An Algebraic View of Inductive Equational Logic Programming

Lutz H. Hamel
Department of Computer Science and Statistics
University of Rhode Island

TR00-278

December 2000

Abstract

We present an algebraic semantics for inductive equational logic programming. Inductive Logic Programming (ILP) concerns itself with the induction of first-order Horn clause theories from facts. Similarly, inductive equational logic programming concerns itself with the induction of first-order equational theories from ground equations representing facts. Traditionally the literature has treated the semantics for the induction of equational theories purely from an operational viewpoint based on directed rewrite rules. In this paper we take advantage of the strong ties that exist between equations and algebra and develop a model theoretic approach to inductive equational logic programming based on algebraic ideas.

1 Introduction

The past decade saw an increasing interest in machine learning as a way to extract knowledge from large databases of facts. Formidable advances in this field have yielded tools and algorithms that have been applied to non-trivial problems providing insights not possible before [12, 19, 21, 22].

Inductive Logic Programming (ILP) is particularly interesting in this context because of its explanatory power due to the fact that it induces first-order Horn clause theories from a set of facts [20]. ILP can be seen as logic programming with inverted deduction. In other words, inductive logic programming is logic programming with inductive inference rather than deductive inference. Since ILP inherits its representational formalism from Logic Programming [23], it is particularly easy to incorporate background or domain knowledge into the induction which is crucial in many problem settings.

Recently other logics than Horn clause logic have been used in the context of ILP. In particular, equational logic has been proposed for inductive equational logic programming. Equational theories are attractive due to their expressiveness and their straight forward denotational semantics based on algebra [5, 7]. These advantageous properties of equational theories have been harnessed for software specification languages such as OBJ3 [9] and ACT-ONE [4] as well as industrial strength equational logic programming environments such as UCG-E [11].

In the inductive equational logic programming setting the semantics of inductive inference are usually developed by viewing equations as directed rewrite rules and the usual deductive term rewrite system is replaced by an inductive inference machinery. Early approaches in this area focused primarily on the synthesis of equational programs [1, 3]. More recently, equational approaches have emerged that resemble the traditional framework of ILP more closely [14, 15]. However, even these recent approaches focus almost exclusively on the operational aspect of the equations. In this paper we take a different approach, rather than developing an operational semantics based on term rewriting we explore the connection between equational logic and algebra and develop a more model theoretic approach to inductive equational logic programming. The aim is to provide an abstract characterization of inductive equational logic programming independent from the exact nature of any particular inductive inference system.

As a starting point for our development we use the *normal semantics for ILP* that interpret background knowledge, hypotheses, positive and negative facts for clausal theories [6, 20]. We generalize these notions and cast them into an algebraic framework. In this setting we investigate the concept of induced equational theory, background knowledge, and positive and negative facts. In particular, we show that the algebras that satisfy an induced equational theory also satisfy the background knowledge and the positive facts. In addition, we investigate syntactic and semantic bias within this semantic framework. The overarching goal in this model theoretic approach is to provide improved clarity and insight into inductive equational logic programming.

The remainder of the paper is structured as follows. Section 2 introduces basic concepts and the normal semantics of ILP. Algebra and algebraic theories are presented in Section 3. Section 4 introduces our algebraic notions for inductive equational logic programming. We take a closer at the details of our semantics by working an example in Section 5. In Section 6 we examine declarative bias within the context of our semantic framework. Finally, we draw conclusions in Section 7 and point to further research directions. In order to make this paper as self contained as possible, Appendix A contains a somewhat lengthy algebra primer.

2 Inductive Logic Programming

The discipline of Inductive Logic Programming (ILP) concerns itself with the induction of clausal theories (hypotheses) from facts and background knowledge. Formalizing this a little bit we can state the normal semantics for ILP as follows [6, 20].

Definition 1 Given a set B of horn clause definitions (background theory), a set P of ground facts to be entailed (positive examples), a set N of ground facts not to be entailed (negative examples), and a hypothesis language L, then a construct $H \in L$ is an **hypothesis** if

$$\begin{aligned} &B \cup H \models p, \ \text{for every } p \in P \text{ (Completeness)}, \\ &B \cup H \not\models n, \ \text{for every } n \in N \text{ (Consistency)}. \end{aligned}$$

□

Here, L is the set of all well-formed logical formulae over a fixed vocabulary. *Completeness* states that the conjunction of the background and the hypothesis entail the positive facts. In other words, together the background knowledge and the hypothesis can explain the facts. *Consistency* states that the background and the hypothesis do not entail the negative facts. On a more intuitive level, in order for the hypothesis to be an explanation of the positive facts it should not hold for any of the counter examples.

Please note that this semantic definition does not say anything about the quality of a particular hypothesis. In fact, it is interesting to note that this semantic definition admits a number of trivial solutions. For instance, let $H = P$. Also consider the case where $B \models p$ for every $p \in P$. Typically, the weighing of one hypothesis over another is left to the operational semantics of an ILP system. In practical ILP systems trivial solutions like the ones above are typically immediately dismissed by the system on its search for an "optimal" hypothesis, since these trivial solutions tend not to pass a set of performance heuristics when compared to other more general hypotheses.

3 Algebra and Theories

Before we proceed with the construction of our algebraic semantics we present here some basic algebraic notions. For a more explicit presentation see one of the following [5, 17, 24]. A more detailed introduction to algebraic ideas is given in the Apendix A. The exposition here closely follows [8].

An equational signature defines a set of sort symbols and a set of function symbols. More precisely,

Definition 2 An **equational signature** is a pair (S, Σ), where S is a set of sorts and Σ is an $(S^* \times S)$-sorted set of operation names. $\sigma \in \Sigma_{w,s}$ is said to have arity $w \in S^*$ and sort $s \in S$. Usually we abbreviate (S, Σ) to Σ. $\Box$

Mappings between signatures map sorts to sorts and operator symbols to operator symbols.

Definition 3 An **equational signature morphism** is a pair of mappings $\phi = (f, g)\colon (S, \Sigma) \to (S', \Sigma')$, we write $\phi\colon \Sigma \to \Sigma'$. $\Box$

A presentation or theory[1] is an equational signature with a collection of equations. In more precise terms, an equational presentation or theory is defined as follows.

Definition 4 A **Σ-presentation** or **Σ-theory** is a pair (Σ, E) where Σ is an equational signature and E is a set of Σ-equations. Each equation in E

[1]We treat the terms *presentation* and *theory* as synonymous; the traditional notion of theory can be recovered by taking the closure of the set of equations.

has the form

$$(\forall X) l = r$$

where X is a set of variables distinct from the equational signature Σ and $l, r \in T_\Sigma(X)$ are the terms over the set Σ and X. If $X = \emptyset$, that is, l and r contain no variables, then we say the equation is **ground**.

When there is no confusion Σ-theories are referred to as theories and are denoted by their collection of equations, in this case E. □

The above can easily be extended to conditional equations[2]. However, without loss of generality we continue the discussion here based on unconditional equations only. Central to our development of an algebraic framework is the notion of satisfaction of a theory by an algebra.

Definition 5 Given a theory $T = (\Sigma, E)$, a Σ-algebra A is a T-model if A satisfies each equation $e \in E$. We write $A \models E$, when there is no confusion we write $A \models T$. □

In general there are many algebras that satisfy a particular theory. We also say that the class of algebras that satisfy a particular equational theory represent the denotational semantics of that theory.

Semantic entailment of an equation from a theory is defined as follows.

Definition 6 An equation e is **semantically entailed** by a theory (Σ, E), write $E \models e$, iff $A \models E$ implies $A \models e$ for all Σ-algebras A. □

We are now in a position to define theory morphisms.

Definition 7 Given two theories $T = (\Sigma, E)$ and $T' = (\Sigma', E')$, then a **theory morphism** $\phi\colon T \to T'$ is a signature morphism $\phi\colon \Sigma \to \Sigma'$ such that

$$E' \models \phi(e), \text{ for all } e \in E.$$

□

In other words, the signature morphism ϕ is a theory morphism if the translated equations of the source theory T are semantically entailed by the target theory T'.

Goguen and Burstall have shown within the framework of institutions [8] that the following holds for many sorted algebra[3]:

[2]Consider the conditional equation, $(\forall X)t = t'$ if c, which is interpreted as meaning the equality holds if the condition c is true.

[3]Actually, Goguen and Burstall have shown the much more powerful result that the implication holds as an equivalence relation. However, for our purposes here we only need the implication.

Theorem 8 Given the theories $T = (\Sigma, E)$ and $T' = (\Sigma', E')$, the theory morphism $\phi\colon T \to T'$, and the T'-algebra A', then

$$A' \models_{\Sigma'} \phi(e) \Rightarrow \phi A' \models_{\Sigma} e$$

for all $e \in E$. $\square$

In other words, if we can show that a given model of the target theory satisfies the translated equations of the source theory, it follows that the reduct of this model, $\phi A'$, also satisfies the source theory, thus, the models behave as expected.

4 Induced Theories

Key to our algebraic framework is the notion of induced theory which represents an equational theory that incorporates the given background knowledge and can explain the positive facts. An induced theory is not unlike the notion of an hypothesis in the normal semantics for ILP.

The goal of this algebraic semantics is to formulate a more model theoretic approach with respect to inductive equational logic programming rather than defining the semantics of induced theories in terms of rewrite relations. We make use of the fact that the denotation of an equational theory is the class of algebras that satisfy the equations of that theory. Here we let the algebras that satisfy the equations of an induced theory be the denotational semantics of the induced theory.

Let us define what we mean precisely by an induced theory. In order to do this we have to define what we mean by facts.

Definition 9 A theory (Σ, E) is called **Σ-facts** if each $e \in E$ is a ground equation. If there is no confusion we will refer to Σ-facts as facts and write E. $\square$

This allows us to define our notion of induced theory.

Definition 10 Given a background theory $B = (\Sigma_B, E_B)$, positive facts $P = (\Sigma_P, E_P)$, and negative facts $N = (\Sigma_N, E_N)$, then an **induced theory** $H = (\Sigma_H, E_H)$, is a theory with a pair of mappings ϕ_B and ϕ_P

$$\begin{tikzcd} & H & \\ B \arrow[ur, "\phi_B"] & & P \arrow[ul, "\phi_P"'] \end{tikzcd}$$

such that

- $\phi_B : B \to H$ is a theory morphism,
- $\phi_P : P \to H$ is a theory morphism,
- and $H \not\models \phi_N(e)$, for all $e \in E_N$, and signature morphism $\phi_N : \Sigma_N \to \Sigma_H$.

□

This definition warrants a closer look. Take the theory morphism ϕ_B that maps the background theory into the induced theory. From the definition of a theory morphism we have

$$\phi_B : B \to H \text{ is a theory morphism if } H \models \phi_B(e),\ \text{ for each } e \in E_B.$$

This is equivalent of saying that in order for this mapping to be valid the induced theory must semantically entail the given background knowledge.

A closer look at the theory morphism ϕ_P that maps the positive facts into the induced theory reveals a similar relationship. Again from the definition of a theory morphism

$$\phi_P : P \to H \text{ is a theory morphism if } H \models \phi_P(e),\ \text{ for each } e \in E_P.$$

This is equivalent of saying that in order for this mapping to hold the induced theory must explain the positive facts. In other words, this can be considered to be the *completeness* criteria of the normal ILP semantics defined in Definition 1 cast into an algebraic framework.

The last part of the definition above is of course the *consistency* statement that insures that only the positive facts and not the negative facts are entailed by the induced theory. More precisely, it states that the translated equations of the negative facts must not hold in the induced theory.

Sofar we have treated models that satisfy H implicitly. Since our denotational semantics is expressed through the algebraic models that satisfy the induced theory, let us make these notions explicit through the following proposition:

Proposition 11 (Denotational Semantics) Given an induced theory H, with the background theory B, the positive facts P, and the negative facts N, then each model M that satisfies the induced theory H also satisfies the background theory B and the positive facts P.

Proof: From the previous section we know that for every theory morphism $\phi : T \to T'$ and a model $M' \models T'$ there is a reduct $\phi M'$ such that $\phi M' \models T$. Let us assume that there exists a model M that satisfies the induced theory H, i.e., $M \models H$. We then have two reducts along the theory morphisms $\phi_B : B \to H$ and $\phi_P : P \to H$, namely $\phi_B M$ and $\phi_P M$, respectively, where

$$\begin{aligned} \phi_B M &\models B, \\ \phi_P M &\models P. \end{aligned}$$

Thus, the models that satisfy the induced theory H have reducts along the theory morphisms and behave as expected. That is, every model that satisfies the induced theory also satisfies the background theory as well as the positive facts. □

Discussion. The key advantage of the algebraic semantics developed here is that it allows us to study the precise meaning of induced theories in terms of the models that satisfy the equations without relying on the notions of a particular operational semantics such as inverse narrowing. Furthermore, the algebraic semantics makes a precise statement on how the induced theory, the background theory, and the positive facts relate to each other via theory morphisms. This is a departure from the set theoretical definitions in the normal ILP semantics.

The key difference between the above approach and the normal semantic setting for ILP given in Definition 1 is the fact that the above approach allows a more general notion of background knowledge in the induced theories. Where the normal semantic setting states that the union of background knowledge and hypothesis needs to entail the positive facts, our approach only requires that the background knowledge is entailed by the induced theory.

On the other hand, similar to the normal semantics, the algebraic semantics defined here also admits trivial solutions. This occurs when both the background theory morphism ϕ_B and the positive facts theory morphism ϕ_P are theory inclusion morphisms. This is analogous to the situation in the normal ILP semantics where the hypothesis is simply the collection of positive facts, i.e.,

$$B \cup P \models p, \text{ for all } p \in P.$$

Also, similarly to the normal ILP semantics the algebraic semantics developed here does not make any statements regarding the quality of the induced theory. That aspect is left to the operational semantics of the induction system.

5 Working an Example

Let us put this machinery to work and look at an example. However, before we can attempt this we need one more thing, we need the soundness and completeness of equational logic [17, 24] in order to dispose of proof obligations using equational deduction.

Theorem 12 (Soundness and Completeness of Equational Logic) Given a set of equations E, an arbitrary equation $(\forall X)t = t'$ is semantically entailed if and only if $(\forall X)t = t'$ is deducible from E. Formally,

$$E \models (\forall X)t = t' \text{ iff } E \vdash (\forall X)t = t'.$$

□

This theorem is very convenient, since it lets us use equational proof theory or deduction to check the theory morphism conditions in the definition of our induced theories.

The following example is due to [14]. Here we assume that the signatures are the obvious constructions and therefore we will only show the set of equations for the underlying theories. Consider the following theories. The background theory B,

$$\begin{array}{lll} (B_1) & \forall x, y & s(x) < s(y) = x < y, \\ (B_2) & \forall y & 0 < s(y) = \textbf{true} \ , \\ (B_3) & \forall x & x < 0 = \textbf{false} \ . \end{array}$$

The positive facts P,

$$\begin{array}{lll} (P_1) & \forall\emptyset & 0 + 0 = 0, \\ (P_2) & \forall\emptyset & s(0) + s(0) = s(s(0)), \\ (P_3) & \forall\emptyset & 0 + s(0) = s(0), \\ (P_4) & \forall\emptyset & s(s(0) + s(0)) = s(s(s(0))), \\ (P_5) & \forall\emptyset & s(s(s(0)) + s(0)) = s(s(s(s(0)))). \end{array}$$

The negative facts N,

$$\begin{array}{lll} (N_1) & \forall\emptyset & s(0) + 0 = 0, \\ (N_2) & \forall\emptyset & 0 + 0 = s(0), \\ (N_3) & \forall\emptyset & s(0) + s(0) = s(0), \\ (N_4) & \forall\emptyset & s(0) + 0 = s(s(0)), \\ (N_5) & \forall\emptyset & s(0 + 0) = s(s(0)), \\ (N_6) & \forall\emptyset & s(s(0) + s(0)) = 0, \\ (N_7) & \forall\emptyset & s(0) = 0. \end{array}$$

Consider the following theory H.

$$\begin{array}{lll}
(H_1) & \forall x, y & s(x) < s(y) = x < y, \\
(H_2) & \forall y & 0 < s(y) = \textbf{true}\ , \\
(H_3) & \forall x & x < 0 = \textbf{false}\ , \\
(H_4) & \forall x, y & x + y = y + x \textbf{ if } x < y, \\
(H_5) & \forall x, y & x + 0 = x, \\
(H_6) & \forall x, y & s(x) + s(y) = s(s(x + y)).
\end{array}$$

The goal is to show that theory H is an induced theory. Therefore, we have to check for three conditions according to Defintion 10. This is done in the following proposition.

Proposition 13 Given the background theory B, the positive facts P, and the negative facts N above, then the theory H above is an **induced theory**.

Proof: Here we assume that the underlying signature morphisms are the obvious mappings. We have to show that the following three conditions hold for theory H:

- The morphism $\phi_B : B \to H$ is a theory morphism;
- the morphism $\phi_P : P \to H$ is a theory morphism;
- and $H \not\models \phi_N(e)$ for all $e \in E_N$ and $\phi_N : \Sigma_N \to \Sigma_H$.

The first condition is easy to check, ϕ_B is simply a theory inclusion morphism. The second condition is readily verified by showing that the equations of the positive facts P hold in H using equational deduction. Equations P_1 and P_2 are easily verified by applying H_5 and H_6, respectively. Equation P_3 can be shown to hold as follows:

$$\begin{array}{lll}
\forall\emptyset\ \ 0 + s(0) = s(0) & & \\
& 0 < s(0) & \text{(evaluating condition of } H_4 \\
& & \text{with } x \mapsto 0 \text{ and } y \mapsto s(0)) \\
& \mathit{true} & \text{(using } H_2) \\
\forall\emptyset\ \ s(0) + 0 = s(0) & & \text{(using } H_4) \\
\forall\emptyset\ \ s(0) = s(0) & & \text{(using } H_5)
\end{array}$$

Also, equation P_4 can be shows to hold as follows:

$$\begin{array}{lll}
\forall\emptyset & s(s(0) + s(0)) = s(s(s(0))) & \\
\forall\emptyset & s(s(s(0 + 0))) = s(s(s(0))) & \text{(using } H_6) \\
\forall\emptyset & s(s(s(0))) = s(s(s(0))) & \text{(using } H_5)
\end{array}$$

Similarly, equation P_5 holds in H. Lastly, there are no such deductions for any of the equations in N. This fulfills our proof obligations Therefore, theory H is an induced theory. □

This result is comforting, since it shows that the theory H is a valid object in both our algebraic semantics as well as in the operational semantics discussed in [14].

6 Declarative Bias

Most ILP systems distinguish between two different kinds of declarative bias [20]: *syntactic* bias and *semantic* bias. This is also true for inductive equational logic programming systems and therefore we will take a brief look at declarative bias in the context of the algebraic semantics developed here.

Traditionally, syntactic bias imposes constraints on the form of the sentences allowed in the hypothesis. That is, hypothesis can only be constructed from a permitted alphabet. In the algebraic setting the form of the equations in the induced theory is defined by the signature of the theory. The kinds of operators and symbols this signature incorporates defines a syntactic bias. In inductive equational logic programming one particular case is typical where the signature of the induced theory, Σ_H, is defined as $\Sigma_H = \phi_B(\Sigma_B) \cup \phi_P(\Sigma_P)$, i.e., the union of the translated signature of the background theory and the translated signature of the positive facts. Here we assume that $\Sigma_N \subseteq \Sigma_P$. In fact, this is also the smallest signature that satisfies our definition of induced theory in Definition 10. It is an open question at this point in the context of inductive equational logic programming if there is a need for larger induced theory signatures due to a process analogous to predicate invention in ILP [18] sometimes referred to as function invention in the equational setting [14]. One could imagine that if one uses a genetic algorithm to discover induced theories the genetic algorithm could be allowed to discover "helper functions" within the induced theories [16].

In contrast to syntactic bias, semantic bias imposes restrictions on the meaning, or behavior of hypotheses [20]. In the algebraic semantics developed here we can identify a semantic bias for inductive equational logic programming. More precisely, equational theories can be interpreted in one of two ways: *strict* via a standard interpretation such as an initial model or *"loosely"* where any model that satisfies the theory will do. These notions find justification in the work on algebraic specifications [5, 7].

7 Conclusions

Using the normal semantic setting for ILP as a starting point, we have set out to develop an algebraic semantics for inductive equational logic programming taking advantage of the close relationship between algebra and equations instead of using a more operational approach by viewing equations as rewrite rules. At the center of this semantics is the notion of an induced theory that is related via theory morphisms to the background knowledge and positive facts. Expressing the relationships between the various theories as theory morphisms allowed us to study the behavior of the algebras satisfying the induced theory in a straight forward algebraic manner. More precisely, any algebra that satisfies the induced theory also satisfies the background theory as well as the positive facts. Intuitively, this is what we would expect and this semantics makes this explicit .

The formalization around theory morphisms also brought to light a more generalized form of background theory inclusion. Where in the standard semantics we are looking at the set theoretic union between the background theory and the hypothesis, in the algebraic framework we are looking at the entailment of the background equations by the induced theory.

We also identified syntactic and semantic biases within the context of inductive equational logic programming. The notion of semantic bias within inductive equational logic programming is a direct result due to our algebraic formulation.

Moving forward there are a number of venues for further research. Firstly, it would be interesting to apply this framework in the context of hidden sorted equational logic [10] where satisfaction no longer needs to hold strictly but behaviorally. Here we only treated the case of many sorted equational logic. Secondly, we would like to apply this framework in the Horn clause logic setting. This would allow us to study traditional ILP within an algebraic setting. More importantly, what is the relationship of the algebraic setting to some of the other semantics that have been developed for ILP such as the non-monotonic semantics [13]? Thirdly, we would like to investigate an inductive equational logic programming system that uses some of the notions developed here as part of its operational semantics for the induction of induced theories.

Acknowledgments. The author would like to thank Stephen Muggleton for many interesting conversations while at the Oxford Computing Laboratory. Also, many thanks to Flaviu Marginean for many comments and pointers. A special thanks go out to my wife Natalie and Enzo for patient

support during the writing of this paper.

A Many Sorted Algebra – A Brief Overview

Many sorted algebra extends the traditional view of an unsorted algebra as a single set with 'structure' by allowing a set of sets or carriers with 'structure' [2, 5, 24]. This algebraic structure is in the form of operations using elements from the carriers as operands and producing elements of the carriers as results. Given a set S of sort names, the set of carriers is sometimes called an *S-sorted set* where each carrier is named by a member of S. More formally:

Definition 14 Given a sort S, an **S-sorted set** A is a collection of sets A_s indexed by elements $s \in S$. All set theoretic operations can be extended to operations on S-sorted sets; for example, if A and B are S-sorted sets, then $A \cup B$ is defined by $(A \cup B)_s = A_s \cup B_s$, and $A \subseteq B$ means that $A_s \subseteq B_s$ for each $s \in S$. Furthermore, an **S-sorted function** $f\colon A \to B$ is a collection of functions indexed by S such that $f_s\colon A_s \to B_s$ for each $s \in S$. $\Box$

Signatures allow one to refer to algebraic structures symbolically, that is, without explicitly referring to the exact underlying structures of any particular algebra. More precisely, signatures provide a uniform, symbolic notation for specifying carriers and operations for algebras. Before we are able to give the formal definition of signatures we need the following:

Notation 15 Let S be a set, then S^* denotes the set of all finite lists of elements from S, including the empty list denoted by $[]$. Given an operation f from S into a set B, $f\colon S \to B$, the operation f^* denotes the extension of f from a single input value to a list of input values, $f^*\colon S^* \to B$, and is defined as follows: $f^*(sw) = f(s)f^*(w)$ and $f^*([]) = []$, where $s \in S$ and $w \in S^*$. $\Box$

We are now in the position to define signatures:

Definition 16 A **many sorted signature** is a pair (S, Σ), where S is a set of sorts and Σ is an $(S^* \times S)$-sorted set of operation names. Thus, if $w \in S^*$ and $s \in S$ then $\Sigma_{w,s}$ is a set of operation names. If Σ is clear from context, we sometimes write $\sigma\colon w \to s$ instead of $\sigma \in \Sigma_{w,s}$ to emphasise the fact that σ denotes an operation mapping the carriers denoted by w to the carrier denoted by s. Usually we abbreviate (S, Σ) to Σ. Elements of $\Sigma_{[],s}$ are referred to as **constants** of sort s.

An operation can be declared to have more than one type; for example we might have $\sigma \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ where w, s is different from w', s'. In this case σ is said to be **overloaded**. $\Box$

For this to make sense, any algebra for a particular signature (S, Σ) should provide a carrier set for each sort name S and should provide an operation for each operation symbol in Σ. Another way of saying this is, that an algebra for a signature Σ interprets the sort names as carrier sets and the operation names as concrete operations or functions between the carriers. We refer to such algebras as Σ-algebras.

Definition 17 Given a many sorted signature Σ, a **Σ-algebra** A consists of the following:

- an S-sorted set, usually denoted A, called the **carrier** of the algebra,
- a **constant** $A_\sigma \in A_s$ for each $s \in S$ and $\sigma \in \Sigma_{[],s}$,
- an **operation** $A_\sigma \colon A_w \to A_s$, for each non-empty list $w = s1 \ldots sn \in S^*$, and each $s \in S$ and $\sigma \in \Sigma_{w,s}$, where $A_w = A_{s1} \times \ldots \times A_{sn}$.

$\Box$

It should be clear from this definition that a signature does not denote only a single algebra but a whole class of algebras where each algebra in this class interprets the sort names and operation names in a particular way. Given a signature Σ, a mapping between two Σ-algebras is called a Σ-*homomorphism*.

Definition 18 Given Σ-algebras A and B, a **Σ-homomorphism** $h \colon A \to B$ is an S-sorted function such that:

- given a constant $\sigma \in \Sigma_{[],s}$, then $h_s(A_\sigma) = B_\sigma$,
- given a non-empty list $w = s1 \ldots sn$ and $\sigma \in \Sigma_{w,s}$ and $ai \in A_{si}$ for $i = 1, \ldots, n$, then

$$h_s(A_\sigma(a1, \ldots, an)) = B_\sigma(h_{s1}(a1), \ldots, h_{sn}(an)).$$

Here, A_σ and B_σ denote the particular interpretations of the operator σ in algebra A and B, respectively. $\Box$

It is easy to see that a Σ-homomorphism is a structure preserving mapping from one Σ-algebra to another.

Given a signature there is a very useful algebra which we may construct whose carriers are sets of terms built up from the given operation names viewed as term constructors.

Definition 19 Given a many sorted signature Σ, the **term algebra** T_Σ is constructed as follows. Let $\cup\Sigma$ be the set of all operation names in Σ, then T_Σ is the least S-sorted set of strings over the alphabet $(\cup\Sigma) \cup \{\underline{(}, \underline{)}\}$ such that for all $s \in S$:

- for each constant symbol $\sigma \in \Sigma_{[],s}$, the string $\sigma \in (T_\Sigma)_s$,
- for each non-empty list $w = s1, \ldots, sn \in S^*$, and each $\sigma \in \Sigma_{w,s}$, and all $ti \in (T_\Sigma)_{si}$ for $i = 1, \ldots, n$, the string $\sigma\underline{(}t1, \ldots, tn\underline{)} \in (T_\Sigma)_s$.

The special symbols '$\underline{(}$' and '$\underline{)}$' are used to emphasise that the carriers of T_Σ are sets of strings; we usually drop the special notation and write '$\sigma(t1, \ldots, tn)$' for '$\sigma\underline{(}t1, \ldots, tn\underline{)}$'.

We give T_Σ the structure of a Σ-algebra by interpreting each operation name in Σ as a term constructor; for all $s \in S$:

- for each constant name $\sigma \in \Sigma_{[],s}$, the constant $(T_\Sigma)_\sigma$ is the string $\sigma \in (T_\Sigma)_s$;
- for each non-empty list $w = s1, \ldots, sn \in S^*$, and each operation name $\sigma \in \Sigma_{w,s}$, the operation $(T_\Sigma)_\sigma \colon (T_\Sigma)_w \to (T_\Sigma)_s$ maps a tuple of strings $t1, \ldots, tn$ to the string $\sigma(t1, \ldots, tn)$.

□

The term algebra T_Σ has the special property of being an *initial* algebra[4].

Definition 20 An **inital Σ-algebra** is a Σ-algebra A such that for each Σ-algebra B there is exactly one Σ-homomorphism $A \to B$. □

Proposition 21 Given a signature Σ, then T_Σ is an initial Σ-algebra. For any Σ-algebra A, the unique Σ-homomorphism $h \colon T_\Sigma \to A$ is defined recursively as follows:

- for each constant symbol $\sigma \in \Sigma_{[],s}$, let $h_s(\sigma) = A_\sigma$,

[4]Here we assume that the signature Σ does not contain any overloaded operations.

- for each non-empty list $w = s1 \dots sn$ and $\sigma \in \Sigma_{w,s}$ and $ti \in (T_\Sigma)_{si}$ for $i = 1, \dots, n$, let

$$h_s(\sigma(t1, \dots, tn)) = (A_\sigma)(h_{s1}(t1), \dots, h_{sn}(tn)).$$

□

The homomorphism h assigns values in A to Σ-terms by interpreting the operation names in Σ as the corresponding operations on A.

Note that so far we have talked about what it means for an algebra to interpret sort and operation symbol names but we have not said anything explicit about the definition of the operations themselves. One way to accomplish this is to give an axiomatisation of the operations. In the case of algebras this axiomatisation is in terms of first order equations. However, before we are able to give the formal definition of equations and what it means for an algebra to satisfy an equation we need to introduce the notion of a variable which serves as a place holder for arbitrary values within a term. Any S-sorted, pairwise disjoint set $X = \{X_s | s \in S\}$ will do as place holder. The set X is usually referred to as a *variable set*. Thus, terms with variables are defined as follows:

Definition 22 Given a many sorted signature (S, Σ) and a variable set $X = \{X_s | s \in S\}$ such that $\Sigma_{[],s} \cap X_s = \emptyset$ for all $s \in S$, **terms with variables from** X are elements in $T_{\Sigma(X)}$, where $\Sigma(X)$ is defined by $\Sigma(X)_{[],s} = \Sigma_{[],s} \cup X_s$ and $\Sigma(X)_{w,s} = \Sigma_{w,s}$ for $w \neq []$. The term algebra $T_{\Sigma(X)}$ can be viewed as a Σ-algebra if we forget about the constants due to the variable set X; we denote this algebra by $T_\Sigma(X)$. □

Since we treat variables as place holders, they only make sense if we are able to assign something concrete to them. More formally this is given by an *assignment*:

Definition 23 Given a Σ-algebra A and an appropriate variable set X, an S-sorted function $\theta \colon X \to A$ is called an **assignment**. The assignment extends to the unique homomorphism $\overline{\theta} \colon T_\Sigma(X) \to A$ such that $\overline{\theta}(x) = \theta(x)$ for all $x \in X$. □

Now we are ready to define equations:

Definition 24 A **Σ-equation** is a triple (X, l, r) where X is an appropriate variable set and $l, r \in T_\Sigma(X)_s$ for some sort $s \in S$. If $X = \emptyset$, that is, l and

r contain no variables, then we say the equation is **ground**. We write equations in the form $(\forall X)\, l = r$.

A **specification** or **theory**[5] is a pair (Σ, E) where Σ is a many sorted signature and E is a set of Σ-equations. $\Box$

The models of a theory are the Σ-algebras that satisfy the equations, that is, a theory denotes a whole class of algebras, namely those algebras which satisfy its equations. Intuitively, an algebra satisfies an equation iff the left and right sides of the equation are equal under all assignments of the variables. More formally:

Definition 25 A Σ-algebra A **satisfies** a Σ-equation $(\forall X)\, l = r$ iff $\overline{\theta}(l) = \overline{\theta}(r)$ for all assignments $\theta\colon\ X \to A$. We write $A \models e$ to indicate that A satisfies the equation e. For a set E of equations we write $A \models E$ iff $A \models e$ for each $e \in E$. We write $E \models e$ iff $A \models E$ implies $A \models e$ for all Σ-algebras A.

Given a theory (Σ, E), a (Σ, E)**-model** is a Σ-algebra A such that $A \models E$. $\Box$

Just as each signature has an initial algebra, each specification has an initial model. The initial model is constructed from the term algebra by identifying exactly those terms that are 'equal' as a consequence of the given equations in the specification. Each equation in a specification gives rise to an equivalence relation in the following way:

Definition 26 Given a Σ-algebra A and a Σ-equation e of the form $(\forall X)\ l = r$ which we assume to be satisfied by A, then define the equivalence relation $\sim_{A,e}$ on A by $a \sim_{A,e} b$ iff $a = \overline{\theta}(l)$ and $b = \overline{\theta}(r)$ for some $\theta\colon\ X \to A$. $\Box$

That $\sim_{A,e}$ is an equivalence relation follows from the fact that the equation generating the relation is reflexive, symmetric, and transitive. We will use this in defining an equivalence relation that contains all the equivalence relations derived from the equations of a specification, and that allows the substitution of equals for equals. This is formalised by the notion of a *congruence*:

Definition 27 Given a Σ-algebra A, a Σ**-congruence** on A is an S-sorted equivalence relation $\sim$ such that the following **substitutivity property**

[5]We treat the terms *specification* and *theory* as synonymous; the traditional notion of theory can be recovered by taking the closure of the set of equations.

holds: for all $\sigma \in \Sigma_{w,s}$ where $w = s1, \ldots, sn$ and $ai, bi \in (A)_{si}$ for $i = 1, \ldots, n$, if $a1 \sim b1, \ldots, an \sim bn$ then $A_\sigma(a1, \ldots, an) \sim A_\sigma(b1, \ldots, bn)$.

If E is a set of Σ-equations and A is a Σ-algebra, then $\equiv_{A,E}$ denotes the least Σ-congruence on A which contains each equation in E; that is, such that $\sim_{A,e} \subseteq \equiv_{A,E}$ for each $e \in E$. We usually write $\equiv_E$ instead of $\equiv_{A,E}$. $\Box$

Definition 28 Given a Σ-algebra A and a Σ-congruence $\equiv$ on A, we construct the Σ-algebra $A/\equiv$, called the **quotient of A by $\equiv$**, as follows:

- for $s \in S$, let $(A/\equiv)_s = \{[a] | a \in A_s\}$, where $[a]$ is the equivalence class of a under $\equiv$,
- for each constant symbol $\sigma \in \Sigma_{[],s}$, let $(A/\equiv)_\sigma = [A_\sigma]$,
- for each non-empty list $w = s1, \ldots, sn$, $\sigma \in \Sigma_{w,s}$, and $[ai] \in (A/\equiv)_{si}$ for $i = 1, \ldots, n$, let

$$(A/\equiv)_\sigma([a1], \ldots, [an]) = [(A/\equiv)_\sigma(a1, \ldots, an)].$$

The last equation is well-defined by the substitutivity property of the congruence $\equiv$. $\Box$

This machinery finally enables us to state what the initial model of a specification is, it is namely the *quotient term algebra* generated by the equations of the specification. More precisely:

Proposition 29 Given a specification (Σ, E), the initial (Σ, E)-model is the **quotient term algebra** $T_{\Sigma,E}$, which is the quotient $T_\Sigma/\equiv_E$. By construction, $T_{\Sigma,E}$ satisfies the equations E. $\Box$

There is a very useful and powerful extension to the usual equations studied so far; the *conditional equations*. Conditional equations allow one to concisely state the conditions under which a particular equation holds. More formally:

Definition 30 A **conditional Σ-equation** is a quadruple (X, l, r, C) where X is an appropriate variable set, $l, r \in T_\Sigma(X)_s$ for some sort $s \in S$, and C is a set of pairs (u, v) where $u, v \in T_\Sigma(X)_s$ for $s \in S$. We usually write conditional equations in the form

$$(\forall X)\ l = r\ \texttt{if}\ C.$$

Furthermore, given a Σ-algebra A, we define

$$A \models_\Sigma (\forall X)\ l = r \ \texttt{if}\ C$$

to mean that, given any assignment $\theta\colon\ X \to A$, if $\overline{\theta}(u) = \overline{\theta}(v)$ for all $(u, v) \in C$, then $\overline{\theta}(l) = \overline{\theta}(r)$; in other words the Σ-algebra A **satisfies** the conditional equation. $\Box$

It is interesting to note that conditional equations do not add to the mathematical power of a specification language. On the other hand, the expressiveness of a specification language is dramatically enhanced by embracing conditional equations. There is a useful relationship between conditional and ordinary unconditional equations, namely:

Fact 31 Given a Σ-equation $e = (\forall X)\ t = t'$, let $e' = (\forall X)\ t = t' \ \texttt{if}\ \emptyset$ for $t, t' \in T_\Sigma(X)$, then for each Σ-algebra A, $A \models e$ iff $A \models e'$. $\Box$

This is interesting in the sense that we may regard ordinary equations as conditional equations with the empty condition. In the following sections and chapters we will not distinguish between conditional and unconditional equations unless a distinction is necessitated by the circumstances.

There is another very useful result which gives us a technique for proving conditional equations.

Proposition 32 Given a conditional Σ-equation $(\forall X)\ t = t' \ \texttt{if}\ C$ and a set E of Σ-equations, then

$$E \models_\Sigma\ (\forall X)\ t = t' \ \texttt{if}\ C \text{ iff } (C \cup E) \models_{\Sigma(X)}\ (\forall \emptyset)\ t = t'.$$

$\Box$

Since signatures denote algebras, it is interesting to see how one algebraic structure can be viewed in terms of another, or more precisely, how one signature can be translated into another signature. The appropriate mapping here is the *signature morphism.*

Definition 33 A **signature morphism** $\phi\colon\ (S, \Sigma) \to (S', \Sigma')$ is a pair (f, g), where $f:\ S \to S'$ maps sorts in S to sorts in S', and g is a collection of functions indexed by $S^* \times S$ such that $g_{w,s}:\ \Sigma_{w,s} \to \Sigma'_{f^*(w),f(s)}$ for each $(w, s) \in S^* \times S$. We usually write ϕ instead of both f and $g_{w,s}$, so that if $\sigma \in \Sigma_{w,s}$ then $\phi(\sigma) \in \Sigma'_{\phi^*(w),\phi(s)}$. $\Box$

Given a signature morphism $\phi\colon\ \Sigma \to \Sigma'$, it is possible to view any Σ'-algebra A' as a Σ-algebra by forgetting any additional structure Σ' might have. This is formalised with the notion of a *reduct*:

Definition 34 Given a signature morphism $\phi\colon\ \Sigma \to \Sigma'$ and a Σ'-algebra A', we define the Σ-algebra $\phi A'$, called the **reduct** of A' along ϕ, by setting $\phi A'_s = A'_{\phi(s)}$ for $s \in S$ and $\phi A'_\sigma = A'_{\phi(\sigma)}$ for $\sigma \in \Sigma_{w,s}$. $\square$

Given this we should look at the more interesting case of theories. We define a *theory morphism* as an appropriate mapping from one theory to another. Note that we assume that signature morphisms extend to equations. More precisely, a given signature morphism $\phi\colon\ \Sigma \to \Sigma'$ extends to the family of mappings $\phi_s : T_\Sigma(X)_s \to T_{\Sigma'}(\phi(X))_{\phi(s)}$ where X is an appropriate variable set and $\phi(X)_{s'} = \bigcup_{\phi(s)=s'} X_s$. Now, given a Σ-equation $e = (X, l, r)$ where $l, r \in T_\Sigma(X)$, then $\phi(e) = (\phi(X), \phi(l), \phi(r))$ is a Σ'-equation where $\phi(l), \phi(r) \in T_{\Sigma'}(\phi(X))$. We are now ready to define theory morphisms formally:

Definition 35 Given two many sorted theories $Th = (\Sigma, E)$ and $Th' = (\Sigma', E')$, then the **theory morphism** $\phi\colon\ Th \to Th'$ is a signature morphism $\phi\colon\ \Sigma \to \Sigma'$ such that $E' \models_{\Sigma'} \phi(e)$ for all $e \in E$. $\square$

Goguen and Burstall have shown within the framework of institutions [8] that the following holds for many sorted algebra:

Theorem 36 Given a theory morphism $\phi\colon\ (\Sigma, E) \to (\Sigma', E')$ and a (Σ', E')-algebra A', then

$$A' \models_{\Sigma'} \phi(e) \Leftrightarrow \phi A' \models_\Sigma e$$

for all $e \in E$. $\square$

Therefore, if we can show that a given model of the target theory satisfies the translated equations of the source theory, it follows that the reduct of this model also satisfies the source theory, thus, the models behave as expected.

References

[1] R. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery*, 24(1):44–67, January 1977.

[2] Paul M. Cohn. *Universal Algebra.* Harper and Row, 1965. Revised edition 1980.

[3] N. Dershowitz and E. Pinchover. Inductive synthesis of equational programs. In *Proceedings, Eighth National Conference on Artificial Intelligence*, pages 234–239. MIT Press, 1990.

[4] Hartmut Ehrig, Werner Fey, and Horst Hansen. ACT ONE: An algebraic specification language with two levels of semantics. Technical Report 83–03, Technical University of Berlin, Fachbereich Informatik, 1983.

[5] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* Springer, 1985. EATCS Monographs on Theoretical Computer Science, Volume 6.

[6] Peter A. Flach. The logic of learning: a brief introduction to Inductive Logic Programming. In *Proceedings of the CompulogNet Area Meeting on Computational Logic and Machine Learning*, pages 1–17, 1998.

[7] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24:68–95, 1977.

[8] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, January 1992. Draft appears as Report ECS-LFCS-90-106, Computer Science Department, University of Edinburgh, January 1990; an early ancestor is "Introducing Institutions," in *Proceedings, Logics of Programming Workshop*, Edward Clarke and Dexter Kozen, Eds., Springer Lecture Notes in Computer Science, Volume 164, pages 221–256, 1984.

[9] Joseph Goguen, Claude Kirchner, Hélène Kirchner, Aristide Mégrelis, and José Meseguer. An introduction to OBJ3. In Jean-Pierre Jouannaud and Stephane Kaplan, editors, *Proceedings, Conference on Conditional Term Rewriting*, pages 258–263. Springer, 1988. Lecture Notes in Computer Science, Volume 308.

[10] Joseph Goguen, Grant Malcolm, and Tom Kemp. A hidden herbrand theorem: Combining the object, logic and functional paradigms. In *Proceedings Programming Language Implementation and Logic Program-*

ming / Algebraic and Logic Programming, Lecture Notes in Computer Science 1490, pages 445–462. Springer-Verlag, 1998.

[11] Lutz H. Hamel. UCG-E: An equational logic programming system. In *Proceedings of the Programming Language Implementation and Logic Programming Symposium 1992*, Lecture Notes in Computer Science 631. Springer-Verlag, 1992.

[12] D. Heckerman, D. Geiger, and D. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[13] N. Helft. Induction as nonmonotonic inference. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 149–156. Morgan Kaufmann, 1989.

[14] J. Hernandez-Orallo and M. Ramirez-Quintana. Induction of functional logic programs. In Lloyd, editor, *Proceedings of CompulogNet Meeting on Computational Logic and Machine Learning*, pages 49–55, 1998.

[15] J. Hernandez-Orallo and M. Ramrez-Quintana. A strong complete schema for inductive functional logic programming. In *Inductive Logic Programming'99*, Lecture Notes in Artificial Intelligence 1634, pages 116–127. Springer-Verlag, 1999.

[16] J. Koza. *Genetic Programming: On Programming Computers by Means of Natural Selection and Genetics.* MIT Press, Cambridge, MA, 1992.

[17] José Meseguer and Joseph Goguen. Initiality, induction and computability. In Maurice Nivat and John Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge, 1985.

[18] S. Muggleton. Predicate invention and utility. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):121–130, 1994.

[19] S. Muggleton. Scientific knowledge discovery using Inductive Logic Programming. *Communications of the ACM*, 42(11):42–46, November 1999.

[20] S. Muggleton and L. de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:669–679, 1994.

[21] J. Quinlan. C4.5. In *Programs for Machine Learning.* Morgan Kaufman, 1997.

[22] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub. Interpreting patterns of gene expression with self-organizing maps. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 96, pages 2907–2912, 1999.

[23] Maartin H. van Emden and Robert Kowalski. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery*, 23(4):733–742, 1976.

[24] Wolfgang Wechler. *Universal Algebra for Computer Scientists.* Springer-Verlag, 1992. EATCS Monographs on Theoretical Computer Science, Volume 25.