in ALL *in addition to* providing a decision certificate?

Hence, using a decision procedure as a starting point, we are now interested in automatic proof-generation. This is an important aspect of computer science and especially within the field of relational methods since providing proofs of various theorems is often required throughout research, and is potentially tedious. Producing a derivation algorithm for ALL will allow researchers to worry less about simpler proofs and focus attention on more challenging problems. Also, the development of such an algorithm may provide motivation to research other possibilities in terms of automatic proof-generation for other fragments of the theory of relations. As relational methods continue to gain popularity, especially with respect to reasoning about computer programs, one cannot overstate the potential benefit of such work.

We will demonstrate a derivation algorithm that derives many provable theorems in ALL by first providing some context while reviewing a known decision algorithm. Also, we will review a complete framework for reasoning about terms in ALL that was proven in [10]. By combining aspects of the decision procedure and the framework mentioned, we will demonstrate that a derivation algorithm can be obtained. We will show that due to some problems with the previous research we are using as a basis, it is impossible to develop a complete derivation algorithm when the algorithm is based solely on that research. Also, by showing that standardization techniques allow us to relate terms in the same equivalence class, we will demonstrate that rules used may always be applied directly to a subterm of any given term throughout the proof. Thus, our methodology will provide enough detail for potential implementation in new or existing software proof assistants.

As relational reasoning in computer science has become more important over the last few decades, an increased market for modern application has grown. As described in [2], relations are well suited for describing certain types of problems and also contribute to making some proofs easier to supply. The following quotation offers this insight:

> Relations, unlike functions, are essentially nondeterministic and one can employ them to specify nondeterministic problems. For instance, an optimisation problem can be specified in terms of finding an optimal solution among a set of candidates without also having to specify precisely which one should be chosen. Every relation has a well-defined converse, so one can specify problems in terms of converses of other problems [2].

Hence, any attempt to streamline the process of relation reasoning would be of benefit especially to those interested in program semantics and correctness. Our motivation to automatically generate proofs involving the use of operations found in the signature of ALL can thus be viewed as contributing to the simplification of relational reasoning, leaving more complex reasoning to be sorted out.

This is the first work we know of that goes beyond decidability with respect to ALL. It has the potential to provide the basis for providing automatic proof generation for other fragments of relational theory (for those fragments that are decidable). For example, if one could determine how to add the union operation to the signature (something akin to distributive allegories) and prove it's decidability, perhaps our derivation algorithm could

be extended to provide proofs for the extended theory. We also provide motivation for future work geared towards producing a complete derivation algorithm for ALL.

Chapter 2

Background

2.1 A Brief Overview of Allegories

It will be assumed that the reader has some background in category theory.

We take the categorical approach to defining Allegories as outlined in [7], which defines an allegory as follows:

Definition 1. *An ALLEGORY is a category with the signature* $\langle 1, ^\circ, ;, \cap \rangle$ *where* 1 *is a constant,* $^\circ$ *is the unary operation Converse,* ; *is the binary operation Composition and* $\cap$ *is the binary operation Intersection. The category has the following identities (we use juxtaposition to represent the operation ;):*

1. $1R = R = R1$
2. $R(ST) = (RS)T$
3. $R \cap R = R$
4. $R \cap S = S \cap R$
5. $R \cap (S \cap T) = (R \cap S) \cap T$
6. $R^{\circ\circ} = R$
7. $(RS)^\circ = S^\circ R^\circ$
8. $(R \cap S)^\circ = R^\circ \cap S^\circ$

9. $R(S \cap T) \subseteq RS \cap RT$

10. $RS \cap T \subseteq (R \cap TS^\circ)S$

where $\subseteq$ denotes inclusion and is defined by $x \subseteq y \iff x \cap y = x$.

The above definition is also related to categorical approaches introduced in [13] and mentioned in [8, 14, 15].

We should mention here that our work is restricted to the equational theory, i.e., where simple equations can be proven, but not Horn formulae. Along these lines, the following definition provides the set of equations used in [10] to describe the equational theory of allegories and which describe the geometry involved when reasoning about graphs representing terms in ALL. The notation will be that which is used in [10] and which we will use throughout the remainder of this paper (e.g. the operation ; will be omitted).

Definition 2 (Definition 6 in [10])**.** *ALL is the equational theory over the signature $\Sigma = \{;, \cap, ()^\circ, 1\}$ axiomatized by the set of equations $E_{ALL} = E_s \cup E_{op}$ where E_s and E_{op} are defined below.*

The set E_s consists of the following 'static' equations:

1. $x1 = x$

2. $x(yz) = (xy)z$

3. $x \cap (y \cap z) = (x \cap y) \cap z$

4. $x \cap y = y \cap x$

5. $x^{\circ\circ} = x$

6. $(xy)^\circ = y^\circ x^\circ$

7. $(x \cap y)^\circ = x^\circ \cap y^\circ$

The set E_{op} consists of the three 'operational' equations:

8. $x \cap x = x$

9. $x(y \cap z) = x(y \cap z) \cap xy$

10. $xy \cap z = (x \cap zy^\circ)y \cap z$

It has already been shown that this axiomatization is equivalent to that mentioned in Definition 1 (see Lemmas 7 and 10 in [10] for proof).

It should be mentioned here how a concrete allegory is defined.

Definition 3 (Concrete Allegory)**.** *A concrete allegory consists of:*

1. *a class of sets as objects*
2. *a set of binary relations for each pair of objects a and b, i.e., a subset of the powerset of the cartesian product of a and b, which is closed under the set-theoretic operations $\cap, ;, ()^{\circ}$, and 1 (in the case where $a = b$).*

A representable allegory is any allegory which is isomorphic to a concrete allegory.

2.2 Previous Work with Allegories

It has already been mentioned that a decision algorithm exists for the theory. This will be covered in detail in Section 2.3.

Much of the previous work on ALL either makes use of some aspect of graph theory or is related to it in some manner (e.g. circuit design, networks, etc.). Even when exploring the decidability of ALL or related theories it is often desirable to represent terms in the theories as some type of graphical structure.

An example of exploring decidability of an extended theory is some work done by G. Hutton with deciding equations in the theory of allegories with products. He has shown that a simple algorithm exists for deciding equations in this extended theory [11]. Terms in the theory are represented as networks; this is related to how terms are represented in theory of allegories without products, as will be demonstrated shortly. To prove equality, Hutton has shown that two terms are equal if and only if homomorphisms exists between the two networks representing the terms. See [11] for a description of an implementation of the algorithm in the Gofer system[12]. Not unlike our current context, Hutton's work is also limited to the equational theory.

Another practical approach to the study of ALL is found in [4] where allegorical equations are used to reason about circuit design. One specific type of allegory, called a pretabular allegory (which is essentially the same as allegories with products, mentioned in the previous paragraph), is used in this research (for more info on different types of allegories see [2]).

Other work involving allegories can be found in [1], where it is shown that it is desirable to approach aspects of generic programming from a relational standpoint and to use allegories to add further abstraction. Also, other applications involve using allegories to aid in the derivation of programs and more specifically to solve optimization problems [2]. The decision algorithm mentioned in this work has been implemented using the programming language Haskell [5].

2.3 Previous Work on Decidability

One example of major work done on ALL is an NP-time decision algorithm presented in [10]. Our main work in this paper is motivated by this decision algorithm. We will discuss the details of the decision algorithm in this section.

The decision algorithm presented in [10] is based on a graph-theoretical framework where the terms in ALL are represented by graphs. We provide a simple definition of the types of graphs we are concerned with.

Definition 4. *A labelled graph $g = \{V, E, L, l(g), s(g), f(g)\}$ consists of a set of vertices V, a set of edges $E \subseteq V \times V$, a set of labels L, unique start and finish vertices, and is both connected and directed. Edges of a graph are labelled via the function $l : E \rightarrow L$; start and finish vertices are denoted $s(g)$ and $f(g)$ (or using the short forms s and f, respectively).*

For an arbitrary graph g, we denote by $V(g)$ the set of vertices and by $E(g)$ the set of edges of g.

One may consider that a relational variable is represented by a graph consisting of a single directed edge (labelled by the variable) connecting two vertices representing the source and target of the relation. Every term in the theory of allegories has a corresponding graph. Before discussing the details of the algorithm, we will consider the set of graphs PLI_X which correspond to the terms in ALL.

The most basic graphs are 1 and 2_a. The graph 1 consists of a single vertex which is both the start and finish and which has no outgoing or incoming edges, and represents the term 1, i.e., the identity. The graph 2_a is the graph which has one edge representing a single relation, as discussed in the above paragraph. The graph operations are parallel composition $(g_1 \| g_2)$, sequential composition $(g_1 | g_2)$, and converse (g^{-1}), which relate to the theoretical operations intersection, composition and converse respectively (diagrams of

$(1 \cap xx^\circ)$

$$\begin{tikzcd} sf \arrow[r, bend left, "x"] \arrow[r, bend right, "x"'] & \circ \end{tikzcd}$$

$(\mathrm{dom}(x))$

$$\begin{tikzcd} sf \arrow[r, "x"] & \circ \end{tikzcd}$$

Figure 2.1: Graphs representing the terms $1 \cap xx^\circ$ and $\mathrm{dom}(x)$.

these operations are given in Figure 2.2. Some circumstances arise during execution of the algorithm where branching occurs in a graph, hence the corresponding operation $\mathrm{br}(g)$ was added to PLI_X. As a consequence, the dom (domain) operation is added to ALL as an operational extension that allows us to deal with such situations in order to allow a term to be constructed from every graph in PLI_X. The motivation here is that the graph corresponding to $1 \cap xx^\circ$ is in PLI_X, but it's subgraph (found by removing one of the edges) is not a member of the same set (Figure 2.1 gives a visualization of these graphs). Adding dom to the signature of ALL allows us to close PLI_X under subgraphs. The bottom graph in Figure 2.2 is an example of a case requiring the branching operation in order to allow an equivalent term in ALL. Since this graph has equal start and finish vertices, there is no respective term in ALL which naturally occurs. Therefore, the following equation was introduced to define the dom operation:

Equation 1. $\mathrm{dom}(x) = 1 \cap xx^\circ$

Thus, the set PLI_X is the set of all graphs that can be constructed from the most basic graphs mentioned above using the graph operations (also mentioned above). Figure 2.3 shows some example terms with their respective graphs which are constructed from the operations mentioned above.

A relatively simple decision procedure consists of finding homomorphisms between graphs representing allegorical terms where homomorphisms in PLI_X are defined as follows (and is similar to the more general case presented as Definition 59 in [10]:

Definition 5 (Homomorphisms in PLI_X)**.** *Given two graphs $g_1, g_2 \in \mathrm{PLI}_X$, a homomorphism $\varphi : g_1 \to g_2$ in PLI_X is a pair of functions $\varphi_V : V(g_1) \to V(g_2)$ and $\varphi_E : E(g_1) \to E(g_2)$ that*

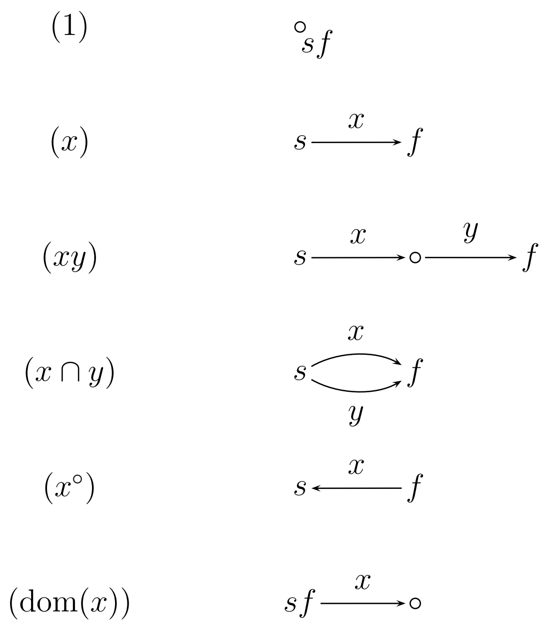


Figure 2.2: Sample graphs representing the basic graphs 1, 2_x, and the operations $g_1|g_2$, $g_1\|g_2$, g^{-1} and $\mathrm{br}(g)$ in PLI_X. The corresponding terms are listed to the left of each graph.

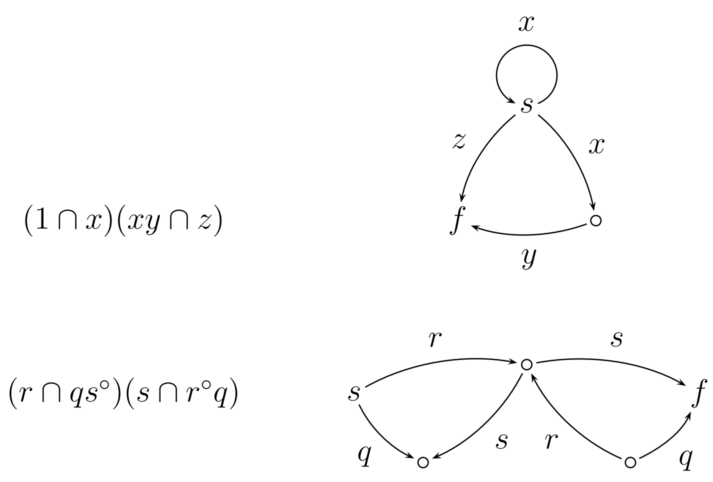


Figure 2.3: Sample terms in ALL with their respective graphs in PLI_X.

1. *Preserve edges and direction, i.e., for all $v, w \in V(g_1)$, if e is an edge in g_1 between v and w, then $\varphi_E(e)$ is an edge in g_2 between $\varphi_V(v)$ and $\varphi_V(w)$.*
2. *Preserve labels, i.e., for all $e \in E(g_1), l(e) = l(\varphi_E(e))$.*
3. *Preserve start and finish vertices, i.e., $\varphi_V(s(g_1)) = s(g_2)$ and $\varphi_V(f(g_1)) = f(g_2)$.*

We normally write a single 'φ' instead of separating it into two functions.

We can say that two terms are equal (in the representable case) if and only if there are are homomorphisms between their respective graphs [7]. This can be weakened to reason about inclusions by looking for a homomorphism in a single direction. However, since we are concerned about non-representable allegories as well as those which are representable, we are interested in those homomorphisms which correspond to equations in ALL. Since we are interested in considering the arrows corresponding to morphisms between graphs, we must consider the categorical context of PLI_X. Before doing so, we repeat the definition of an n-arrow as defined in [10] for convenience.

Definition 6 (n-arrow, see Def 63 in [10])**.** *Let $\varphi : g_1 \to g_2$ be an arrow in D_X. We call $\varphi : g_1 \to g_2$ a n-arrow if and only if*

$$|V(g_1)| \leq |V(\varphi(g_1))| + n$$

In practical terms, a 1-arrow corresponds to the process of identifying at most two vertices in a graph where edge-direction, labels, and the start and finish vertices are all preserved.

We are now ready to present the categorical context of PLI_X. The following definition is also similar to the more general case presented in Lemma 60 of [10].

Definition 7 (The category PLI^1_X)**.** *The category PLI^1_X contains, as objects, the graphs in the set PLI_X where the morphisms are compositions of 1-arrows.*

We now consider arbitrary compositions of 1-arrows, or morphisms in PLI^1_X.

Theorem 1. *$\forall t_1, t_2 \in ALL$, $t_1 \subseteq t_2 \iff$ there is a morphism in PLI^1_X $g_{t_2} \longrightarrow g_{t_1}$. We can prove equality of the same terms if and only if there are two arbitrary morphisms $g_{t_1} \longrightarrow g_{t_2}$ and $g_{t_2} \longrightarrow g_{t_1}$ both in PLI^1_X.*

However, it is the process of proving the above theorem that has motivated our current work. We give credit to the following insight found in [10]:

> It turns out that studying the relation $\rightleftharpoons$ (defined as $g_1 \rightleftharpoons g_2$ if $g_1 \longrightarrow g_2$ and $g_2 \longrightarrow g_1$) in a more general setting, that of category theory, is more fruitful and simple. We show that under very general conditions $\rightleftharpoons$ is an equivalence relation and has normal forms. Moreover, there is a confluent and terminating rewrite system that generates them.

Not only is the categorical setting more fruitful and simple, but it provides a framework whereby we have been able to extract a mechanism to also provide a derivation for many provable equations. We refer the reader to [10] to learn the categorical setting, but give some commentary here to provide context for our current work.

While considering the category PLI^1_X, we will discuss the normalization technique shown in [10]. This will in turn allow us to see why it is important to consider morphisms in PLI^1_X.

If we are to determine whether two different terms are equal, we can perform a series of reductions on the graphs representing the terms until a normal form for each graph is found. These two normal forms are then checked for an isomorphism. Instead of having to consider arbitrary compositions of 1-arrows between two graphs, this normalization process generates the required 1-arrows which can be analyzed in order to yield a derivation by giving a deterministic method of finding a composition of 1-arrows between two graphs.

Figure 2.4 demonstrates the normalization process. As the figure demonstrates, we start with two terms and their respective graphs. These graphs are reduced (a finite number of times) until a normal form is produced. It is then determined whether the normal forms of the two graphs are isomorphic.

If we consider that the process of identifying one set of vertices corresponds to a single step in the normalization process, then there are $n \geq 0$ steps which must occur before a normal form of a graph can be found (the trivial case being that the original graph is already in normal form). To determine whether a reduced graph is indeed in normal form, there are two conditions which must hold. First, no further steps in the normalization process are possible; and second, we must check to see whether there is an

$$\begin{tikzcd}
t_1 \arrow[r, dotted] & g_1 \arrow[r, dotted, no head, "\rightleftharpoons"] \arrow[d, bend left, "e"] & g_2 \arrow[d, bend left, "e"] & t_2 \arrow[l, dotted] \\
& {g_1}' \arrow[u, bend left, "m"] \arrow[d, bend left] & {g_2}' \arrow[u, bend left, "m"] \arrow[d, bend left] & \\
& {g_1}'' \arrow[u, bend left] \arrow[d, dotted] & {g_2}'' \arrow[u, bend left] \arrow[d, dotted] & \\
& \mathrm{nf}(g_1) \arrow[r, dotted, no head, "\cong"] & \mathrm{nf}(g_2) &
\end{tikzcd}$$

Figure 2.4: Normalization process of graphs representing terms in ALL

monomorphism in PLI^1_X from the reduced graph to the original. The normalization process computes an epi-mono factorization of the arrows from g_{t_1} to g_{t_2} and vice versa (if they exist). Therefore, the normal form must always be a subgraph, i.e., there must be a monomorphism from the reduced graph to the original. (We refer the reader to Chapter 4 of [10] for the characterization of epimorphisms and monomorphisms in PLI^1_X.)

We have already defined n-arrows and mentioned their theoretical implication. In practice, however, since we are concerned about monomorphism and epimorphisms (since we attempt to find a normal form which by definition is a subgraph) we consider a 1-arrow to be the process of identifying two vertices in a graph *where one of the vertices has at least the same set of edges as the other*. This allows us to now state exactly how an arbitrary graph would be reduced to its normal form.

The process of finding the normal form of a graph is as follows. A series of steps are performed by generating a composition of 1-arrows on a graph. At each step of the composition a vertex is removed from the graph. The composition ends when there is no longer a possibility of identifying two vertices. Given two graphs g_1 and g_2, if we find a subgraph after a single step has been performed (recall that a step in the normalization process corresponds to identifying exactly two vertices), then we say g_1 has been *reduced to* ${g_1}'$. However, it is possible that the identification of two vertices does not produce the desired subgraph. Hence, we continue to reduce the original graph by identifying more vertices until we find a subgraph (if one can be found); and thus we have a composition of arrows where the overall reduction is an epimorphism and monomorphism (i.e. results in a subgraph). Referring again to Figure 2.4 for a visual, one can see that the entire reduction-process, including isomorphism checks between normal forms, provides an epi-mono factorization of the original morphisms from g_1 to g_2 and from g_2 to g_1 respectively. Figure 2.4 also gives an overall picture of a general epi-mono factorization. Instead of considering arbitrary homomorphisms — which are compositions of 1-arrows — between the original graphs, we reduce the first graph to its normal form and thus have an epimorphism from the graph to its normal form. We then reduce the second graph to its normal form and thus have a monomorphism from the normal form to the second graph. Since the normal forms of both graphs are isomorphic, we now have an epi-mono factorization of the original graph morphisms, and as a consequence we obtain a series, or composition, of 1-arrows from one graph to the other.

Looking for subgraphs at each individual step of the normalization process is what has motivated some of our results, as will be shown hereafter. Recall that a single reduction may be a composition of 1-arrows (i.e. it may take several steps before we find a subgraph).

The obvious benefit of the work mentioned is that it has provided the foundation for relevant future work. For example, decidability for other theoretical fragments of the theory of relations can be researched. It has also provided motivation for this current work. Combining this work with the work in [10], one should be able to produce a complete derivation algorithm. Hence, in any case where an equation is shown to be provable according to the decision algorithm just described, one should also be able to automatically generate a derivation.

This leads us to some of the drawbacks of the previous work mentioned. While describing the graph machinery used to describe the rewrite system described in [10], we have found that cases should have been mentioned but which are missing. Also, a fundamental error in the proof of Lemma 74 of the same work provides a major stumbling block when trying to produce a complete derivation procedure. This error must be corrected before completeness can be shown. Some suggestions towards this end will be presented in this work.

Chapter 3

The Problem of Derivation

3.1 Motivation

We desire to extend previous work on decidability to the point where a derivation of any provable theorem in ALL can be produced. Although it is desirable to show completeness, we will demonstrate throughout this chapter that a complete derivation algorithm that is based on previous work in [10] cannot be produced.

Specifically, our main goal was to extend the work done in [10] on decidability. By the very nature of the decision algorithm itself, it seems clear that a relatively simple derivation algorithm could be extracted. This is for two reasons. First, the reduction mechanism used throughout the decision algorithm results in a number of steps proportional to the size of the input graph. Since graphs representing terms in most interesting equations are relatively small, the number of steps required to find the normal form is also usually relatively small. For example, assuming that the start and finish vertices of a graph are distinct, one can easily show that the graph in normal form will have at least those two vertices. Since every 1-arrow in a composition of 1-arrows identifies exactly two vertices, we would generally find this normalized graph quite quickly, i.e., in at most as many steps as there are vertices in the graph. Secondly, the graph machinery presented in [10] describes a small finite number of cases that describe all possible 1-arrows where the target is a subgraph of the source. Each of these cases is described by a theorem in ALL. Naturally, not every 1-arrow results in a subgraph, but the proof of the decision procedure accounts for this drawback. Hence, since at a first glance it seems that a consequence of the proof of the decision procedure is that one can describe all possible 1-arrows using theorems in ALL, it is natural to assume that a complete derivation procedure could be extracted from the decision algorithm.

The question of whether this can be done has not been answered. The work in [10] stops after decidability. Other work involving implenting the decision procedure or even implementing procedures for other decidabile fragments does not explore derivation (see, for example [5] and [11]). These related works have already been discussed in the previous chapter.

Why are we even intersted in the automated derivation of provable theorems in ALL? Any work towards automating the proving of theorems is always interesting to researchers involved in relational reasoning. One of the past criticisms of the language of relations is the large 'number of operations and laws one has to memorize in order to do proofs effectively' [2]. Any effort towards simplifying the proving process would be beneficial to anyone attempting to prove relational theorems. Furthermore, as relational reasoning becomes more popular when used as a methodology to verify computer programs, it will become more desirable to implement as much automated reasoning as possible. Also, we desire to motivate future work on automated reasoning with other theoretical fragments.

Several provisos should be mentioned here. We are not interested in automated theorem proving from an artificial intelligence paradigm. We make no attempt to pass the well-known Turing test. We do not consider the search-space of all possible proofs, make use of heuristics, or attempt proof reduction. Furthermore, we are not interested in finding the most elegant proofs, nor are we concerned with the length of generated proofs. Our primary goal and focus is to simply provide an algorithm that generates proofs. Future research may be concerned with elegance and length. For the reader interested in these topics in general we refer to [16] and [6].

Since we are not concerned with search-space, we are not confined to using well-known algorithms like brute-force or depth-first search whose running time is less than desirable. As already mentioned, the decision algorithm itself hints towards the possibility of providing a relatively short proof in a relatively small amount of time.

Our approach will be very mechanical. As will be shown, the overall methodology will actually be quite similar to that of a researcher attempting to prove a theorem, i.e., every step of a derivation is the result of the application of some rule that moves us closer to the goal.

3.2 Drawbacks of Previous Work

In this section we demonstrate that no complete derivation mechanism can be extracted from the previous work on decidability found in [10]. As will be shown, this is due to some errors in the proofs of some of the lemmas that would have allowed us to extract the mechanism. We will describe the limitations so as to provide no doubt that they indeed exist. Discussion as to how to overcome these limitations will be reserved for the next chapter.

We will start by assuming that a complete derivation mechanism *can* be extracted from previous work on decidability. By taking this approach we can show how the mechanism should be derived, demonstrate exactly where problems arise, and then give motivation for needed improvements.

Recall the decision procedure outlined in Section 2.3. Since a sequence of 1-arrows leads to a normal form of a given graph, and since 1-arrows can be described using equations in ALL, we can show that by connecting each part of an epi-mono factorization that we should be able to provide a complete derivation of a theorem. Before doing so, we will recap some previous work in order to provide context. We will discuss some details regarding arrows between graphs in PLI^1_X that will provide the basis for our discussion.

3.2.1 A Rewrite System for ALL

In [10], a 'rewrite system which gives a complete computational procedure for doing arithmetic in the theory of allegories' is presented. The proof of the completeness of this system is the combination of several lemmas which we will outline here (the proofs for each of which can also be found in [10]). *It is the proofs of these lemmas that have motivated this work*, since the mechanics of the proofs provide hints of how to extract a derivation procedure.

Lemma 2 (Lemma 71 in [10]). *Let $\approx$ denote the congruence in PLI^1_X generated by the equations in E_{op}, and let r,t be terms in $T_\Sigma(X)$ (the term algebra of the signature Σ over X). Then the following statements hold:*

1. *$r = t$ in ALL if and only if $g_r \approx g_t$ in PLI^1_X*
2. *$r = t$ in $E_s \cup \{(71),(72),(73),(74)\}$ if and only if $g_r \approx g_t$ in PLI^1_X,*

where equations 71-74 are outlined in [10] as:

$$a \cap a = a, a \in X \cup \{1\} \tag{71}$$

$$xy \cap xy = (x \cap x)(y \cap y) \tag{72}$$

$$(x \cap x)(y \cap y \cap z) = x(y \cap z) \cap xy \tag{73}$$

$$x(y \cap y) \cap z \cap z = (x \cap zy^\circ)y \cap z \tag{74}$$

Lemma 3 (Lemma 72 in [10])**.** *Let h be a graph in PLI_X. If $\varphi : h \to h$ is an arrow which identifies exactly one pair of vertices, then $h \approx \varphi(h)$.*

Claim 4 ('Lemma' 73 in [10])**.** *Let g,h be a graphs in PLI_X. If there is a 0-arrow $\varphi : h \to g$, then $g \approx g\|h$ in PLI^1_X.*

Claim 5 ('Lemma' 74 in [10])**.** *Let g,h be graphs in PLI_X. The following statements hold:*

1. *If there is a 1-arrow $h \to g$ in PLI^1_X, then $g \approx g\|h$ in PLI^1_X.*
2. *If there is an arrow $h \to g$ in PLI^1_X, then $g \approx g\|h$ in PLI^1_X.*

3.2.2 Extracting the Derivation Procedure

In order to extract a derivation procedure, we start by considering individual reduction steps throughout the normalization process. The individual steps of the normalization process were discussed in Section 2.3, but here we are concerned with those compositions of 1-arrows where the target graph is a subgraph of the original. Before proceeding, we formally define the notion of an n-reduction to simplify our discussion.

Definition 8 (n-reduction)**.** *Let $g, g' \in \mathrm{PLI}_X$. g' is an n-reduct of g if and only if the following hold:*

1. *there is an embedding $e : g' \to g$*
2. *there is a composition of 1-arrows $f : g \to g'$, i.e., $f_n f_{n-1} ... f_1$*
3. *there is no graph in $g_1, ..., g_{n-1}$ such that there is an embedding into g.*

g, g' together with the functions f and e form what we call an n-reduction.

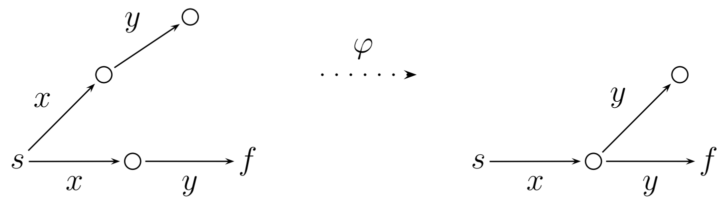


Figure 3.1: A 1-arrow that is difficult to characterize.

Lemma 3 allows us to reason about the trivial case, 1-reductions, using equations in ALL (see Definition 2). Every time we reduce a graph (i.e. identify one pair of vertices), we consider the minimum algebraic subgraph induced by the reduction. The minimum algebraic subgraph is the smallest subgraph that contains the identified vertices v and v' and the edges incident to v and v' (for a formal definition we refer the reader to Chapter 5 of [10]). The general case is described in Figure 18 of [10]. The specific cases (twelve in total, modulo symmetries) are then described. We offer some improvements to these cases in Appendix A. The proof suggests that the cases outlined are sufficient to reason about all possible algebraic subgraphs resulting from arbitrary 1-reductions.

The problem now arises when attempting to reasoning about n-reductions where $n > 1$. The individual reduction steps of such an n-reduction correspond to those 1-arrows where the target graph is not a subgraph of the source graph. Consider the 1-arrow demonstrated in Figure 3.1 as an example. The graph on the left-hand side of the arrow corresponds to the term $\mathrm{dom}(xy)xy$, while the graph on the right-hand side corresponds to the term $x(\mathrm{dom}(y))y$. The minimal algebraic subgraph extracted from this reduction corresponds to the equation $\mathrm{dom}(x)x = x$ (see case 2 of Lemma 72 in [10]). There is no obvious application of this equation to the present scenario regardless of the fact that the overall terms are provably equal (use the decision procedure described in Section 2.3 to prove this fact). The suggested solution to overcoming this problem lies in the mechanics of the proofs of Lemmas 3-5. We will describe these details and then demonstrate how they would help overcome this problem and also lead to a more general derivation procedure were it not for some errors in the proofs.

Referring back to Part 1 of Lemma 5, we can see that whenever we have a 1-arrow, φ, between two graphs h and g we can show that $g \approx g\|h$ in PLI^1_X. From the proof of this lemma we see that a new graph h^* must be constructed to prove this congruence. h^* is constructed by taking the image of h, $\varphi(h)$, and adding a single vertex and any edges required such that an embedding exists from h to h^*. We then extend φ to $\varphi^* : h^* \to h^*$ in the obvious way since $h, \varphi(h)$ are both subgraphs of h^*. We can then use Lemma 3 to conclude $g \approx g\|\varphi(h) \approx g\|\varphi(h)\|h \approx g\|h$. Figure 3.2 gives an example where a graph h^* is generated based on the source and target of a 1-arrow. The boldly-outlined vertex in the graph h^* is the vertex inserted to generate the embedding from h to h^*. From Part 2 of Lemma 5 we can see that in the case where we have a composition of 1-arrows we can still show that $g \approx g\|h$ still holds. Although we are really interested in finding a proof for the terms relating to $h \approx \varphi(h)$, we will demonstrate that deriving $g \approx g\|h$ is just as beneficial when combined with aspects of Lemmas 2 and 4 to prove the overall equation in question. Before doing so, we must show that a derivation of the terms relating to $g \approx g\|h$ can always be produced.

Theorem 6. *Consider the relation $g \approx g\|h$ in PLI^1_X. The equation $t_1 = t_2$, where t_1 is the term corresponding to the graph g and t_2 is the term corresponding to the graph $g\|h$, can always be derived.*

Proof. A result of Part 1 of Lemma 2. □

In practise, we will derive the equation corresponding to $g \approx g\|h$ using aspects of both Lemmas 2 and 4. We will demonstrate how this is done in the next section.

Lemma 4 states that whenever a 0-arrow exists (between two graphs h and g) we can show that $g \approx g\|h$ in PLI_X. In practice, a 0-arrow will occur whenever we remove an edge without identifying two vertices. This implies that the result is an embedding from g to h. The key insight here as that if an equation is decidable, then there are embeddings from the normal forms of each side of the equation to both terms in the original equation. We will now prove this fact.

Theorem 7. *Let $t_1, t_2 \in ALL$ and let $g_{t_1}, g_{t_2} \in \mathrm{PLI}_X$ correspond to the graphs of the terms t_1 and t_2. Now, consider the equation $t_1 = t_2$, and let $\mathrm{nf}(g_{t_1})$ and $\mathrm{nf}(g_{t_2})$ be the normal forms of g_{t_1} and g_{t_2} respectively. If the equation $t_1 = t_2$ is provable according to the decision algorithm described in Section 2.3, then the following embeddings exist:*

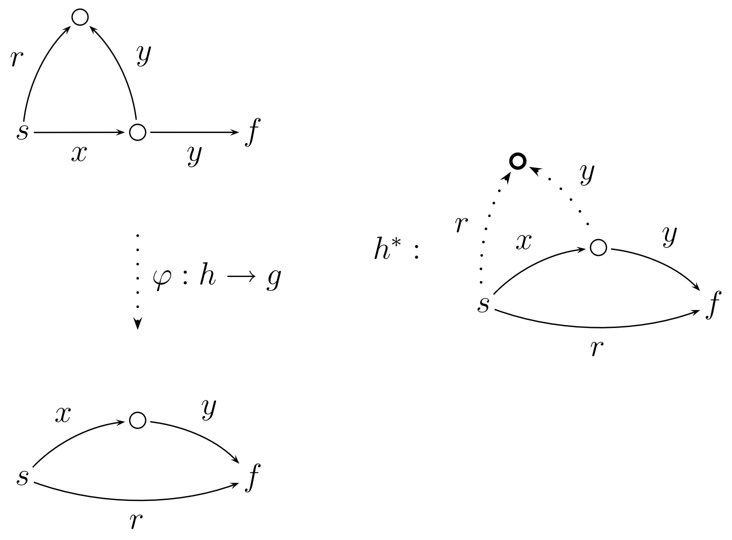


Figure 3.2: An example of generating the graph h^*.

1. $\varphi_1 : \mathrm{nf}(g_{t_1}) \to g_{t_1}$
2. $\varphi_2 : \mathrm{nf}(g_{t_2}) \to g_{t_2}$
3. $\varphi_3 : \mathrm{nf}(g_{t_1}) \to g_{t_2}$
4. $\varphi_4 : \mathrm{nf}(g_{t_2}) \to g_{t_1}$

Proof. To prove Part 1, we know that the equation $t_1 = t_2$ is provable, therefore the embedding φ_1 exists by virtue of the decision procedure. Part 2 is similar to Part 1. For Part 3, we know that the equation $t_1 = t_2$ is decidable, therefore an isomorphism exists between $\mathrm{nf}(g_{t_1})$ and $\mathrm{nf}(g_{t_2})$. Hence, there is an embedding from both $\mathrm{nf}(g_{t_1})$ and $\mathrm{nf}(g_{t_2})$ to g_{t_2}. Part 4 is similar to Part 3. See Figure 3.3 for embeddings that occur during the decision procedure. ☐

The proof of Lemma 4 uses induction to show that a simple derivation of the terms relating to $g \approx g \| h$ can be accomplished using the equation $a \cap a = a$.

We can now proceed to combine aspects of the proofs of Lemmas 3, 4 and 5 to provide a general mechanism for extracting a derivation from the

$$\begin{tikzcd}
t_1 \arrow[r, dotted] & g_{t_1} \arrow[d, dotted, "\varphi_1"'] & g_{t_2} \arrow[d, dotted, "\varphi_2"] & t_2 \arrow[l, dotted] \\
\mathrm{nf}(t_1) & \mathrm{nf}(g_{t_1}) \arrow[l, dotted] \arrow[u, bend left] \arrow[ur] \arrow[r, dotted, dash, "\cong"] & \mathrm{nf}(g_{t_2}) \arrow[r, dotted] \arrow[ul] \arrow[u, bend right] & \mathrm{nf}(t_2)
\end{tikzcd}$$

Figure 3.3: Some embeddings that occur during the decision procedure (i.e. the solid arrows represent the embeddings, while φ_1,φ_2 represent compositions of 1-arrows).

decision procedure. By considering the morphisms that occur during the decision process and by relating them to the lemmas just mentioned, we should be able to extract a method of combining different proofs to provide an overall proof of an equation (e.g. like one would combine several lemmas to prove an overal theorem).

Consider the morphisms that are shown in Figure 3.4. Due to the morphisms that occur during the decision procedure, we can apply Lemmas 4 and 5 and thus generate a derivation of the equations mentioned in the next Theorem:

Theorem 8. *Let $t_1, t_2 \in ALL$, and consider the graphs g_{t_1}, g_{t_2}, $\mathrm{nf}(g_{t_1})$ and $\mathrm{nf}(g_{t_1}) \in \mathrm{PLI}_X$. If $t_1 = t_2$ is provable according to the decision algorithm described in Section 2.3, then the following equations hold:*

1. $g_{t_1} \approx g_{t_1} \| g_{t_2}$
2. $g_{t_2} \approx g_{t_2} \| g_{t_1}$

Proof. Part 1:

$$\begin{aligned}
g_{t_1} &\approx g_{t_1} \| \mathrm{nf}(g_{t_2}) && \text{(Lemma 4, emb. } \mathrm{nf}(g_{t_2}) \to g_{t_1}\text{)} \\
&\approx g_{t_1} \| \mathrm{nf}(g_{t_2}) \| g_{t_2} && \text{(Lemma 5, arrow } g_{t_2} \to \mathrm{nf}(g_{t_2})\text{)} \\
&\approx g_{t_1} \| g_{t_2} && \text{(Lemma 4, emb. } \mathrm{nf}(g_{t_2}) \to g_{t_1}\text{)}
\end{aligned}$$

$$\begin{tikzcd}
t_1 \arrow[r, dotted] & g_{t_1} \arrow[d, bend right=40, "e"'] \arrow[d, dotted, "\varphi_1"] & & g_{t_2} \arrow[d, bend left=40, "e"] \arrow[d, dotted, "\varphi_2"'] & t_2 \arrow[l, dotted] \\
\mathrm{nf}(t_1) & \mathrm{nf}(g_{t_1}) \arrow[l, dotted] \arrow[urr, "m"'] \arrow[rr, dotted, dash, "\cong"] & & \mathrm{nf}(g_{t_2}) \arrow[ull, "m"] \arrow[r, dotted] & \mathrm{nf}(t_2)
\end{tikzcd}$$

Figure 3.4: Some epimorphisms (e) and monomorphisms (m) that occur during the decision procedure that allow us to extract a derivation mechanism.

Part 2:

$$\begin{aligned}
g_{t_2} &\approx g_{t_2} \| \mathrm{nf}(g_{t_1}) && \text{(Lemma 4, emb. } \mathrm{nf}(g_{t_1}) \to g_{t_2}\text{)} \\
&\approx g_{t_2} \| \mathrm{nf}(g_{t_1}) \| g_{t_1} && \text{(Lemma 5, arrow } g_{t_1} \to \mathrm{nf}(g_{t_1})\text{)} \\
&\approx g_{t_2} \| g_{t_1} && \text{(Lemma 4, emb. } \mathrm{nf}(g_{t_1}) \to g_{t_2}\text{)}
\end{aligned}$$

□

Since we already know that the individual steps of the proof of Theorem 8 correspond to Lemmas 3, 4, and 5, we know that a derivation of the term-equations (corresponding to each step of the proof) can always be provided. We are left to show how this can be done. Doing this will should allow us to provide a general mechanism that will generate a derivation of any equation $t_1 = t_2 \in ALL$ where $t_1 = t_2$ is decidable (and which we attempt to demonstrate in the next section). We say that a general mechanism *should* be demonstrable because of some errors we have found in the proofs of Lemmas 3 and 4. We will proceed as if were going to produce the general algorithm and then discuss the errors in the proofs that stop us from showing completeness.

3.2.3 The Pseudo-Algorithm

We give a general outline of the derivation algorithm that we attempt to produce, followed by specific details as to how each equation of the outline can always be derived. When we speak of 'deriving' a graph-equation, we mean that we derive the corresponding term-equation.

The simplest way to provide a derivation would be to use the results of Lemma 2 by applying the appropriate case to each graph-reduction. If each 1-arrow during the normalization process always corresponded to an n-reduction where $n = 1$, we would simply apply the appropriate rule to derive the term corresponding to the reduced graph. However, we attempt to provide a more general, but more involved derivation mechanism that accounts for the cases where n-reductions are of size $n > 1$. This is why we are interested in the results of Theorem 8, where instead of deriving $g_{t_1} \approx g_{t_2}$ directly, we derive $g_{t_1} \approx g_{t_1} \| g_{t_2} \approx g_{t_2}$.

In order to derive $g_{t_1} \approx g_{t_1} \| g_{t_2}$ we start by deriving two smaller equations. We follow the same process when deriving $g_{t_2} \approx g_{t_2} \| g_{t_1}$. The process of deriving $g_{t_1} \approx g_{t_1} \| g_{t_2}$ would occur in three steps:

1. Derive: $\mathrm{nf}(g_{t_1}) \approx \mathrm{nf}(g_{t_1}) \| g_{t_1}$
2. Derive: $g_{t_2} \approx g_{t_2} \| \mathrm{nf}(g_{t_1})$
3. Use (2) followed by (1) to derive: $g_{t_2} \approx g_{t_2} \| g_{t_1}$

We would then derive $g_{t_2} \approx g_{t_2} \| g_{t_1}$ in a similar fashion:

4. Derive: $\mathrm{nf}(g_{t_2}) \approx \mathrm{nf}(g_{t_2}) \| g_{t_2}$
5. Derive: $g_{t_1} \approx g_{t_1} \| \mathrm{nf}(g_{t_2})$
6. Use (5) followed by (4) to derive: $g_{t_1} \approx g_{t_1} \| g_{t_2}$

The last step would be to combine the derivations of $g_{t_2} \approx g_{t_2} \| g_{t_1}$ and $g_{t_1} \approx g_{t_1} \| g_{t_2}$ to show a complete derivation of $g_{t_1} \approx g_{t_2}$, i.e.,

8. Combine the derivations of (3) and (7) to produce the complete derivation of: $g_{t_1} \approx g_{t_2}$

Part 1 follows from the fact that the decision procedure results in a composition of 1-arrows from g_{t_1} to $\mathrm{nf}(g_{t_1})$. Part 2 follows from the embedding that exists from $\mathrm{nf}(g_{t_2})$ to g_{t_2}. Both parts 1 and 2 were used in the proof of Theorem 8 to show that the equations corresponding to $g_{t_1} \approx g_{t_1} \| g_{t_2}$ and $g_{t_2} \approx g_{t_2} \| g_{t_1}$ can always be derived. The same follows for parts 4 through

6. The bottom line is that we should be able to show that following steps 1 through 7 comprises an algorithm that derives any provable equations in ALL.

At this point we are left to show exactly how these equations should be derived using equations in ALL. We will proceed by attempting to prove that each equation can be derived using the mechanics of the proofs of Lemmas 2 through 5. We will then demonstrate how errors in the proofs of these lemmas stop us from showing completeness.

Claim 9. *Let $t_1, t_2 \in$ ALL and $E_\rightarrow$ be the set of equations representing the 1-arrows mentioned in Lemma 2. If the terms of the equation $t_1 = t_2$ correspond to the graphs in the equation $\mathrm{nf}(g_{t_1}) \approx \mathrm{nf}(g_{t_1}) \| g_{t_1}$, then $E_\rightarrow \cup a \cap a = a \vdash t_1 = t_2$.*

Proof. We attempt to prove this claim in three parts:

Part 1 corresponds to the trivial case where $\mathrm{nf}(g_{t_1}) = g_{t_1}$ (i.e. no vertices were identified). In this case the derivation is trivial.

Part 2 corresponds to the case where the normal form of g_{t_1} is found in one step (i.e. only one set of vertices is identified). In this case we would generate the graph h^* as described in Section 3.2.2. The derivation would then be extracted from the following:

$$\begin{aligned}
\mathrm{nf}(g_{t_1}) &\approx h^* && \text{(Lemma 3)}\\
&\approx h^* \| g_{t_1} && \text{(Lemma 4)}\\
&\approx \mathrm{nf}(g_{t_1}) \| g_{t_1} && \text{(Lemma 3)}
\end{aligned}$$

The term equation corresponding to the first line of the derivation would be derived using an equation from the set $E_\rightarrow$, since there is a 1-arrow *and* an embedding from $\mathrm{nf}(g_{t_1})$ to h^* (by virtue of how h^* is constructed). The second line is derived by applying the rule $a \cap a = a$ to the subterm (of the previous line) corresponding to the graph g_{t_1} (which is a subgraph of h^*). The rule $a \cap a = a$ is applied because of the proof of Lemma 4 since there is an embedding from g_{t_1} to h^*. The third line is then derived in a similar manner as the first line.

Part 3 of the proof corresponds to the case where the normal form of g_{t_1} is found in more than one step. Again, we rely on the construction of the graph h^* for each step of the normalization process in order to derive the equation corresponding to the epimorphism from the source of the reduction to the target. We then work our way back from the normal form to the original graph using the same procedure in Part 2 of this proof with a slight modification (assume there are n-intermediate reductions):

$$\begin{aligned}
\mathrm{nf}(g_{t_1}) &\approx \mathrm{nf}(g_{t_1})\|g_{t_1^n} && \text{(see proof of Part 2)}\\
&\approx \mathrm{nf}(g_{t_1})\|(g_{t_1^n}\|g_{t_1^{n-1}}) && (g_{t_1^n} \approx (g_{t_1^n}\|g_{t_1^{n-1}}))\\
&\approx \mathrm{nf}(g_{t_1})\|g_{t_1^n}\|g_{t_1^{n-1}}\|...\|(g_{t_1^2}\|g_{t_1^1}) &&\\
&\approx (\mathrm{nf}(g_{t_1})\|g_{t_1^n})\|g_{t_1^{n-1}}\|...\|g_{t_1^2}\|g_{t_1^1} && \text{(associativity of } \|)\\
&\approx \mathrm{nf}(g_{t_1})\|g_{t_1^{n-1}}\|...\|g_{t_1^2}\|g_{t_1^1} && \text{(see proof of Part 2)}\\
&\approx (\mathrm{nf}(g_{t_1})\|g_{t_1^{n-1}})\|...\|g_{t_1^2}\|g_{t_1^1} && \text{(associativity of } \|)\\
&\approx \mathrm{nf}(g_{t_1})\|...\|g_{t_1^2}\|g_{t_1^1} && \text{(Part 2 of Lemma 5)}\\
&\approx ... &&\\
&\approx \mathrm{nf}(g_{t_1})\|g_{t_1} && \text{(see proof of Part 2)}
\end{aligned}$$

The only difference here from the proof of Part 2 is that we have more intermediate steps corresponding to the proofs of equations of the type $g \approx g\|h$ for neighboring graphs g and h of the overall normalization of g_{t_1}. $\square$

We can see from the proof of this claim that, assuming the proofs of Lemmas 2 - 5 are correct, we can always provide a derivation for the terms corresponding to $\mathrm{nf}(g_{t_1}) \approx \mathrm{nf}(g_{t_1})\|g_{t_1}$. A problem arises when we consider the proof of Lemma 5. The mechanics of generating the graph h^* should allow us to generalize those situations where we have n-reductions, which would be a major step towards producing a complete derivation mechanism. However, we have constructed some scenarios where the graph h^* is not even in PLI_X, which implies that there are situations where the graph h^* does not even have a corresponding term in ALL. Consider the scenario demonstrated in Figure 3.5. The only way one could generate the graph h^* such that there is an embedding from h to h^* is to add a single vertex that results in a graph that has the shape of a diamond. As explained in [10], this graph is outside of the theory and does not correspond to any term in ALL. Since it is possible to generate a graph h^* that is outside of the theory, the h^*-generation mechanism breaks down at this point.

It is important to note here that just because the proof mechanism for Lemma 5 is incorrect, this does not mean the Lemma itself is not provable. In

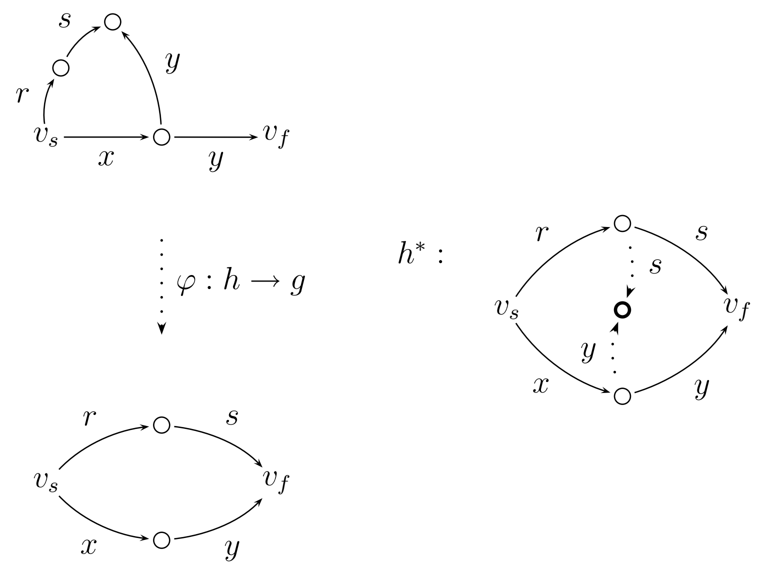


Figure 3.5: A theoretical example of generating the graph h^* where h^* is not in PLI_X.

fact, we believe it is provable. The bottom line here is that we simply cannot use the proof mechanism to help us produce a derivation procedure. We now present a more formal proof of the incorrectness of the proof of Lemma 5.

Lemma 10. *There is no graph h^* for the 1-arrow $\varphi : h \to g$ shown in Figure 3.5 which allows us to show $g \approx g \| h$.*

Proof. We use a contradiction to prove this argument.

Assume that there is a graph h^* which can be generated from the 1-arrow described in Figure 3.5. Next, consider the 1-arrow $\varphi : h \to g$ where h is the graph of the term $(rsy^\circ \cap x)y \in \text{ALL}$ and g is the graph of the term $rs \cap xy \in \text{ALL}$. Since φ is a 1-arrow, according to the proof of Lemma 5 we can generate a new graph h^* where there are embeddings from both h and g to h^*. However, the graph h^* in this case is a diamond and is therefore outside of PLI_X. Hence, there is no graph $h^* \in \text{PLI}_X$ that can be generated. □

It should be noted here that the example demonstrated in Figure 3.5 is purely theoretical, i.e., it would never arise in an equational proof since the target graph of the 1-arrow is neither a subgraph of the source, nor can it be reduced further. However, there are examples which do arise in equational proofs. For instance, consider the 1-arrows of the reduction of the left term of the equation $xyz \cap xyz = xyz$. It can easily be seen that there is no proper graph h^* which can be generated from the first 1-arrow of the composition, since the procedure for generating the graph h^* results in a diamond. We refer the reader to Figure 3.6 for a visualization of this scenario.

We refer the reader to [10] for details as to why any diamond-shaped graph is outside of PLI_X. At this point we cannot continue to demonstrate that our suggested algorithm works due to this major shortcoming. However, we will also mention another problem at this point.

In the proof of Lemma 4, we should be able to extract a derivation mechanism for situations where 0-arrows occur. For example, if we were to succesfully use the h^* methodology when reasoning about 1-arrows, we would still need to reason about the embeddings that occur from the source graph (of the 1-arrow) to h^* (see Part 2 of the attempted proof of Claim 9). The problem with the proof of this lemma is that the only base case of the induction has to do with the graph $h = 2_a$. The first inductive step then talks about parallel graphs, but not all of these can be described using the base case. Consider the situation described in Figure 3.7. Here we have an em-

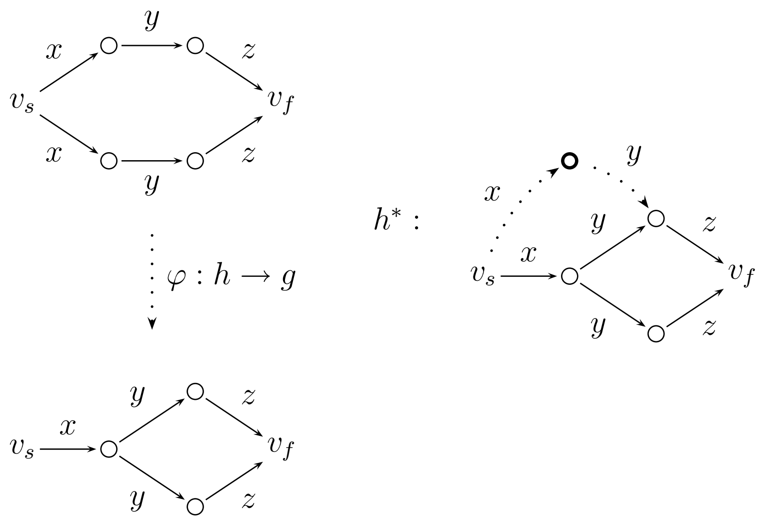


Figure 3.6: A concrete example of generating the graph h^* where h^* is not in PLI_X.

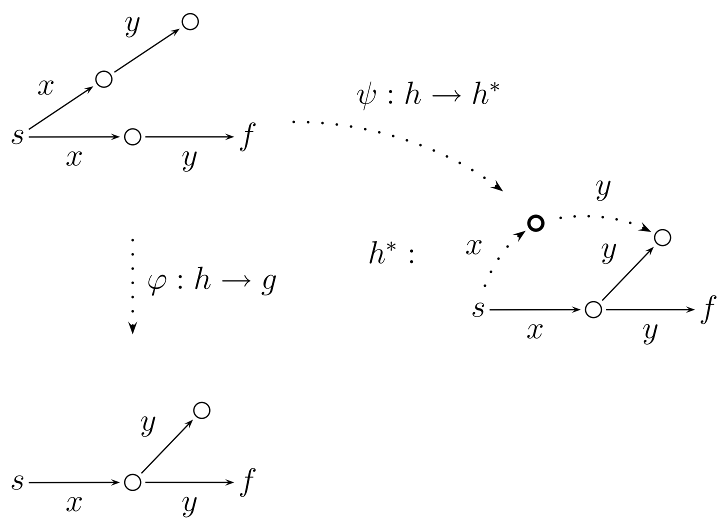


Figure 3.7: An example where the embedding $\psi : h \to h^*$ is not described by Lemma 4.

bedding from the graph g to the graph h^*, but there is no reasonable method to describe this embedding given the proof of Lemma 4.

Chapter 4

The Solution

In this chapter we address the issues discussed in the previous chapter, namely, how to overcome the problem of not being able to use the h^*-generation technique to reason about arbitrary n-reductions. We then discuss a standardization technique which addresses specific implementation-related issues.

4.1 Addressing h^*-generation

This section will focus on overcoming the limitation of the h^*-generation approach. We will start by showing that attempting to *fix* the notion of h^*-generation does not help us with our problem of providing automated derivation. We then suggest an alternate approach, disregarding h^*-generation altogether.

4.1.1 h_n^*-generators

We start by defining the notion of an h_n^*-generators to provide motivation for our current discussion.

Definition 9 (h_n^*-generator)**.** *Consider the graphs $g, h \in \mathrm{PLI}_X$ and the 1-arrow $\varphi : h \to g$. An h_n^*-generator is a function which generates a graph h^* by adding n vertices (with the required edges) to g such that there is an embedding from h to h^*.*

The definition of an h_n^*-generator is motivated from the fact that although we may not necessarily be able to construct the graph h^* by adding a single

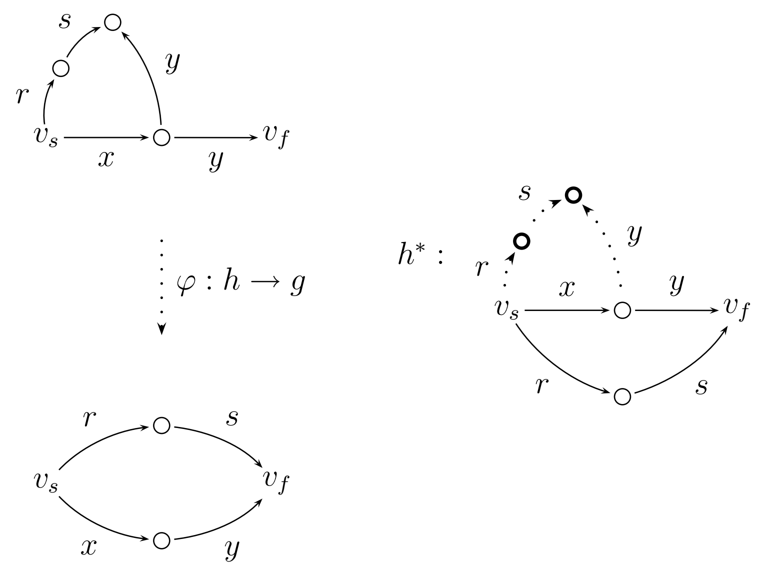


Figure 4.1: An example where two vertices are required to construct h^*).

vertex (recall the example in Figure 3.5), we may be able to construct it by adding n vertices. For the example in Figure 3.5, the graph h^* could be constructed by adding two vertices instead of just one. This way we would end up with a graph in PLI_X (see Figure 4.1 for an example of an h_2^*-generation). The bottom line here is that we are attempting to construct h^* in a manner such that it will always be in PLI_X. However, the obvious problem is that adding more than one vertex to construct h^* properly causes us to need to reason about n-arrows in general, which takes us outside of PLI_X^1.

4.1.2 Compositions of 1-arrows

A seemingly easier approach to addressing the aforementioned problems would be to consider compositions of 1-arrows in general. By looking at an overall n-reduction, i.e., ignoring the intermediate reduction steps (recall Definition 8) we can attempt to classify it according to one of the cases described in Lemma 3 (see Appendix A for a description of the cases). For the trivial case, where $n = 1$, we use Lemma 3 to characterize the situation as previously discussed.

In the cases where we consider n-reductions where $n > 1$, we must consider complete paths of a graph as representing single variables in a term. In [10], there is a hint that considering arbitrary 1-arrows alone can be problematic. On page 83, there is the single line 'graphs corresponding to equations with non-trivial 1-arrows are shown in Figure 31'. Figure 31 does indeed describe scenarios where some 1-arrows do not result in a subgraph. We assume that the h^*-generation mechanism was considered to overcome this problem, but as we've already demonstrated, it is not sufficient. However, if we consider the variables (or single edges) in Figure 31 as complete paths, then we find that we are closer to a solution. Figure 4.2 demonstrates an example of a composition of 1-arrows which can be reasoned about using one of the cases in Lemma 3 if we replace similar paths with single edges. Recall in Figure 3.7 an example where a 1-arrow is difficult to characterize. If we ignore the first 1-arrow but consider the composition of two 1-arrows (i.e. an n-reduction where $n = 2$) and then replace the identical paths with single edges, we can see that this creates a reduction that does indeed correspond to a case described in Lemma 3. We simply replace the paths corresponding to the term xy with a single edge corresponding to the term x, which then allows us to apply the rule $\mathrm{dom}(x) = x$.

While replacing paths with single variables, problems arise when one path is a subpath of another. Since the subgraphs represented by these paths are not isomorphic, we cannot simply replace them with a single edge even though their respective terms in ALL are provably equal. We will consider a scenario almost identical to that which is demonstrated in Figure 4.2.

If we consider the scenario demonstrated in Figure 4.3, it is clear that we cannot simply replace the paths xy and $x(y \cap z)$ with a single edge. However, the important thing to note here is that there is still an embedding from the larger path xy to the smaller path $x(y \cap z)$ (where *larger* implies that the respective term in ALL is larger); this implies that we can show an appropriate derivation of $x(y \cap z) = x(y \cap z) \cap xy$ (see Lemma 4). We would find the embedding we are looking for by considering the pairs of vertices identified throughout the normalization process.

The proof of the reduction in Figure 4.3 would now look as follows:

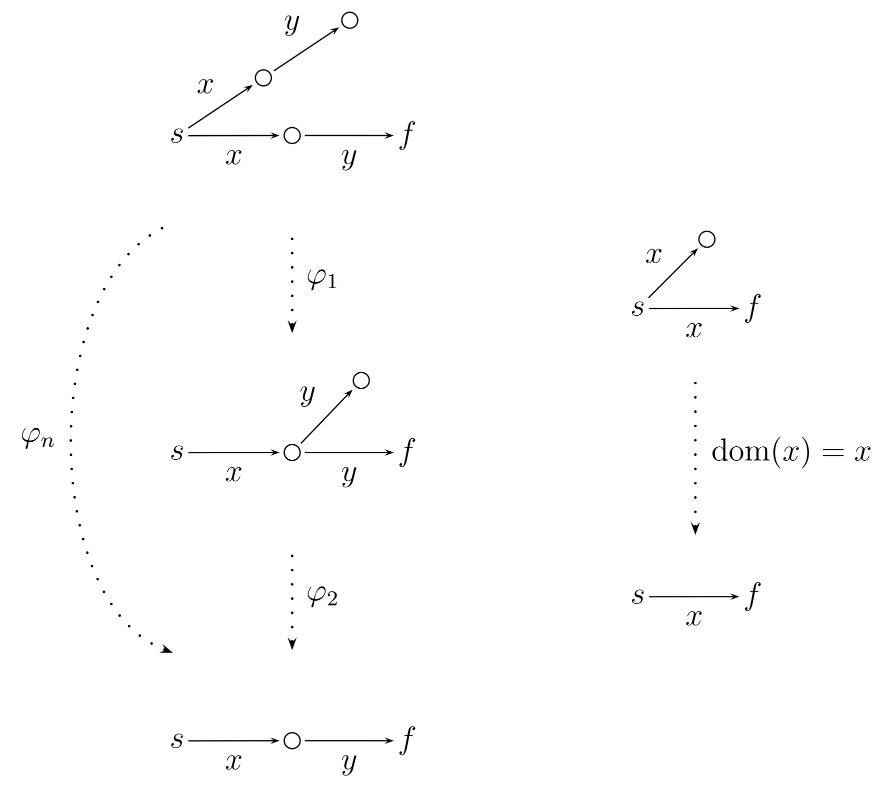


Figure 4.2: An example where a composition of two 1-arrows, φ_n, is described by the rule $\mathrm{dom}(x) = x$ when we replace the paths xy with the single edge x.

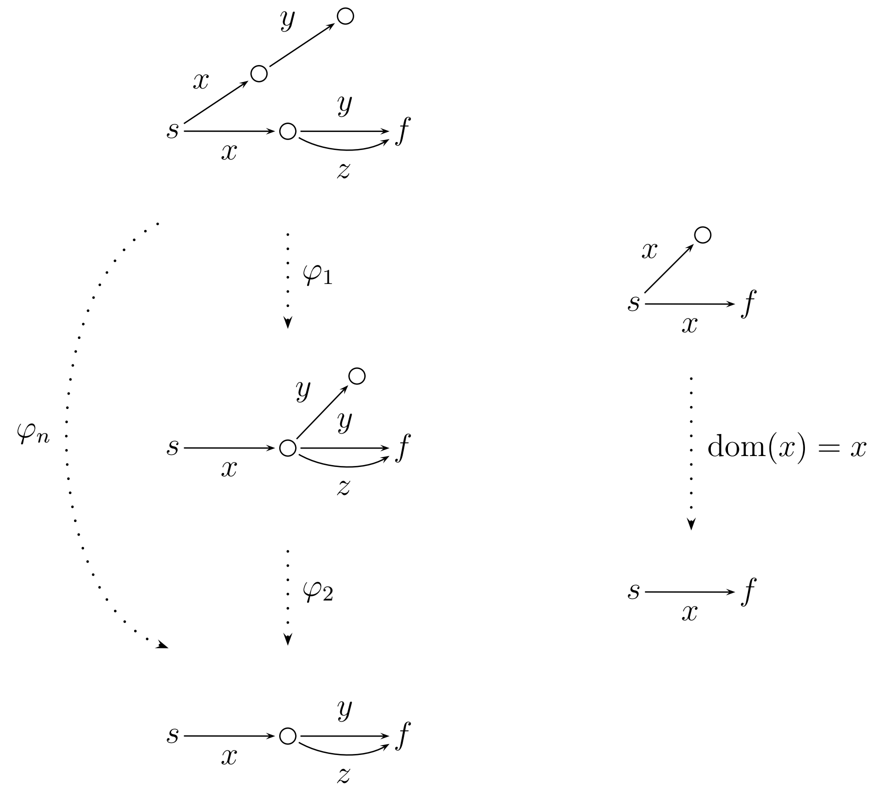


Figure 4.3: An example where the overall n-reduction φ_n cannot be described by the rule $\mathrm{dom}(x) = x$ when replacing entire paths with single edges.

$$\begin{aligned}
\mathrm{dom}(xy)x(y \cap z) &=^{(\mathrm{L4})} \mathrm{dom}(xy)(xy \cap x(y \cap z)) \\
&=^{(\mathrm{T12})} \mathrm{dom}(xy)xy \cap x(y \cap z) \\
&=^{(\mathrm{L3,arrow})} xy \cap x(y \cap z) \\
&=^{(\mathrm{L4})} x(y \cap z)
\end{aligned}$$

It is our claim that this more general procedure would then allow us to generate a derivation in any situation, but this claim remains to be proven.

Claim 11. *Combining the arithmetic demonstrated in Lemma 3 with our approach of considering arbitrary n-reductions, where $n > 1$, and where we must replace paths where one is a subpath of another (described in this section) gives a complete derivation mechanism.*

The motivation behind this claim is twofold, as follows.

The first part of the motivation is a result of considering a composition of 1-arrows that is just a single 1-arrow, $\varphi : h \to g$, where there is an surjection from h to g and an embedding from g to h. This scenario is covered by Lemma 3, since during the proof of this lemma there is the statement 'there must be a vertex $v \in V(h)$ such that $\varphi(h) = \varphi(h - v) \cong (h - v)$'. Since the image of h, $\varphi(h)$, must be isomorphic to the graph h less the vertex v, there must be an embedding from $\varphi(h)$ to h. The cases mentioned in Lemma 3 (which are enumerated in Appendix A for the reader's convenience) are sufficient to allow a derivation of any such trivial composition.

The second part is where we have a composition of at least two 1-arrows, where there is an embedding from the target graph of the composition to the source. Since the target graph is a subgraph of the source of the composition, it is easy to see that an entire path must have been removed and that this path must have a counterpart in the original graph. If the removed path and its counterpart are isomorphic (as subgraphs), we replace each of them in the graph before the composition with a single identical edge. It is easy to see that performing a reduction on this new graph will be a 1-arrow where the target graph is a subgraph of the source graph, and thus will correspond to one of the cases in Lemma 3. The only remaining point is where the two paths are not isomorphic but where there is a monomorphism from one to the other. The explanation of Figure 4.3 in this section describes this scenario.

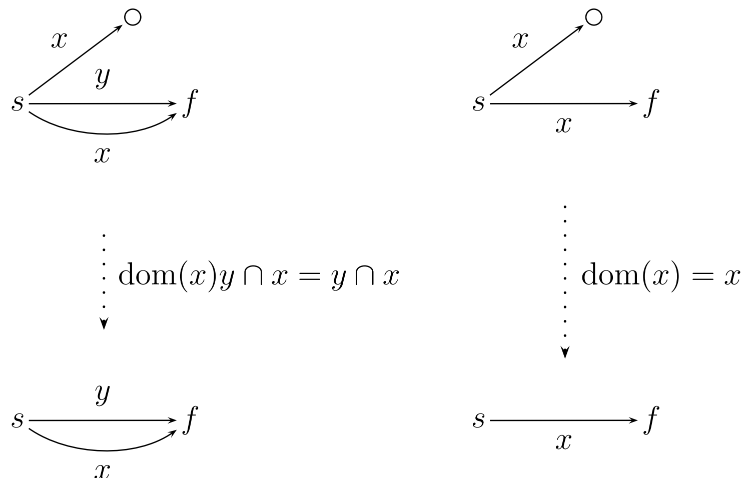


Figure 4.4: The reduction on the left side is an example of a 1-arrow where the corresponding rule (found by taking the algebraic subgraph) cannot be applied directly to any subterm of the equation.

4.2 Standardization Technique

Another problem that arises when attempting to extract a derivation mechanism occurs when attempting to apply rules directly to specific terms. For instance, a certain reduction may be described by one of the cases in Lemma 3, but it may not be possible to directly apply the rule corresponding to that case. An example of this is demonstrated in Figure 4.4 where the algebraic subgraph found in the reduction corresponds to the case where the rule $\mathrm{dom}(x)x = x$ should be used. However, when we translate the graphs into their respective terms, we end up with the equation $\mathrm{dom}(x)y \cap x = y \cap x$, and hence there is no obvious application of the rule since we cannot find the subterm $\mathrm{dom}(x)x$ in the equation.

Some previous work gives us a hint as to how we can overcome this problem. In Section 2.3 of [10], a standardization process is given in order to 'give a first approximation to normal forms for terms in the theory of allegories'. This process gives us the mechanics by which we can associate different terms that are represented by the same graph. For example, the first graph in Figure 4.4 could actually be translated as $\mathrm{dom}(x)y{\cap}x$, $\mathrm{dom}(x)(x{\cap}y)$, $\mathrm{dom}(x)(y \cap x)$ and even $\mathrm{dom}(x)x \cap y$ etc. The standardization process gives us a standard form for these terms (modulo commutativity). This is done by applying a sequence of equations to an original term which in turn gives us the standard form. We refer the reader to [10] for the details of this process.

The bottom line here is that we have a technique by which we can associate terms whose graphs are in the same equivalence class. For the example described in Figure 4.4, although we cannot work directly with the term of the form $\mathrm{dom}(x)y \cap x$, we could apply the rule if the term were in the form $\mathrm{dom}(x)x \cap y$. Since both of these terms are in the same equivalence class (i.e. they are both represented by the same graph), we can use the standardization process to create a proof of their equality. Then, we can apply the rule $\mathrm{dom}(x)x = x$ as desired. Specifically, we would derive the standard form for each of the two terms, and then combine the two derivations to provide an overall derivation of $\mathrm{dom}(x)y \cap x = \mathrm{dom}(x)x \cap y$, as follows:

$$\begin{aligned}
\mathrm{dom}(x)y \cap x &= (1 \cap xx^\circ)y \cap x \\
&= (1 \cap xx^\circ)(x \cap y) \\
&= (1 \cap xx^\circ)(y \cap x) \\
&= (1 \cap xx^\circ)x \cap y \\
&= \mathrm{dom}(x)x \cap y
\end{aligned}$$

The second and fourth steps of the above derivation are generated using a symmetric version of the equation $x \cap y(1 \cap z) = (x \cap y)(1 \cap z)$, which is one of the steps of the standardization process. The other steps correspond to the application of the rules $x \cap y = y \cap x$ and $\mathrm{dom}(x) = 1 \cap xx^\circ$. The middle of the derivation demonstrates that the standard form of the beginning and ending term is $(1 \cap xx^\circ)(y \cap x)$.

The following theorem generalizes this process:

Theorem 12. *Let $t_1, t_2 \in ALL$. If the terms t_1 and t_2 are in the same equivalence class (according to their graphical structure), then* $\mathrm{ALL} \vdash t_1 = t_2$. *Specifically, we could derive $t_1 = t_2$ from the equations in E_s.*

Proof. The process (outlined in [10]) to standardize terms gives us a proof of this fact, since there will be a derivation of each term t_1, t_2 to their equivalent standard forms. $\square$

Chapter 5

Implementation

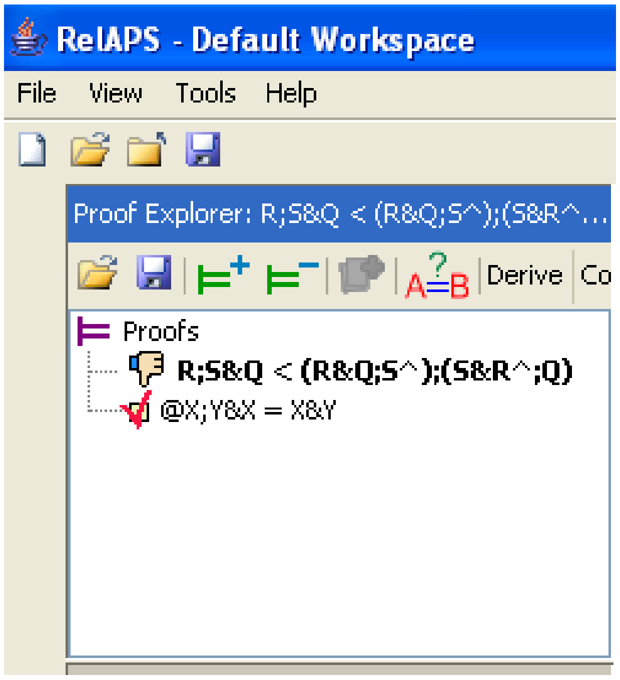
This chapter will discuss the implementation of our work in a software proof assistant called RelAPS [9]. We will start by giving a brief overview of the RelAPS system by discussing its purpose and some of its functionality. We then discuss how we implemented our research in the RelAPS system, that is, how we have extended RelAPS to produce some derivations automatically.

As RelAPS has been developed using the Java programming language, some of the classes used to implement the results of this work are described in Appendix B.

5.1 RelAPS - An Overview

RelAPS was designed with the intent of developing a system that would allow a user to complete relation-algebraic proofs as if they were being done by hand. Originally, the system would not provide any assistance with respect to completing proofs. As a user would attempt a proof, the system would simply provide a list of possible rules that could be applied to a certain situation. The user then has the freedom to choose the rule that he/she wished to apply to a selected term or formula. Hence, the system provides verification only in the sense that a user cannot inappropriate apply a rule, i.e., an invalid proof-step is not a possibility. However, there is no guarantee that an arbitrary proof-step will lead the user closer to the goal.

The bottom line is that RelAPS provides an environment which allows a user to feel as if he/she is doing a proof on a piece of paper, but with some help provided in the form of lists of rules which can be applied at each step.


RelAPS - Default Workspace
File
View
Tools
Help
Proof Explorer: R;S&Q < (R&Q;S^);(S&R^...
Derive
Co
Proofs

R;S&Q < (R&Q;S^);(S&R^;Q)

@X;Y&X = X&Y

Figure 5.1: The proof explorer view of the RelAPS system.

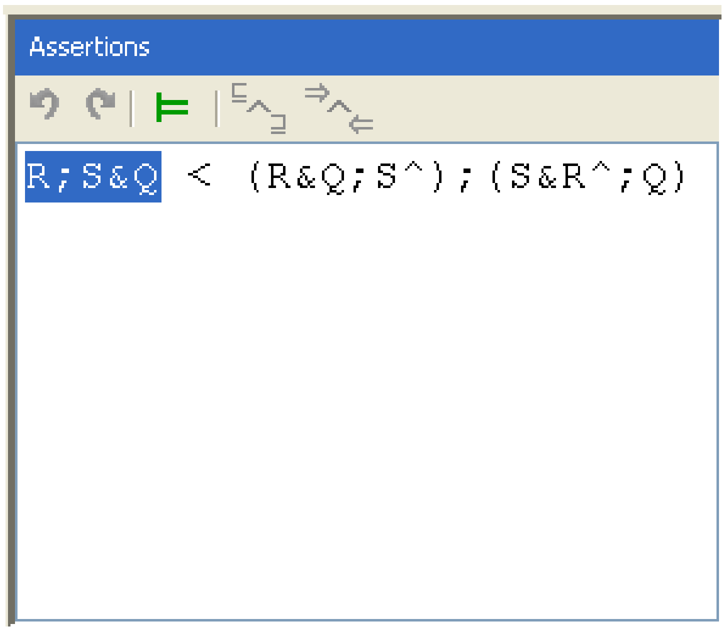
The system also lets the user know when a proof is complete - although this should be obvious to the user.

5.1.1 The Interface

The system's graphical user interface is made up of four main views: the proof explorer view, the assertions view, the assumptions view, and the main work area.

The proof explorer provides a tree-view of all the current proofs that are currently active. The user may select any of them from the tree and work on the proof by using the other views, discussed below. The user may also add other formulas to the system that he/she wishes to work on, each of which will be displayed in this view. Figure 5.1 is a screenshot of the proof explorer.

The assertions view allows the user to select which part of an assertion is to be modified in the working area. Within the assertions view the user



```
R;S&Q < (R&Q;S^);(S&R^;Q)
```

Figure 5.2: The assertions view of the RelAPS system.

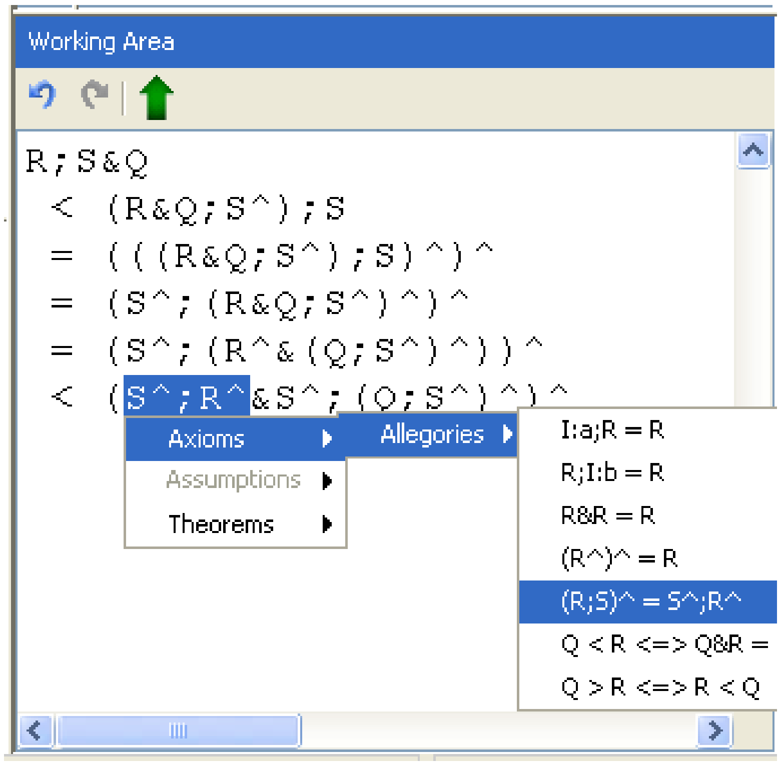
may also ‘split’ either an equation into two inclusions, or an equivalence into two implications. Once the user has selected which term of a formula he/she would like to modify, the is then made available in the working area where the user may apply appropriate rule(s). A screenshot of the assertions view with a sample formula is displayed in Figure 5.2.

The assumptions view is similar to the assertions view. It simply displays the assumptions of a formula, i.e., those formulas A_i in a horn formula of the style

$$A_1 \wedge A_2 \wedge A_3 \wedge ... \wedge A_n \Rightarrow B$$

where each A_i and B are atomic formulas. The user may also select any formula in the assumption view and modify it in the working area.

The working area is the important component of the interface when considering how one actually constructs a proof. After the user has decided which term is to be manipulated, the system allows the user to apply rules to the term in the working area. This is done by using the mouse to select a



```
R;S&Q
  <  (R&Q;S^);S
  =  (((R&Q;S^);S)^)^
  =  (S^;(R&Q;S^)^)^
  =  (S^;(R^&(Q;S^)^))^
  <  (S^;R^&S^;(Q;S^)^)^
```

Figure 5.3: The working area of the RelAPS system.

term (or subterm) to which the user wishes to apply a rule. Once the term is selected, a menu appears displaying those rules which are applicable within the current context. Figure 5.3 is a screenshot of a scenario where the user has selected a term and the menu has appeared displaying all applicable rules within a given theory. If the user selects a rule, the system then generates the next line of the derivation. Once the user is satisfied that the derivation of the current term is complete, he/she may press the button which applies the derivation to the formula displayed in the assertions view. The working area also allows the user to undo or redo any step of the proof.

RelAPS also offers additional functionality which includes, but is not limited to, defining new theories, defining new operations, proving monotonicity of operations, etc. We refer the reader to `http://www.joelglanfield.com/relaps/` to learn more about the RelAPS system.

5.2 Extending RelAPS

When we talk about *extensions* to RelAPS in this section, we simply refer to additions to the system which fall outside of its original purpose.

For instance, the first extension to RelAPS was the implementation of the decision algorithm with respect to the equational theory of allegories. We have already discussed this algorithm in detail in Section 2.3. When using RelAPS, if the user specifies that he is working within the theory of allegories and enters an equation, then a button becomes active which allows the user to ask the system whether there is a proof of the equation. Once the button is pressed, the system responds with either a ‘yes’ or ‘no’, depending on whether the equation is derivable. Notice that this process only lets the user know whether there is a derivation of the equation, but does not provide the derivation itself.

Although RelAPS was not designed with the intent of providing automated derivation, we felt that it would be a suitable system for testing the results of this work. In a similar vein to the extension discussed in the preceeding paragraph, we felt that we could extend the system to allow a user to simply press a button that will then tell the system to generate a proof of any provable equation in ALL. The same restrictions apply (as discussed in the preceeding paragraph), namely, the user must be working within the theory of allegories and must have entered an equation.

Figure 5.4 displays the situation where a user has entered a provable

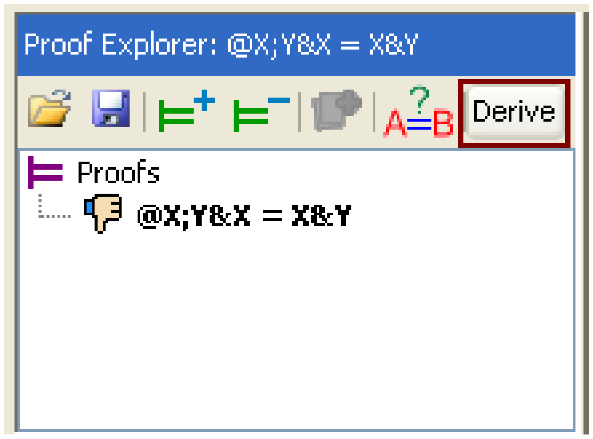


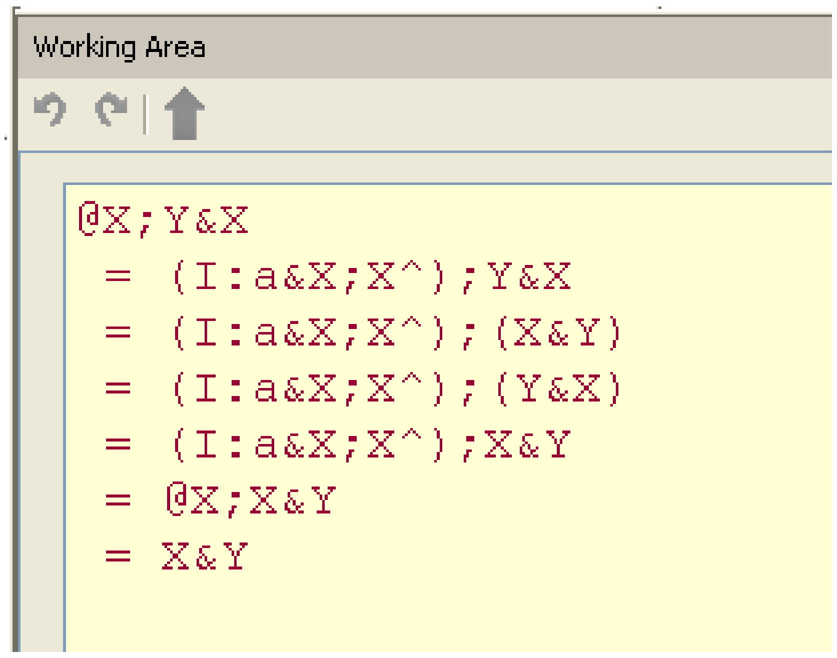
Figure 5.4: The proof explorer showing the formula the user will ask the system to prove.

equation into the system. Notice that a button is enabled which allows the user to tell the system to find a derivation (the button is labelled ‘Derive’). Once the user presses the button, the system attempts to find a derivation of the formula.

Figure 5.5 shows how the system was successful in finding a derivation which proves the left side of the formula is equal to the right side. Since the algorithm is derived from the decision procedure, the system has essentially followed the process of finding the normal form of the left side of the equation. Since the right side of the equation is already in its normal form, no derivation is attempted. One may also realize that lines 2 to 6 of the derivation shown in Figure 5.5 demonstrate the implementation of the standardization procedure discussed in Section 4.2. Thus, although there are several lines to the derivation, there is really only one reduction step since only one vertex would have been removed during the process. The left side of Figure 4.4 demonstrates this situation specifically.

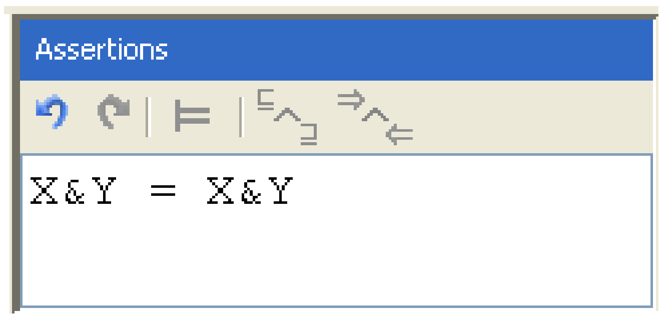
Figure 5.6 demonstrates how the assertions area of the system shows the user that the derivation was applied to the left side of the equation.

Figure 5.7 shows how the proof explorer was updated to show that the derivation of the formula is complete. The user may now view the proof of the equation whenever it is loaded into the system. At this point the user may wish to move on to proving other formulas.



```
@X;Y&X
 =  (I:a&X;X^);Y&X
 =  (I:a&X;X^);(X&Y)
 =  (I:a&X;X^);(Y&X)
 =  (I:a&X;X^);X&Y
 =  @X;X&Y
 =  X&Y
```

Figure 5.5: The working area displaying the derivation of a formula.



```
X&Y  =  X&Y
```

Figure 5.6: The assertions area displaying a formula which appears to have been derived.

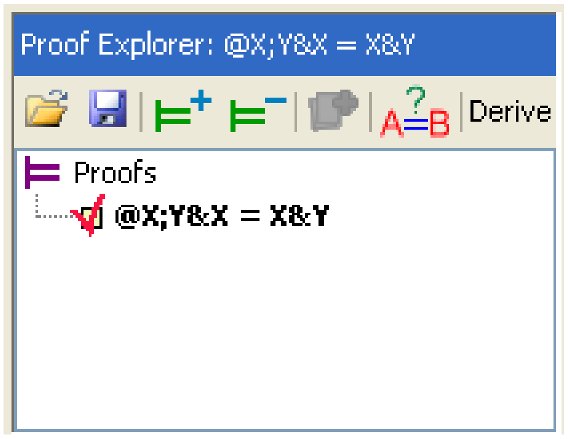


Figure 5.7: The proof explorer displaying that a formula has been derived.

This extension to the RelAPS system also provides the unintended benefit of producing a piece of software which may help students learn how to do relation-algebraic proofs. A user may enter an arbitrary formula and attempt to do a derivation, or he/she may wish to determine whether the formula is derivable by applying the decision procedure. The user may then tell the system to produce a derivation. Combining these three actions allows a user to see how formulas may be derived, which having the option to attempt the derivation his/herself.

Chapter 6

Conclusions

We now revisit the question of whether we can give a complete derivation procedure for any provable theorem in ALL, using the main work in [10] on decidability as a basis. After showing how we have answered this question we will suggest some motivation and ideas for future work.

6.1 Summary of Findings

The seemingly unfortunate answer to our original question is a resounding 'no'. We cannot show completeness for a derivation algorithm that is extracted from the previous work on decidability. We have already demonstrated that attempting to extract such a mechanism from the previous work cannot be done simply because there are some significant errors in the proofs of some of the major lemmas outlined in [10]. The main error occurs in the h^*-generation procedure that is used to prove Lemma 5 regarding what can be concluded about 1-arrows. It is very likely that the statement of the lemma is indeed correct, but the proof mechanism is incorrect as we have shown by contradiction in Lemma 10.

An unfortunate conclusion that must be drawn from the fact that Lemma 5 has not technically been proven is that the main theorem in [10] has actually not been proven correctly either, since it relies on the proof of this lemma. However, this has not stopped us from at least providing a derivation mechanism that will still derive many provable theorems in ALL; just not all theorems. Proving many theorems in ALL can be done by simply considering Lemma 3 where we restrict ourselves to 1-arrows where there is a subgraph from the target graph to the source graph. We then simply classify the 1-arrow according to the cases described in the same lemma and then apply the appropriate equation. We have demonstrated that we must add a couple of cases that do not appear in [10] (which we discuss in Appendix A).

Also, we have shown that it is still very likely that one could prove a complete derivation procedure that is still largely based on the work in [10]. We have shown that we must consider so-called non-trivial 1-arrows where the target graph of the arrow is not a subgraph of the source. As discussed in Section 4.1.2, we must look at n-reductions in general, where we consider replacing complete paths with single edges. We have also noted that it is possible for situations to arise where we must replace two paths with the same edge where one path is a subpath of another. This can be overcome using Lemma 4 regarding 0-arrows, since there would be an embedding from the subpath to the main path.

Another significant finding is that we have shown how specific equations can always be directly applied to subterms in the proper situations. It does not matter whether we can find an appropriate subterm, since previous standardization techniques have indirectly given us a mechanism whereby we can associate terms that are equal. This is done by standardizing two equal terms (which have different syntactic representations) given the procedure in [10] and then combining the derivations, as explained in Section 4.2. This finding is significant when we consider implementation issues, since it is likely that when implementing a derivation mechanism in a software proof assistant that one would need to consider how to apply rules directly to terms.

6.2 Future Work

Some obvious future work remains to be explored. The main issue at this point is completeness. If we are to generate a complete derivation mechanism based on the work in [10], then the proof of Lemma 5 in that work must first be repaired. This is outside of the scope of this work. However, one may consider taking a different approach and try to prove completeness for the method suggested in Section 4.1.2 of this work, for which details are given in Claim 11.

Some other interesting future work would include, but not be limited to, implementing our proposed derivation mechanism in existing theorem provers. We have provided an implementation for the system called 'RelAPS' (see Chapter 5).

Once completeness has been shown, it may be interesting to consider whether the derivation mechanism could be extended to handle Horn-style formulae (i.e. $A_0 \wedge A_1 \wedge ... \wedge A_n \Rightarrow B$).

Bibliography

[1] Roland Carl Backhouse and Paul F. Hoogendijk. Final Dialgebras: From Categories to Allegories. *Informatique Theorique et Applications*, 33(4/5):401–426, 1999.

[2] Richard Bird and Oege de Moor. *Algebra of Programming.* Prentice Hall, 1996.

[3] Carolyn Brown and Graham Hutton. Categories, Allegories and Circuit Design. In *IEEE Symposium on Logic in Computer Science*, July 1994.

[4] Carolyn Brown and Alan Jeffrey. Allegories of Circuits. In *Logical Foundations of Computer Science*, pages 56–68, 1994.

[5] Shiqi Cao and Wolfram Kahl. An Implementation of Gutiérrez' decision procedure for the equational theory of allegories, 2005. Presented at Relation Day, Brock University.

[6] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Springer-Verlag, New York, USA, 1995.

[7] Peter J. Freyd and Andre Scedrov. *Categories, Allegories.* North-Holland, 1990.

[8] Hitoshi Furusawa and Wolfram Kahl. A study on symmetric quotients. `http://citeseer.ist.psu.edu/furusawa98study.html`.

[9] Joel Glanfield. Relaps: A proof assistant for relational categories. `http://www.joelglanfield.com/relaps/`.

[10] Claudio Gutiérrez. *The Arithmetic and Geometry of Allegories - normal forms and complexity of a fragment of the theory of relations.* PhD thesis, Wesleyan University, 1999.

[11] Graham Hutton, Erik Meijer, and Ed Voermans. A Tool for Relational Programmers. Mathematics of Programming (MOP) mailing list, January 1994.

[12] Mark Jones. Introduction to gofer 2.20. `http://web.cecs.pdx.edu/~mpj/goferarc`, 1992.

[13] J.P. Olivier and D. Serrato. Catégories de Dedekind. Morphismes dans les catégories de Schröder. *C.R. Acad. Sci. Paris*, 290:939–941, 1908.

[14] Gunther Schmidt and Thomas Ströhlein. *Relations and graphs: discrete mathematics for computer scientists.* Springer-Verlag, London, UK, 1993.

[15] Michael Winter. A new algebraic approach to L-fuzzy relations convenient to study crispness. *Inf. Sci.*, 139(3-4):233–252, 2001.

[16] Larry Wos and Gail W. Pepper. *Automated Reasoning and the Discovery of Missing and Elegant Proofs.* Ave Maria Press, 2003.

Appendix A

Updating the State of the Art

We have found that, in order to implement Lemma 3 in software, it was necessary to add a couple of additional cases to those enumerated in [10]. We also add a correction to Case 4 part (c). We will start by suggesting a correction to Case 4, then we will enumerate the two extra cases we have alluded to. In Section A.3 we provide diagrams for the other possible reductions.

A.1 A Correction

For Part (c) of Case 4 (corresponding to Figure 26 in [10]), a close inspection will show that this scenario does not really describe the minimum algebraic subgraph corresponding to a 1-arrow. The equation corresponding to this case is:

$$xy \cap (x \cap v)(y \cap w) \cap z = (x \cap v)(y \cap w) \cap z \tag{88}$$

Since the edge corresponding to the term z appears on both sides of the 1-arrow and is not an influencing factor on the vertices which are identified, it is safe to remove this edge from the minimum algebraic subgraph and thus end up with the equation:

$$xy \cap (x \cap v)(y \cap w) = (x \cap v)(y \cap w) \tag{88}$$

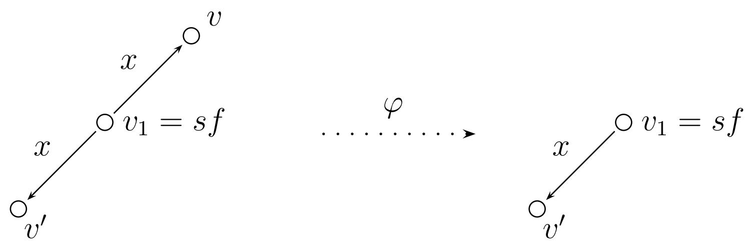


Figure A.1: $p = 1$ and $q = 0$ and $s_m = f_m = v_1$.

A.2 Some Additions

Part (a) of Case 2 in [10] describes the scenario relating to the equation

$$\operatorname{dom}((x \cap y)x^\circ) = \operatorname{dom}(x \cap y). \tag{82}$$

The minimum algebraic subgraph related to this term contains three vertices, where v' is both the start and the finish vertex (see page 74 of [10] for a description of how to construct the minimum algebraic subgraph). However, there is no case to describe a similar situation where the vertex v_1 is actually both the start and the finishing vertex. This occurs whenever there is an intersection (or composition) of independent dom terms.

For example, if we consider the terms $\operatorname{dom}(x) \cap \operatorname{dom}(x)$ and $(\operatorname{dom}(x))(\operatorname{dom}(x))$ (both of which are obviously equal), we would have the scenario demonstrated in Figure A.1.

This is a situation where we would employ the equation $a \cap a = a$ even though this case is not a 0-arrow.

The other addition we offer is related to the scenario described in Part (d) of Case 4. Our motivation for this addition came when attempting to classify the second 1-arrow of the scenario demonstrated in Figure 17 of [10]. The equation that relates to the 1-arrow shown is not sufficient to reason about scenarios where the subgraph D12 does not occur. This type of reduction introduces a dom construction on the right hand side. Figure A.2 in this work describes this situation.

We introduce the equation

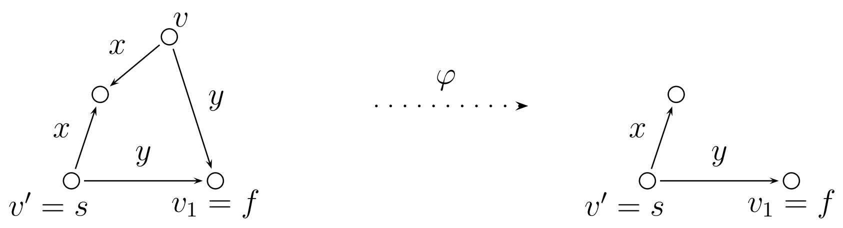


Figure A.2: $p = 1$ and $q = 0$ and $v' = s_m, v_1 = f_m$.

$$xx^\circ \cap y = \mathrm{dom}(x)y$$

to reason about this reduction. The proof is done in two steps; the first is the right inclusion:

$$\begin{aligned} \mathrm{dom}(x)y &= (1 \cap xx^\circ)y && [(54)\ \textit{in}\ [10]] \\ &\subseteq xx^\circ y \cap y && [(38s)\ \textit{in}\ [10]\ \textit{with}\ v = u] \end{aligned}$$

The converse inclusion is derived using the modular axiom (39) in [10].

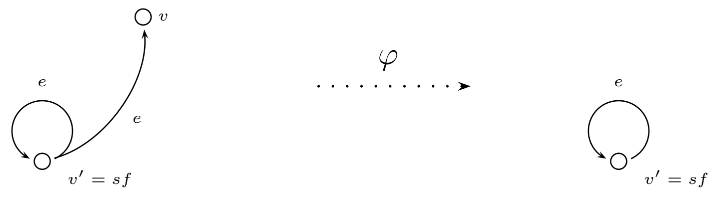


Figure A.3: $p = 0$ and $q \geq 1$.

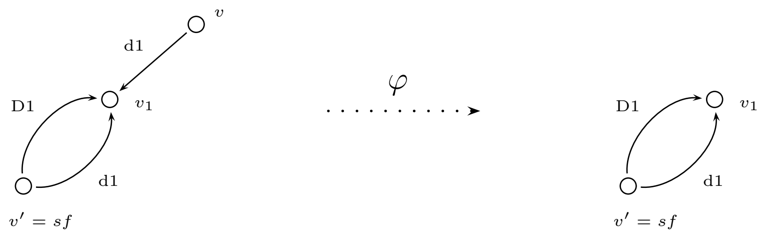


Figure A.4: $p = 1$ and $q = 0$ and $s_m = f_m = v'$.

A.3 Diagrams of Reductions

In this section we provide diagrams of the other possible reductions in ALL. These diagrams are already given in [10], but we add them here for convenience. We refer the reader to [10] for further details (Chapter 5 specifically).

Figure A.3 demonstrates the need for the equation

$$(1 \cap x)(\text{dom}x) = 1 \cap x.$$

Figure A.4 demonstrates the need for the equation

$$\text{dom}((x \cap y)x^\circ) = \text{dom}(x \cap y).$$

Figure A.5 demonstrates the need for the equation

$$\text{dom}(x)x = x.$$

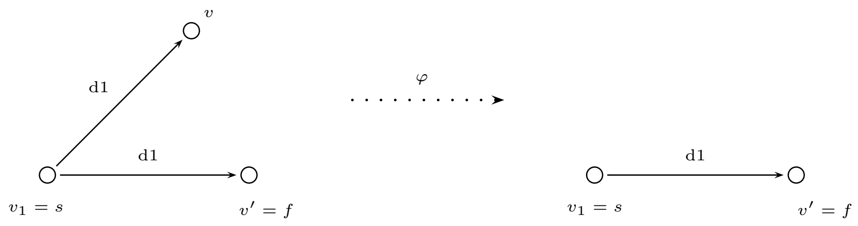


Figure A.5: $p = 1$ and $q = 0$ and $s_m = v_1, f_m = v'$.

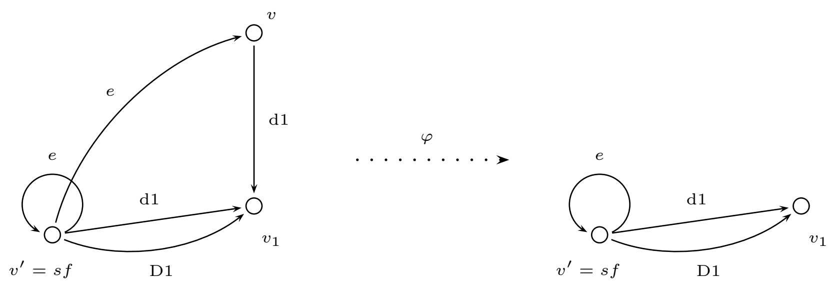


Figure A.6: $p = 1$ and $q \geq 1$ and $s_m = f_m = v'$.

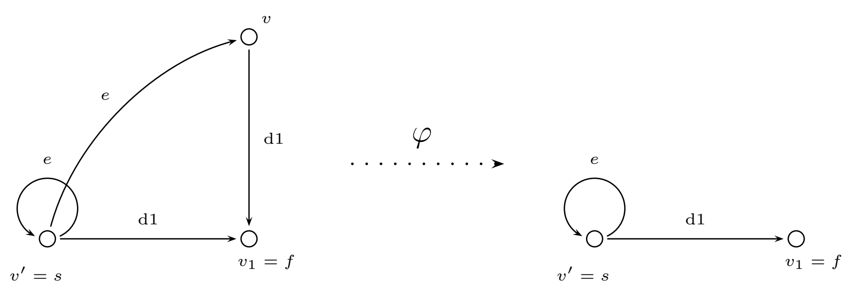


Figure A.7: $p = 1$ and $q \geq 1$ and $s_m = v', f_m = v_1$.

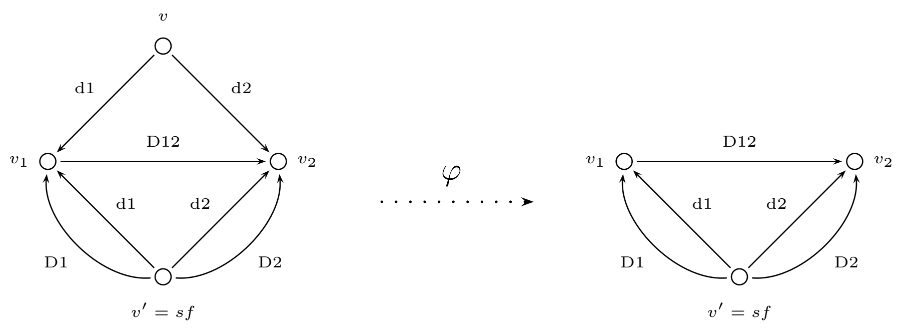


Figure A.8: $p = 2$ and $q = 0$ and $s_m = f_m = v'$.

Figure A.6 demonstrates the need for the equation

$$(1 \cap x)(1 \cap (y \cap z)y^\circ x) = (1 \cap x)\mathrm{dom}(z \cap y).$$

Figure A.7 demonstrates the need for the equation

$$(1 \cap x)((xy) \cap y) = (1 \cap x)y.$$

Figure A.8 demonstrates the need for the equation

$$1 \cap (x \cap u)(x^\circ y \cap v)(y^\circ \cap w) = 1 \cap (x \cap u)v(y^\circ \cap w).$$

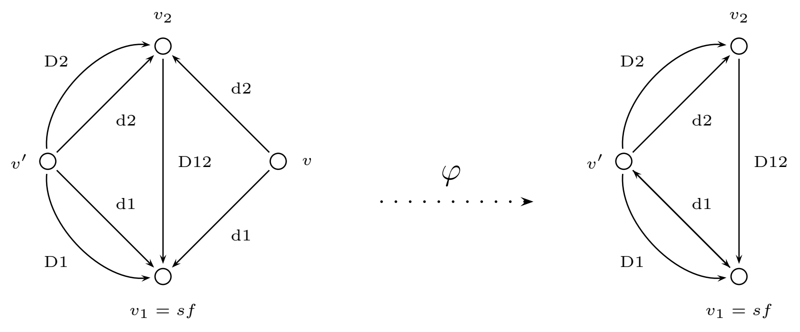


Figure A.9: $p = 2$ and $q = 0$ and $s_m = f_m = v_1$.

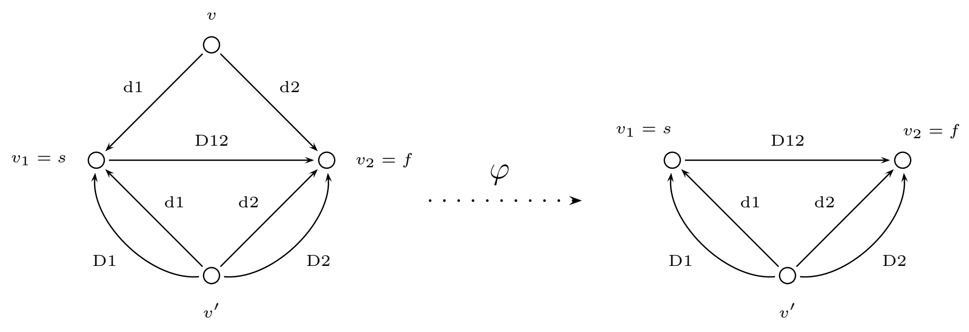


Figure A.10: $p = 2$ and $q = 0$ and $s_m = v_1$ and $f_m = v_2$.

Figure A.9 demonstrates the need for the equation

$$1 \cap [(x^\circ \cap v)(y \cap w)(y^\circ x \cap u)] = 1 \cap [(x^\circ \cap v)(y \cap w)u] .$$

Figure A.10 demonstrates the need for the equation

$$xy \cap (x \cap v)(y \cap w) \cap z = (x \cap v)(y \cap w) \cap z.$$

Figure A.11 demonstrates the need for the equation

$$(x \cap zy^\circ)(y \cap u) \cap z = x(y \cap u) \cap z.$$

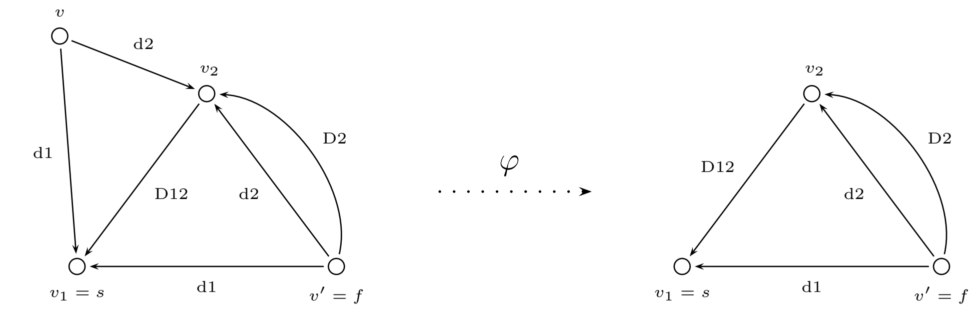


Figure A.11: $p = 2$ and $q = 0$ and $s_m = v_1$ and $f_m = v'$.

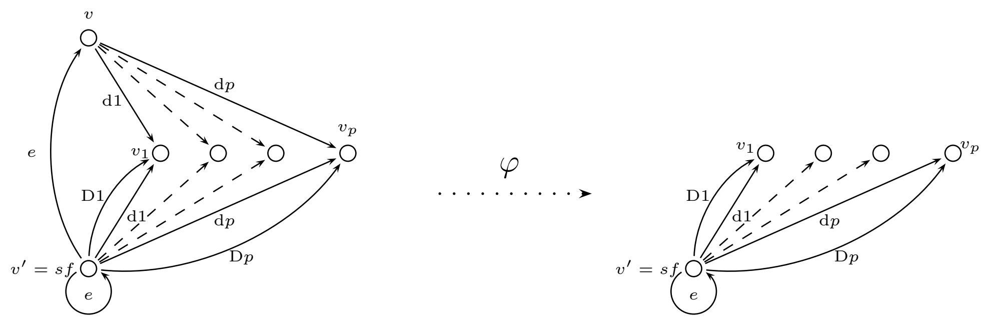


Figure A.12: $p \geq 2$ and $q \geq 1$ and $s_m = f_m = v'$.

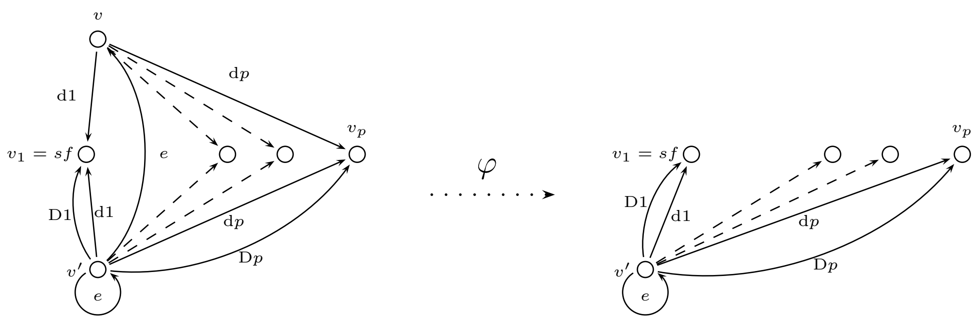


Figure A.13: $p \geq 2$ and $q \geq 1$ and $s_m = f_m = v_1$.

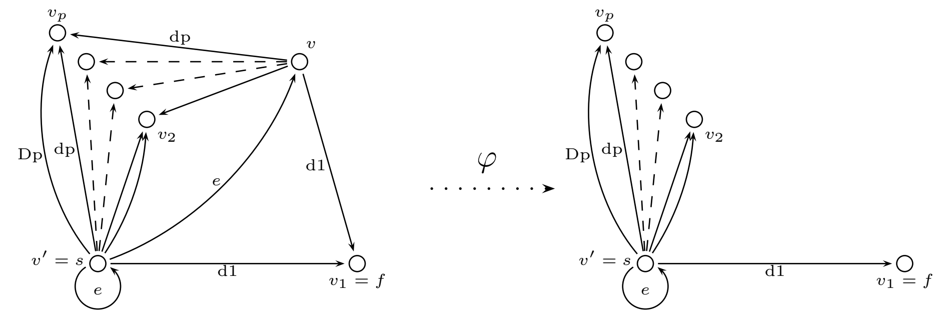


Figure A.14: $p \geq 2$ and $q \geq 1$ and $s_m = v'$, $f_m = v_1$.

Figure A.12 demonstrates the need for the equation

$$(1 \cap x)(1 \cap \bigcap_i (y_i \cap z_i) y_i^\circ c^\circ) = (1 \cap x) \bigcap_i \mathrm{dom}(z_i \cap y_i).$$

Figure A.13 demonstrates the need for the equation

$$1 \cap y_1^\circ (x \cap \bigcap_{i \geq 2} y_i (y_i^\circ \cap z_i^\circ)) = \mathrm{dom}((y_1 \cap z_1)^\circ (1 \cap x) \bigcap_{i \geq 2} \mathrm{dom}(y_i \cap z_i)).$$

Figure A.14 demonstrates the need for the equation

$$(1 \cap x)((x \cap \bigcap_{i \geq 2} (y_i \cap z_i) y_i^\circ) y_1 \cap y_1) = (1 \cap x)(\bigcap_{i \geq 2} \mathrm{dom}(y_i \cap z_i)) y_1.$$

Appendix B

Class Diagrams

This appendix describes the relationship between the main classes in the RelAPS system which contribute to the implementation of the derivation algorithm described in this work.

Figure B.1 provides an overview of the interfaces which allow a derivation to be generated. The Formula and Term interfaces are the base types of those concrete objects which are relational formulas and terms respectively. The GraphTerm interface correspond relational terms to the types of graph operations described in Section 2.3. An object of type GraphTerm can be converted into a concrete graph, whereas a graph can always provide the concrete Term it represents (hence the toTerm() method in the Graph class).

The algorithm interface allows our derivation mechanism to be implemented, and is used by the graphical interface to display a derivation of some provable theorem.

Figure B.2 describes the concrete classes which provide the algorithms described in this work. The first algorithm implemented in the RelAPS system was the decision algorithm for allegories given in [10]; hence the implementation of the DecisionAlg class. The DerivationAlg class is the concrete implementation of the results of this work. The generateProof() method generates a Proof object used by the interface to display a derivation. The StandardForm class is an implementation of the standardization technique described in Section 4.2.

Figure B.3 gives an overview of the concrete implementation of those graphs corresponding to the operations described in Section 2.3. The GraphToTermConverter class is a utility class where the convert() function converts a Graph (generated by the createGraph() function in the GraphTerm inter-

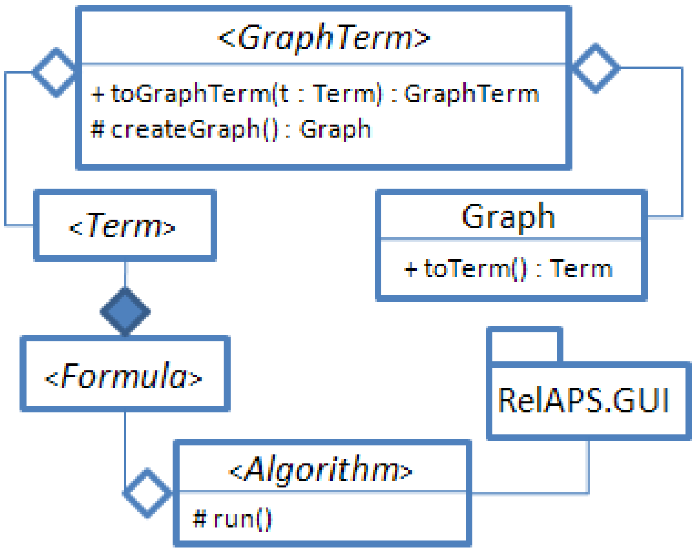


Figure B.1: Overview of the class structure used to implement the results of this work.

face) into a concrete Term. Thus, at any step during the normalization process we can determine the term corresponding to a subgraph.

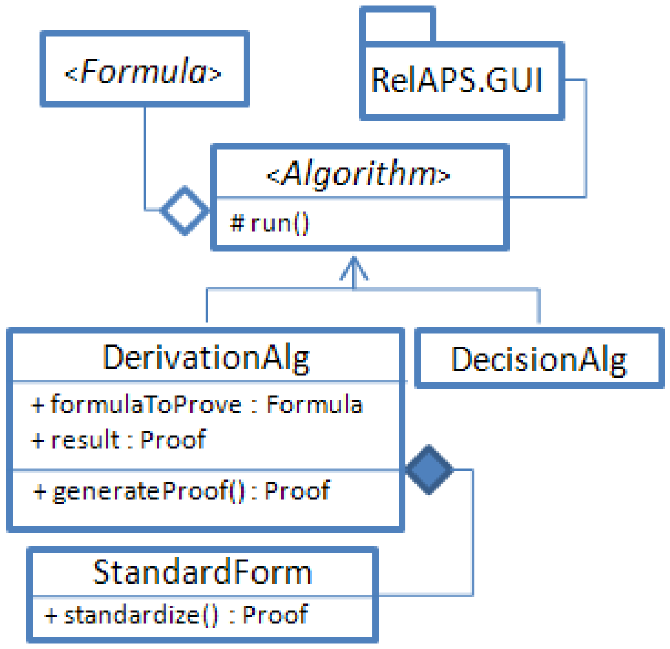


Figure B.2: Overview of the classes responsible for executing the derivation mechanism.

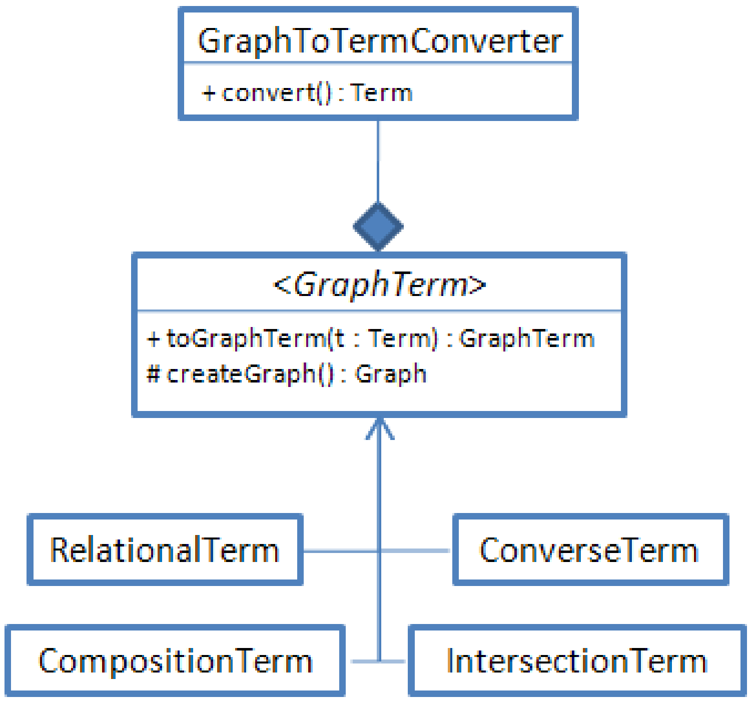


Figure B.3: Overview of GraphTerm classes.