Category-based Semantics for Equational and Constraint Logic Programming

by

Răzvan Diaconescu

Saint Anne's College

July 1994

Submitted in partial fulfillment of the requirements for the Doctor of Philosophy in Computation

Oxford University Computing Laboratory
Programming Research Group

Abstract

This thesis proposes a general framework for equational logic programming, called *category-based equational logic* by placing the general principles underlying the design of the programming language Eqlog and formulated by Goguen and Meseguer into an abstract form. This framework generalises equational deduction to an arbitrary category satisfying certain natural conditions; completeness is proved under a hypothesis of quantifier projectivity, using a semantic treatment that regards quantifiers as models rather than variables, and regards valuations as model morphisms rather than functions. This is used as a basis for a model theoretic category-based approach to a paramodulation-based operational semantics for equational logic programming languages.

Category-based equational logic in conjunction with the theory of institutions is used to give mathematical foundations for modularisation in equational logic programming. We study the soundness and completeness problem for module imports in the context of a category-based semantics for solutions to equational logic programming queries.

Constraint logic programming is integrated into the equational logic programming paradigm by showing that constraint logics are a particular case of category-based equational logic. This follows the methodology of free expansions of models for built-ins along signature inclusions as sketched by Goguen and Meseguer in their papers on Eqlog. The mathematical foundations of constraint logic programming are based on a Herbrand Theorem for constraint logics; this is obtained as an instance of a more general category-based version of Herbrand's Theorem.

The results in this thesis apply to equational and constraint logic programming languages that are based on a variety of equational logical systems including many and order sorted equational logics, Horn clause logic, equational logic modulo a theory, constraint logics, and more, as well as any possible combination between them. More importantly, this thesis gives the possibility for developing the equational logic (programming) paradigm over non-conventional structures and thus significantly extending it beyond its tradition.

Acknowledgments

I would like to express my warmest thanks towards my supervisor, Professor Joseph Goguen, who supported this work in a very special way ranging from moral encouragement to direct technical guidance. His presence can be felt everywhere in this work. Most importantly, I would like to thank Joseph for offering me his wonderful friendship which words cannot describe.

Special thanks go to Professor Burstall and Dr Meseguer for their contribution to Computing Science, contribution that made this work possible. Professor Căzănescu and Dr Stefănescu unveiled to me the elegant world of, what I would call now, the *Goguen culture in Computing* years before starting my doctoral programme in Oxford. That was in fact the real beginning of this thesis. With Rod Burstall, Tom Kemp, Hendrik Hilberdink, and Virgil Căzănescu I had the opportunity to discuss various aspects of my work at different stages. I also owe a lot to Jocelyn Paine for sharing with me his experience with conventional logic programming. The final version of the thesis benefited greatly in terms of presentation from the constructive criticism of my examiners, Dr Lincoln Wallen and Professor Jean-Pierre Jouannaud.

I would like to make an acknowledgement to all those who made my life in England easier. They include my PRG colleagues Hendrik, Petros, José, Paulo, Lutz, Francisco, Jason, as well as Aslan, Gianluca, Bart, Pero, Pavle with whom I shared the discomfort of Oxford accommodations at different stages of my stay here. Joseph Goguen's research group was like a family for me, always ready to offer their precious help. All these years I felt especially close to my colleagues Hendrik Hilberdink and Petros Stefaneas; we shared many hopes, and above everything, a belief in a certain life style.

My friendship with Sandu Mateescu, cemented in the Carpathians, was of the greatest help during the last couple of years. I would also like to thank my parents, Stefan and Elena, for their love and patience.

Finally, I gratefully acknowledge the financial support from British Telecom, Fujitsu Laboratories Limited, MITI[1], International Computers Limited, Sharp, UK Government[2], Oxford University, British Council in Bucureşti, and St. Anne's College Oxford.

[1]The Information Technology Promotion Agency, Japan, as part of the R & D of Basic Technology for Future Industries "New Models for Software Architecture" project sponsored by NEDO (New Energy and Industrial Technology Development Organization).

[2]Through the ORS scheme.

1 Introduction

This thesis is mainly about equational logic programming. It belongs to the tradition of equational and constraint logic programming started by Goguen and Meseguer in their pioneering work on the programming language Eqlog during the mid-eighties [38, 39]. Eqlog has been implemented in Oxford by the author of this thesis as an extension of the SRI implementation of OBJ3.[3]

1.1 The Equational Logic Programming Paradigm

1.1.1 A historical perspective

Equational logic programming can be regarded as joining two major cultures in Computing: algebraic specification and logic programming.

Logic programming began in the early 1970's as a direct outgrowth of earlier work in automatic theorem proving and artificial intelligence. The theory of clausal-form [first order] logic, and an important theorem by the logician Jacques Herbrand constituted the foundation for most activity in theorem proving in the early 1960's. The discovery of resolution — a major step in the mechanization of clausal-form theorem proving — was due to J. Alan Robinson [81]. In 1972, Robert Kowalski and Alain Colmerauer were led to the crucial idea that *logic could be used as a programming language* [95]. A year later the first Prolog system was implemented. SLD-resolution, which is a refinement of the resolution principle restricted to Horn clause logic, became the core of the operational semantics for most of the further logic programming implementations, although logic programming is by no means limited to Prolog.

One of the main slogans of logic programming, due to Kowalski, is

$$\mathsf{Program} = \mathsf{Logic} + \mathsf{Control}$$

meaning that a problem has a declarative side asserting *what* the problem is and what properties solutions should have, as well as a control side describing *how* the problem is to be solved. The ideal of declarative programming in general, and of logic programming in particular, is that the user should specify the logic component of the problem, and control should be exercised as much as possible by the programming system. Unfortunately, the users of Prolog-like systems still need to supply a lot of control information.

During the 1980's, the constraint programming paradigm gradually grew out of logic programming (see [58]). This brought a new perspective on logic programming, in which the concept of unification is generalised to the concept of constraint solving [16, 15]. However, Lassez showed that constraint logic programming is still part of the logic programming paradigm in a fundamental way[4] [66].

[3] Appendix A gives a brief description of how to use the Eqlog system; some examples of Eqlog runs are given in Chapter 4.

[4] In Chapter 6 we show how generalised constraint logic programming can be foundationally regarded as a particular case of equational logic programming.

Algebraic specification is now a particularly mature field of Computing Science, because of its strong and stable mathematical foundations. The theory of algebraic specification has been implemented in many computing systems, and is also an important technique in Software Engineering methodologies.

While the insight that operations should be associated with data representations seems to have been due to David Parnas [77], the legendary group ADJ[5] made a decisive step forward by using initiality (a category-theoretic concept) as a characterisation for the notion of standard model [44]. Many sorted equational logic became the main logical system underlying the theory of algebraic specifications and abstract data types. It was proved complete by Goguen and Meseguer before mid 1980's [37], but because of its inability to handle erorrs, it was replaced by order sorted equational logic (which is many sorted equational logic with subtyping [41]) as the modern logical system underlying the theory of algebraic specifications and abstract data types.

The theory of algebraic specifications entered a completely new era with the discovery of the theory of institutions by Goguen and Burstall [33], transcending its origins in equational logic to encompass a wide variety of logic systems, including first order logic, Horn clause logic, higher order logic, infinitary logic, dynamic logic, intuitionistic logic, order sorted logic, temporal logic, etc. Today, nearly 15 years after the first insights given by the work on the specification language Clear [13], the spirit of abstract model theory (in its institutional form) is a significant part of the culture of algebraic specification.

The language OBJ [46] played a major rôle in the development of algebraic specification and, more generally, of declarative programming. It began as an algebraic specification language at UCLA about 1976, and has been further developed at SRI International and several other sites as a declarative specification and rapid prototyping language. Its mathematical semantics is given by order sorted equational logic, and it has a powerful type system featuring subtypes and overloading. In addition, OBJ has user definable abstract data types with user-definable mixfix syntax, and a powerful parameterised module facility that includes views and module expressions. A subset of OBJ is executable by order sorted rewriting. OBJ has been extended towards object-oriented programming (the language FOOPS [40]), theorem proving (the metalogical framework theorem prover 2OBJ [42]) and logic programming (the language Eqlog [38], which is also further discussed in this thesis).

1.1.2 Equational logic programming

The equational logic programming paradigm unifies logic programming based on Horn clause logic and equational (i.e., functional) programming based on equational logic, i.e., the logic of substituting equals for equals. One of the earliest contributions to this field was [76]. As Goguen and Meseguer repeatedly pointed out [38, 39], the best way to achieve this goal should be to unify the two logics involved. However, because equational logic is more fundamental than Horn clause logic[6], it is enough to base the new paradigm only on equational logic. The main difference between equational logic programming and equational programming lies in the fact that the former deals with the problem of *solving queries*. This implies the (somehow subtle) involvement of existentially quantified sentences, which is explained by Herbrand's Theorem.

[5] Originally Goguen, Thatcher, Wagner and Wright.

[6] This will be explained in detail in Section 2.3.3.

Such a combination is desirable for both the algebraic specification and the logic programming traditions. The query solving capability extends equational programming to a very powerful paradigm in which a specification is already a program (or at least it is very close to being a program). This not only enormously simplifies the correctness-verification problem, but also brings in all the advantages of algebraic specification languages (clarity, simplicity, reusability, maintainability, etc).

From the logic programming point of view, this is the best way to integrate [semantic] equality into logic programming; a major problem with relational programming, because many of the compromises of the logic programming ideal found in actual languages (e.g., Prolog) have to do with the inability of relational programming to cope with equality. These compromises created a gap between the original vision of logic programming (i.e., *programming in logic*) and most of the actual implementations which are far from having a logic-based semantics. In general, they tend in the direction of imperative programming, which can be confusing and inefficient [2]. (The argument is that the denotational semantics of imperative programs is complex and complicated, with the ultimate practical consequence being that the debugging is very hard.)

As Goguen and Meseguer pointed out in the context of the programming language Eqlog [38], the equational logic programming paradigm provides as much practical programming power as possible without compromising the underlying logic. In fact, equational logic programming seems to match very well the slogan of *logical programming* (i.e., programming rigorously based on a logical system) as formulated in [39]:

> Computation is deduction in the underlying institution.

Any of the advantages of Eqlog over Prolog can be regarded as a direct consequence of its semantical purity, which sharply contrasts with the many extralogical features of Prolog[7]. Although "cut" may be the most notorious, "is" is probably the most outrageous, since it is an assignment statement with declarative syntax. Thus, real Prolog programs can be far from having a simple foundation in Horn clause logic. Constraint logic programming is implemented by PrologIII in a fixed rather than extensible way, while Eqlog is enough flexible to be considered as a *framework* for constraint logic programming [39, 38]. This means that Eqlog supports constraint solving over any user defined data type. For this reason we call Eqlog an **extensible**[8] constraint programming language.

In general, the operational semantics of equational logic programming systems is based on narrowing (which is similar to the resolution used in logic programming). Different rather sophisticated refinements of narrowing can be in practice as efficient as Prolog's SLD-resolution, which can even be regarded as a special case of narrowing by viewing the relation symbols as operations (i.e., functions). Therefore narrowing already contains the mixture of resolution and narrowing that occurs in the context of the operational semantics of equational logic programming languages based on Horn clause logic with equality.

1.2 Contributions of this Thesis

This thesis develops a category-based semantics for equational and constraint logic programming in the style of the language Eqlog, by placing the general principles underlying

[7] A major advantage of Prolog is its good compiler.

[8] In [39] Goguen and Meseguer use the terminology "generalised" instead of "extensible".

the design of the programming language Eqlog and formulated by Goguen and Meseguer in [38, 39] into an abstract form. The actual implementation of Eqlog is faithful to this semantics, and experimentations with the system helped the development of the theory.[9]

The category-based framework of this thesis gives the possibility to develop equational logic (programming) over non-conventional structures. In this way, equational logic programming is liberated from the traditional set theoretic point of view. This is similar to the way functional programming and algebraic specification got their true meaning and power with cartesian closed categories and institutions, respectively. The development of equational logic programming over different types of models and domains (some of them could have a much richer structure than the usual set theoretic domains) and might prove very beneficial in terms of unifying equational logic programming with other programming paradigms. By following the results of this thesis one can easily develop the equational logic (programming) over continuous lattices insead of sets and functions, for example. Although the examples we provide in this thesis don't depart fundamentally from the tradition of equational logic programming as it is today, this framework proved already to be very effective in integrating equational logic programming with constraint programming (see Chapter 6).

1.2.1 Beyond conventional "abstract model theory"

The framework underlying this thesis can be characterized as *abstract model theory* in the same spirit as the work by the "Hungarian School" in late seventies,[10] for example, is characterized as abstract model theory. By abstract model theory (abbreviated *AMT*) we mean far more than the respective tradition in logic which abstracts the Tarskian approach to cover other logical systems[11] (e.g., [6, 5]). Our category-based framework is very close in spirit to the theory of institutions [33] in the sense that

- it abstracts Tarski's classic semantic definition of truth [93], based on a relation of satisfaction between models and sentences, and
- it uses category theory in a very similar manner to achieve generality and simplicity; in both approaches the models have the abstract structure of a category.

In fact, the theory of institutions was a great source of inspiration for our framework; we view that theory as fulfilling the original vision of abstract model theory. Two main differences between our approach and the theory of institutions are:

- the concept of satisfaction between models and sentences is significantly less abstract in our approach because, although the models are fully abstracted and the sentences generalise the traditional notions of equation, the actual satisfaction relation is defined in a way that abstracts exactly the traditional equational logic satisfaction between algebras and equations, rather than being an undefined primitive as in the theory of institutions; and
- our framework does not contain a *direct* mathematical formulation of the intuition that "truth is invariant under change of notation," which is somehow central for the theory of institutions.

[9] In fact, all code presented as examples in the thesis has been run under the Eqlog system.

[10] [1] is a representative piece of work of this school.

[11] The goal of research in this area being to generalise as much of classical first order model theory as possible.

The second point addresses the problem of the technical relationship between our category-based framework and the theory of institutions. Chapter 5[12] shows that our framework can be naturally embedded into the theory of institutions. On the other hand, our category-based framework can be internalised in any *many sorted liberal institution*.[13]

1.2.2 Category-based equational logic

One of the main contributions of this thesis is to propose a general framework for the equational logic programming paradigm called **category-based equational logic** which distills the essential ingredients characterising equational logics. Equations, equational deduction, models (algebras), congruences, satisfaction, etc. are treated in an arbitrary category satisfying certain mild conditions which plays the rôle of the category of *models* for the equational logical system. This category of models comes equipped with a forgetful functor to an [abstract] category of domains. This encodes the principle that any model is an interpretation of a signature[14] into a domain which is usually a set, or a collection of sets in the case of typed logical systems. All concepts are introduced and results are proved at the highest appropriate level of abstraction. Through a gradual refinement process (which could be seen as "climbing down" the abstraction hierarchy) all concepts (including the rules of inference for category-based equational deduction) can be made explicit in the concrete cases, while still avoiding all irrelevant details when focusing on a particular equational logical system. By taking a semantic perspective on terms as elements of a carrier of a free model,[15] the quantification of equations is abstracted from variables to models, as a result, valuations are abstracted from simple assignments of the variables to model morphisms.

The framework of category-based equational logic is used in this thesis to deal with operational semantics, modularisation and constraint programming for the equational logic programming paradigm. Such a framework must achieve a delicate balance between abstraction and concreteness; this balance makes possible the natural encoding of all important principles and phenomena related *exactly* to the equational logic programming paradigm, while still avoiding the details of any particular logical system. This explains why the category-based framework of this thesis technically lives on a lower level of abstraction than the theory of institutions which was designed to be used in the wider context of declarative programming. The analogy with classical algebra might be enlightening. Although the mathematical structure underlying modern algebra is that of a *ring*, the structure of *module*[16] is more important for the more specialised area of linear algebra. However, there is a close relationship (both technically and in spirit) between rings and modules, although rings may also be fundamental for number theory, which is only indirectly related to linear algebra. In the same way, institutions may be relevant to an area only remotely related to equational logic programming, such as semantics for the object paradigm [29, 11, 35].

A uniform treatment of the model theory of classical equational logic is now possible due to the comprehensive development of categorical universal algebra; without any claim of completeness, I mention the so-called Lawvere algebraic theories, either in classical form [69] or in monadic form [70] (although neither of these fits order sorted algebra

[12] Devoted to modularisation issues.
[13] The precise definition is given in Chapter 5.
[14] Sometimes called language or vocabulary in classical logic textbooks.
[15] As opposed to to the syntactic perspective that regards terms as tree-like syntactic constructs.
[16] Not to be confused to the Computing concept of module!

nicely), the theory of sketches [4], and the recently developed theory of "abstract algebraic institutions" [89, 91]. However, no uniform proof theory has previously been developed for all these equational logics. It could be argued that, at least for computation, the proof theory is more important than the model theory. In Computing Science model theory is far more important as a methodology or style of thinking than it is in itself. A major contribution of this thesis is that it lays bare the *architecture* of equational deduction, i.e., the conceptual structure that underlies it. The key to the completeness of category-based equational deduction is to regard the congruence determined on an arbitrary model A by an arbitrary collection Γ of [conditional] equations in two different ways: as the collection of all unconditional equations quantified by A that are syntactically inferred from Γ, and as the collection of equations that are a semantic consequence of Γ. Because of the semantic treatment of equation and satisfaction, there is no distinction between the congruence determined by Γ on the free models and on other models. Under some additional conditions related to the finiteness of the hypotheses of the conditions in Γ and to the finiteness of the model operations (both of them encoded in category-theoretic terms), this congruence can be obtained in an effective way.

A relevant consequence of the completeness results for category-based equational deduction is a generic Herbrand's Theorem (in two versions) formulated in the style of [39], i.e., characterising Herbrand models as initial models of the program regarded as an equational theory. This provides mathematical foundations for the equational logic programming paradigm in the style of Eqlog [38, 39]. When applied to constraint logics (in Chapter 6), this gives a version of Herbrand's Theorem for extensible constraint logic programming. Despite the sophistication of this last result, it is obtained with minimal effort due to the category-based machinery.

1.2.3 Category-based operational semantics

Equational deduction bridges the gap between the operational semantics and the model theory of equational logic programming; such a reconciliation is essential for understanding the correctness of computer implementations. The completeness and soundness of a computing system rigorously based on some equational logic depends on the completeness and soundness of the operational semantics with respect to the deduction system of the equational logic involved, as well as on the completeness and soundness of the equational deduction system with respect to its model theory.

Our category-based framework supports the development of category-based equational logic into a purely model theoretic approach to the completeness of operational semantics for various programming paradigms that are based on some form of equational logic; this result is independent of the particular [equational] logic involved, as opposed to the combinatorial treatments of the paramodulation-based operational semantics seen in the literature. We generalise the concept of paramodulation to **model theoretic paramodulation** by defining paramodulation as an inference rule with respect to an arbitrary fixed model. We propose a generic scheme for proving the completeness of the paramodulation-based operational semantics for equational logic programming. The core of this scheme is the analysis of the relationship between the congruence determined by a program Γ on a model A and the relations induced on A by the operational inference rules. This scheme also clarifies the rôle played by the Theorem of Constants, the Completeness of Equational Logic, and the Lifting Lemmas in proving the completeness of operational semantics. In this approach rewriting is defined on algebraic entities that are

more abstract than terms. This is achieved by isolating the abstract properties of what are known *contexts* in the standard case of many sorted algebra.

An important class of applications concerns equational deduction *modulo a theory*. This arises when some equations in a program are non-orientable, making them useless as rules (i.e., for rewriting or narrowing). The most notorious cases are associativity (A), commutativity (C) and their combination (AC). Also, graph rewriting is a particular case of rewriting modulo a theory [7]. By taking a semantic perspective on computations modulo a theory we introduce the more abstract concept of **paramodulation modulo a model morphism** and use it for showing that computing in the quotient model of a theory is the same as computing modulo that theory. One conclusion of this thesis is that there is no fundamental difference between ordinary equational deduction and equational deduction modulo a theory. Based on this level of denotational semantics, Chapter 4 extends this conclusion to the realm of operational semantics.

1.2.4 Modularisation and extensible constraint logic programming

By integrating our framework and the theory of institutions, we define the mathematical structure underlying modularisation for equational logic programming in the style of OBJ: the institution of category-based equational logic supports a general treatment of the modularisation issues that are specific to the equational logic programming paradigm, as well as a category-based semantics for queries and solution forms in the context of OBJ-like modularisation.

In the institution of category-based equational logic, the signatures are functors. This abstraction of the notion of signature is based upon the fact that in any equational logic system, a signature determines a category of models, a category of domains, and a forgetful functor between them. A morphism of signatures consists of a pair of "reduct" functors, one on models and the other on domains. Forgetting from models to domains commutes with the reduct functors.

The concept of solution form is shown to correspond to the Satisfaction Relation in a special "non-logical" institution. This correspondence is useful for understanding the soundness and completeness problem for equational logic programming module importation in the wider context of institution theory, and thus relating it to the usual logical concepts of soundness and completeness.[17] The solution to this problem is given at the level of the institution of category-based equational logic.

Finally, the category-based machinery is used for integrating extensible constraint logic programming into the equational logic programming paradigm by defining its underlying logic and regarding it as category-based equational logic in which the models form a comma category over a "built-in" model. This idea is based on the insight of [39] to use free expansions of models of built-ins along signature morphisms. This represents a significant generalisation of the initial algebra approach from abstract data types to constraint solving. In this way extensible constraint logic programming becomes a paradigm based essentially on equational logic.[18] This is a big advantage because extensible constraint logic programming could benefit from the high maturity of the semantics of equational logic, and possibly from some implementation techniques specific to equational logic. At the semantic level, this is already very transparent. It will be interesting

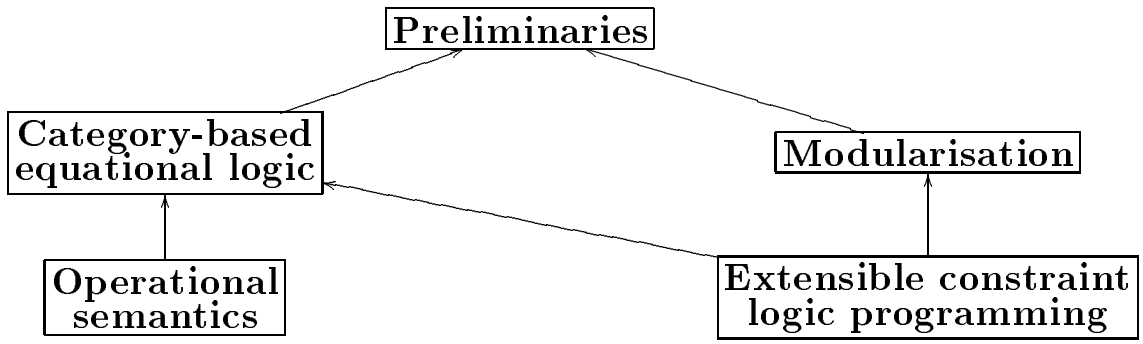
[17] This is an example of the use of "abstract model theory" beyond the realm of logical systems, and of extension of concepts from logic to different areas.

[18] More precisely, on category-based equational logics.

to explore the benefits of such an approach at the level of operational semantics.

1.3 The Structure of the Thesis

After the Introduction and Preliminaries, we devote one chapter to each of the four main topics. The technical dependencies between chapters are shown in the following diagram:



1.3.1 Preliminaries

The basic categorical concepts of this work are introduced and the category-based framework of this thesis is introduced. The first section is devoted to various aspects of *categorical relations*, which are at the center of the categorical machinery of this thesis. The second section discusses finiteness from a categorical angle and applies it to categorical relations. Equivalence, composition of binary relations, closures of relations and confluent relations are analysed within this framework.

The last section defines the category-based framework underlying this thesis and gives a list of examples relevant to equational logic programming: many sorted algebra, order sorted algebra, Horn clause logic (with or without equality), and equational logic modulo a theory. Each example is presented with a fair amount of detail; we also show how they formally fit into the category-based framework previously introduced. The presentation of Horn clause logics contains a body of results showing how they can be technically regarded as ordinary (conditional) equational logics. Constraint logics are also mentioned, but we devote the whole of Chapter 6 to this example.

1.3.2 Category-based Equational Deduction

The categorical proof theory for equational logics is developed in this chapter. This begins with a category-based treatment of the concept of *congruence*. At this level, the finiteness of operators (or predicates) arity is encoded as a category-based finitarity condition related to congruences. The first section gives a category-based definition of the notion of $\mathcal{U}$-*equation* and of the satisfaction relation between models and $\mathcal{U}$-equations. The completeness of category-based equational logics is obtained in the next section, and Section 3.4 derives a first version of Herbrand's Theorem as its consequence.

The last section explores the consequences of the existence of free models. We get a more concrete formulation of the completeness of category-based equational deduction similar to the classical approaches. At this level we discuss the rôle played by the Axiom of Choice and of "finiteness of model operations" for the completeness of category-based

equational deduction. This section ends with a "non-empty sorts" version of Herbrand's Theorem.

1.3.3 Operational Semantics

This chapter begins with a very brief historical perspective on narrowing, followed by a discussion on the principles underlying our approach on the operational semantics. A preliminary section defines the category-based context of our treatment of the operational semantics, and approaches the notion of *rewriting context* from a category-based angle. The next section presents the inference rules of the paramodulation-based operational semantics for equational logic programming and establishes some related notations. Section 4.3 is devoted to the completeness of model theoretic paramodulation, Section 4.4 to paramodulation modulo a model morphism, and Section 4.5 to the rôle of confluence in establishing the completeness of paramodulation for the case of oriented rules. The theory developed in the first part of this chapter is applied in the next section to proving the completeness of many sorted narrowing when programs are term rewriting systems, and it also reviews the completeness of many sorted basic narrowing assuming the canonicity of the rewriting system.

The chapter on operational semantics ends with a section illustrating order sorted basic narrowing with runs of the Eqlog system. The *constructor discipline* is briefly presented as a control strategy in the context of the Eqlog system.

1.3.4 Modularisation

The chapter begins with a general discussion on the OBJ-like modularisation principles (including some history) and its advantages, a description of the soundness and completeness problems for module imports specific to equational logic programming, and a discussion on the rôle of category-based in the treatment of modularisation in equational logic programming. Section 5.1 presents some basic results in the context of semiexact institutions including a theorem that is fundamental to the semantics of parameterisation (i.e., generic modules) for OBJ-like languages.[19] Section 5.2 provides the bridge between the theory of institutions and the category-based framework of the thesis, and proves a generic[20] Satisfaction Condition in this context. Quantifier translations appear as free models along signature morphisms, and sentence translations as universal morphisms between Kleisli categories. This provides a basis for the category-based semantics of queries and solution forms *versus* modularisation developed in the next section, where the main result is the soundness of any module import and the completeness of persistent module imports. The soundness and completeness for equational logic programming module imports is shown to be an instantiation of the more abstract notion of soundness and completeness for institutions with an entailment relation. This involves an eccentric institution in which models are queries, sentences are substitutions, and signatures are collections of logical variables.

The last section gives a generalisation of the Theorem of Constants within the framework of category-based equational logics.

[19] Including Eqlog viewed as a specification language.

[20] For equational logics.

1.3.5 Extensible Constraint Logic Programming

This chapter gives a category-based semantics to extensible constraint logic programming by embedding constraint logics within the framework of category-based equational logics. It then uses the machinery of the previous chapters for proving a constraint logic version of Herbrand's Theorem.

1.4 The Programming Language Eqlog

Eqlog [38] is a programming and specification language being developed by the author at Oxford University, to combine constraint logic programming with equational programming. Its default operational semantics is order sorted narrowing[21], but particular cases can be computed by efficient built in algorithms over suitable data structures, with their functions and relations, including equality, disequality, and the usual orderings for numbers and lists. Initiality in Horn clause logic with equality provides a rigorous semantics for functional programming, logic programming, and their combination, as well as for the full power of constraint programming, allowing queries with logical variables over combinations of user-defined and built in data types [39].

Eqlog has a powerful type system that allows subtypes, based on order sorted algebra [41]. The method of *retracts*, a mathematically rigorous form of runtime type checking and error handling, gives Eqlog a syntactic flexibility comparable to that of untyped languages, while preserving all the advantages of strong typing [34]. The order sortedness of Eqlog not only greatly increases expressivity and the efficiency of unification (see [74]), but it also provides a rigorous framework for multiple data representations and automatic coercions among them. Uniform methods of conversion among multiple data representations are essential for reusing already programmed constraint solvers, because they will represent data in various ways. Order sorted algebra provides a precise and systematic equational theory for this, based on initial semantics (see [73] for a detailed discussion, [34] and [73] for some further examples).

Eqlog also supports loose specifications through its so-called *theories*, and provides *views* for asserting the satisfaction of theories by programs as well as relationships of refinement among specifications and/or programs. This relates directly to Eqlog's powerful form of modularity, with generic (i.e., parameterised) modules and views, based on the same principles as the OBJ language (see [38]). Theories specify both syntactic structure and semantic properties of modules and module interfaces. Modules can be parameterised, where actual parameters are modules. Modules can also import other modules, thus supporting multiple inheritance at the module level. For parameter instantiation, a view binds the formal entities in an interface theory to actual entities in a module. *Module expressions* allow complex combinations of already defined modules, including sums, instantiations and transformations; moreover, evaluating a module expression actually builds a software system from the given components.[22] Thus parameterised programming in Eqlog gives significant support for large programs through module composition, and [28] shows that it also provides the power of higher order functions. The semantics of module importation is given by conservative extensions of theories in Horn clause logic with equality [39]. The stronger notion of persistent extension underlies generic modules.

[21] Section 4.7 contains some examples of order sorted narrowing based Eqlog runs.

[22] Chapter 5 contains some simple examples of parameterised modules and instantiations.

1.4.1 Eqlog as a framework for decision procedures

From the very beginning logic programming was based on first order logic, paying tribute to its success in the foundations of mathematics. Prolog is now the only logic programming language that is quite widely used worldwide. Eqlog not only combines traditional logic programming with equational programming, it is also an *extensible modular* constraint programming language, which permits user-defined abstract data types and the reuse of existing code for constraint solvers for various problems. The fact that Eqlog is implemented in Kyoto Common Lisp supports this flexibility, because both Common Lisp and C programs can easily be included, and many other languages have translators into C. Gaussian elimination for systems of linear equations or packages for solving systems of linear inequalities are examples of what can be done. Of course, many decidable problems may not already have such efficient algorithms, but they can still be solved by the general method of narrowing, which in some cases can be as efficient as computation in an ordinary functional language.

This work assumes some familiarity with the basic notions of universal algebra and category theory. We generally use the same notation and terminology as Mac Lane's standard category theory textbook [64], except that the composition of arrows is denoted by ";" and written in the diagrammatic order. Application of functions (functors) to arguments may be written either normally by using parentheses, or else in the diagrammatic order without parentheses.

Categories are usually denoted by capital bbold letters; the standard ones usually have a name whose first letter is written in capital bbold. For example, the category of sets and functions is denoted by $\mathbb{S}et$, and the category of categories and functors is denoted by $\mathbb{C}at$. The opposite of a category $\mathbb{C}$ is denoted by $\mathbb{C}^{op}$; it has the same class of objects as $\mathbb{C}$, but all arrows are reversed. Functors are usually (but not always!) denoted by caligraphic capital letters, particularly for 'functor variables' as opposed to functors whose action is known. Objects in categories are usually denoted by small or italic capital letters; the class of objects of a category $\mathbb{C}$ is denoted by $|\mathbb{C}|$. The set of arrows in $\mathbb{C}$ having the object a as source and the object b as target is denoted by $\mathbb{C}(a, b)$.

2.0.2 Comma categories

Recall from [64] that given two functors $\mathbb{C} \xrightarrow{\mathcal{C}} \mathbb{E} \xleftarrow{\mathcal{D}} \mathbb{D}$, the **comma category** $(\mathcal{C}{\downarrow}\mathcal{D})$ has arrows $c\mathcal{C} \xrightarrow{t} d\mathcal{D}$ as objects and pairs of arrows $\langle f, g\rangle$ as morphisms, such that

$$\begin{tikzcd} c\mathcal{C} \arrow[r, "t"] \arrow[d, "f\mathcal{C}"'] & d\mathcal{D} \arrow[d, "g\mathcal{D}"] \\ c'\mathcal{C} \arrow[r, "t'"'] & d'\mathcal{D} \end{tikzcd}$$

commutes. For functors collapsing everything to a constant object (i.e., to an identity arrow) we use the object itself as notation. For any object $e \in |\mathbb{E}|$, the forgetful functor $(e{\downarrow}\mathbb{E}) = (e{\downarrow}1_{\mathbb{E}}) \to \mathbb{E}$ is denoted $\mathbb{E}_e$.

2.0.3 Limits and colimits

A **diagram** in a category $\mathbb{C}$ is a functor $J \xrightarrow{C} \mathbb{C}$. A **cone** $\gamma\colon\ d \to C$ consists of an object $d \in |\mathbb{C}|$ (called the **apex** of the cone) and a $|J|$-indexed family of arrows $\{d \xrightarrow{\gamma_i} C(i)\}_{i\in|J|}$ such that $\gamma_j; C(u) = \gamma_i$ for any u in J:

$$j \xrightarrow{\quad u \quad} i$$

$$\begin{tikzcd} C(j) \arrow[rr, "C(u)"] & & C(i) \\ & c \arrow[ul, "\mu_j"'] \arrow[ur, "\mu_i"] & \\ & d \arrow[u, "f"] \arrow[uul, "\gamma_j"] \arrow[uur, "\gamma_i"'] & \end{tikzcd}$$

A **limit** of C is a *minimal* cone over C, i.e., a cone $\mu\colon c \to C$ such that for any other cone $\gamma\colon d \to C$ there exists a unique arrow $f\colon d \to c$ in $\mathbb{C}$ such that $f;\mu = \gamma$.

Co-cone and **colimit** are dual to the notions of cone and limit, i.e., their defintion can be obtained by reversing the arrows in the definition of limits. This can be visualised by the following diagram:

$$\begin{tikzcd}
C(j) \arrow[rr, "C(u)"] \arrow[dr, "\mu_j"] \arrow[ddr, "\gamma_j"'] & & C(i) \arrow[dl, "\mu_i"'] \arrow[ddl, "\gamma_i"] \\
& c \arrow[d, "f"] & \\
& d &
\end{tikzcd}$$

Particular limits and colimits are obtained by fixing the shape of the diagrams, i.e., the category J. When J is discrete (i.e., it consists only of identity arrows) we get products and coproducts, respectively. When J consists only of two objects and a parallel pair of arrows between these, we get equalisers and coequalisers, respectively.

A functor $\mathbb{D}' \xrightarrow{\mathcal{U}} \mathbb{D}$ **creates colimits** iff for any colimit $D \xrightarrow{\mu} c$ (of a diagram $J \xrightarrow{D} \mathbb{D}$ in $\mathbb{D}$), there exists a colimit μ' in $\mathbb{D}'$ such that $\mu'\mathcal{U} = \mu$.

A category $\mathbb{J}$ is **filtered** iff for any objects $i, j \in |\mathbb{J}|$, there is an object $k \in |\mathbb{J}|$ such that $i \to k \leftarrow j$.

2.0.4 2-categories

Given two functors $\mathcal{S}, \mathcal{T}\colon \mathbb{A} \to \mathbb{B}$, a **natural transformation** $\tau\colon \mathcal{S} \to \mathcal{T}$ consists of an $|\mathbb{A}|$-indexed family of arrows in $\mathbb{B}$, $\{a\mathcal{S} \xrightarrow{a\tau} a\mathcal{T}\}_{a \in |\mathbb{A}|}$ such that for all f in $\mathbb{A}$ the following diagram commutes:

$$\begin{tikzcd}
a \arrow[d, "f"'] & a\mathcal{S} \arrow[r, "a\tau"] \arrow[d, "f\mathcal{S}"'] & a\mathcal{T} \arrow[d, "f\mathcal{T}"] \\
a' & a'\mathcal{S} \arrow[r, "a'\tau"'] & a'\mathcal{T}
\end{tikzcd}$$

As "functor homomorphisms" natural transformations compose point-wise in the obvious way. This is called the *vertical* composite of natural transformation:

$$\begin{tikzcd}
\mathbb{A} \arrow[r, shift left=2] \arrow[r, "\downarrow\sigma"] \arrow[r, shift right=2] \arrow[r, phantom, shift left=1, "\downarrow\tau"] & \mathbb{B}
\end{tikzcd}$$

i.e., $a(\sigma;\tau) = a\sigma; a\tau$. There is another *horizontal* composite of natural transformations $\tau\tau'\colon \mathcal{S};\mathcal{S}' \to \mathcal{T};\mathcal{T}'$

$$\begin{tikzcd}
\mathbb{A} \arrow[r, shift left=2, "\mathcal{S}"] \arrow[r, "\downarrow\tau"] \arrow[r, shift right=2, "\mathcal{T}"'] & \mathbb{B} \arrow[r, shift left=2, "\mathcal{S}'"] \arrow[r, "\downarrow\tau'"] \arrow[r, shift right=2, "\mathcal{T}'"'] & \mathbb{C},
\end{tikzcd}$$

and there is an *Interchange Law*: given three categories and four natural transformations

$$\begin{tikzcd}
\mathbb{A} \arrow[r, shift left=2] \arrow[r, "\downarrow\sigma"] \arrow[r, shift right=2] \arrow[r, phantom, shift left=1, "\downarrow\tau"] & \mathbb{B} \arrow[r, shift left=2] \arrow[r, "\downarrow\sigma'"] \arrow[r, shift right=2] \arrow[r, phantom, shift left=1, "\downarrow\tau'"] & \mathbb{C},
\end{tikzcd}$$

the "vertical" composites and the "horizontal" composites are related by

$$(\sigma;\tau)(\sigma';\tau') = (\sigma\sigma');(\tau\tau').$$

Functors and natural transformations form a 2-category (i.e., $\mathbb{C}at$ is a 2-category). A **2-category** is a class of arrows (called **2-cells**) for two different compositions which together satisfy the Interchange Law, and in which every identity arrow for the first composite is also an identity for the second composite. The identities for the vertical composites are called **1-cells**, and the identities for the horizontal composites are called **0-cells**.

2.1 Categorical Relations

The categorical version of binary relation plays a central rôle in this work.

2.1.1 Representations of binary relations

Definition 2.1 Let a be an object of a category $\mathbb{X}$. A **binary relation representation on** a is a parallel pair of arrows $s, t \in \mathbb{X}(k, a)$, denoted $k \xrightarrow{\langle s,t \rangle} a$ or just $\langle s, t \rangle$. $\square$

Here k plays the rôle of "object of indices" and s, t stand for the projections which give the left hand side and the right hand side of any pair of elements belonging to the relation.[23]

Example 2.2 Let $\leq$ be the usual "less than or equal" relation on the set ω of natural numbers. We can define the set of indices to be $\{(x, y) \mid x, y \in \omega \text{ and } x \leq y\}$, and let $s, t \colon k \to \omega$ be the projections, i.e., $s(x, y) = x$ and $t(x, y) = y$. $\square$

Definition 2.3 Let $k \xrightarrow{\langle s,t \rangle} a$ and $k' \xrightarrow{\langle s',t' \rangle} a$ be binary relation representations on the same object a. Then $\langle s, t \rangle$ is **included in** $\langle s', t' \rangle$ (denoted $\langle s, t \rangle \subseteq_a \langle s', t' \rangle$, or just $\langle s, t \rangle \subseteq \langle s', t' \rangle$) iff there is a map $h \colon k \to k'$ between the objects of indices such that $s = h; s'$ and $t = h; t'$. $\square$

$$\begin{tikzcd}
k \arrow[r, shift left, "s"] \arrow[r, shift right, "t"'] \arrow[d, shift right, "h"'] & a \\
k' \arrow[ur, shift left, "s'"] \arrow[ur, shift right, "t'"'] \arrow[u, shift right, "h"'] &
\end{tikzcd}$$

Fact 2.4 For any category $\mathbb{X}$ let $\mathbb{X}_{\rightrightarrows}$ be the category having the same objects as $\mathbb{X}$ and pairs of parallel arrows as maps. Let $\Delta_{\mathbb{X}}$ be the functor $\mathbb{X} \to \mathbb{X}_{\rightrightarrows}$ doubling each arrow in $\mathbb{X}$. Then for any object a in $\mathbb{X}$, the inclusion $\subseteq_a$ between binary relation representations on a is the preorder obtained by collapsing[24] the comma category $(\Delta_{\mathbb{X}} \downarrow a)$. $\square$

Definition 2.5 Two relation representations Q and Q' on the same object a are **equivalent** (denoted $Q \equiv_a Q'$, or $Q \equiv Q'$ for short) if and only if $Q \subseteq Q'$ and $Q' \subseteq Q$. $\square$

[23] For technical simplicity, we don't require s and t to be monics. In this way, a binary relation can have more than one representation, each having different objects of indices. Some of these objects of indices are not necessarily isomorphic; this allows repetitions of "elements" in a representation of a relation.

[24] The elements of the preorder are the objects of the category, and two elements are related under the preorder iff there is an arrow between them.

Binary relations are classes of equivalent representations:

Definition 2.6 Let a be an object of a category $\mathbb{X}$. A **binary relation** on a is an equivalence class of $\equiv_a$. $\square$

For simplicity, we will often use representations instead of equivalence classes as binary relations. Notice that the concept of inclusion between binary relation representations can be extended to binary relations proper. We will often write sQt for $\langle s, t\rangle \subseteq Q$, where Q is a binary relation.

2.1.2 Unions of relations

Definition 2.7 Let $\{Q_i\}_{i\in I}$ be a family of relations on an object a of a category $\mathbb{X}$. The **union** $\bigcup_{i\in I} Q_i$ is the least upper bound of this family with respect to the inclusion relation. Dually, the **intersection** $\bigcap_{i\in I} Q_i$ is the greatest lower bound. $\square$

Lemma 2.8 If $\mathbb{X}$ has colimits, then the union $\bigcup_{i\in I} Q_i$ of any family of binary relations on an object a of $\mathbb{X}$ exists, and may be constructed as a colimit in the comma category $(\Delta_{\mathbb{X}}{\downarrow}a)$.

Proof: This follows from Fact 2.4 and from the fact that the forgetful functor $(\Delta_{\mathbb{X}}{\downarrow}a) \to \mathbb{X}$ creates colimits. $\square$

Corollary 2.9 If $\mathbb{X}$ has binary coproducts and colimits of filtered preorders, then it has unions of binary relations.

Proof: By the construction of [small] colimits from binary coproducts and colimits of filtered preorders (see [64]). $\square$

Fact 2.10 Assume $\mathbb{X}$ has coproducts. Let $\langle s_i, t_i\rangle_{i\in I}$ be a family of relations on $a \in |\mathbb{X}|$ and let $f\colon a \to b$ be an arrow in $\mathbb{X}$. Then

$$(\bigcup_{i\in I}\langle s_i, t_i\rangle); f = \bigcup_{i\in I}\langle s_i; f, t_i; f\rangle.$$

Proof: Let k be $\coprod_{i\in I} k_i$, where k_i is the object of indices for $\langle s_i, t_i\rangle$. Then $\bigcup_{i\in I}\langle s_i, t_i\rangle$ can be regarded as the coproduct of $\langle s_i, t_i\rangle_{i\in I}$ in $(\Delta_{\mathbb{X}}{\downarrow}a)$. By the universal property of coproducts, $(\bigcup_{i\in I}\langle s_i, t_i\rangle); f$ is the coproduct of $\langle s_i; f, t_i; f\rangle_{i\in I}$, that is, $\bigcup_{i\in I}\langle s_i; f, t_i; f\rangle$. $\square$

Definition 2.11 A binary relation Q is **atomic** iff it does not have any proper subrelations, i.e., the empty relation and Q are its only subrelations. $\square$

In the case of [many-sorted] sets, the atomic relations are exactly the one-element relations.

Definition 2.12 A coproduct $\coprod_{i\in I} k_i$ in a category $\mathbb{X}$ is **disjoint** iff any map $f\colon p \to \coprod_{i\in I} k_i$ can be represented as $f = \coprod_{i\in I} f_i$ with $f_i\colon p_i \to k_i$ and $p = \coprod_{i\in I} p_i$. A category has **disjoint coproducts** iff it has coproducts and all its coproducts are disjoint. $\square$

Example 2.13 In $\mathbb{S}et$ any function $f\colon p \to \coprod_{i\in I} k_i$ can be written as $f = \coprod_{i\in I} f_i$ where $f_i\colon f^{-1}(k_i) \to k_i$. This works because the coproducts of sets are disjoint unions.

The same situation holds for the case of many-sorted sets and functions. $\square$

Lemma 2.14 Let $\mathbb{X}$ be a category with disjoint coproducts. If $R \subseteq \bigcup_{i \in I} Q_i$ (as binary relations), then R can be represented as $R = \bigcup_{i \in I} R_i$ with $R_i \subseteq Q_i$.

Proof: Let R be $p \xrightarrow{\langle s,t \rangle} a$ and k_i be the object of indices of Q_i for each $i \in I$. Then the object of indices of $\bigcup_{i \in I} Q_i$ can be taken as $\coprod_{i \in I} k_i$. Let $f\colon\ p \to \coprod_{i \in I} k_i$ be the map between the indices representing the inclusion $R \subseteq \bigcup_{i \in I} Q_i$. Then $f = \coprod_{i \in I} f_i$ with $f_i\colon\ p_i \to k_i$ and $\coprod_{i \in I} p_i = p$. Define R_i to be $\langle j_i; s, j_i; t \rangle$ for each $i \in I$, where $\{j_i\colon\ p_i \to p\}_{i \in I}$ are the injections of the coproduct co-cone. Now it is easy to see that $R_i \subseteq Q_i$ for each $i \in I$ and that $R = \bigcup_{i \in I} R_i$. □

2.1.3 Equivalences

In this subsection we introduce the notion of equivalence as a special binary relation. The following is a well known categorical definition of equivalence relations:

Definition 2.15 The **kernel** of an arrow h, denoted $ker(h)$, is the pullback of h with itself. A relation $\langle s, t \rangle$ on a is an **equivalence** iff there is a map h such that $\langle s, t \rangle = ker(h)$. □

Fact 2.16 If $\mathbb{X}$ has pullbacks, then ker is a functor $(a{\downarrow}\mathbb{X}) \to (\Delta_{\mathbb{X}}{\downarrow}a)$. □

In ordinary set theory, equivalences are characterised as reflexive, symmetric and transitive binary relations. The following definition deals with reflexivity, symmetry and transitivity at the level of categorical binary relations.

Definition 2.17 Let $\mathbb{X}$ be a category and consider an object a in $\mathbb{X}$. The **diagonal** of a is the relation

$$D_a = \bigcup \{\langle t, t \rangle \mid t \in \mathbb{X}(k, a)\}$$

Then a relation Q on a is **reflexive** iff $D_a \subseteq Q$.
$\langle l, r \rangle$ is **symmetric** iff $\langle l, r \rangle = \langle r, l \rangle$, and
Q is **transitive** iff $\langle s, u \rangle \subseteq Q$ whenever $\langle s, t \rangle \subseteq Q$ and $\langle t, u \rangle \subseteq Q$ for some t. □

Fact 2.18 Any equivalence is reflexive, symmetric and transitive. □

Fact 2.19 The symmetric closure of a binary relation $\langle l, r \rangle$ on a exists, and is given by

$$sym\langle l, r \rangle = \langle l, r \rangle \cup \langle r, l \rangle.$$

□

Definition 2.20 A category $\mathbb{X}$ has **filtered unions of equivalences** iff for each object a the functor $ker\colon\ (a{\downarrow}\mathbb{X}) \to (\Delta_{\mathbb{X}}{\downarrow}a)$ preserves filtered colimits. □

Fact 2.21 The forgetful functor $(a{\downarrow}\mathbb{X}) \to \mathbb{X}$ creates filtered colimits. □

Example 2.22 The category $\mathbb{S}et^S$ of S-sorted sets and functions has filtered unions of equivalences. This reduces to the fact that unions of filtered families of equivalence relations on a set A are still equivalence relations. Filteredness is essential, as suggested by the two equivalences on $\{1,2,3\}$ generated by $\{(1,2)\}$ and, $\{(2,3)\}$ respectively. Their union is not an equivalence since it is not transitive because it does not contain $(1,3)$.

More formally, consider a set A and let $\mu\colon \{A \xrightarrow{f_i} B_i\}_{i\in I} \to (A \xrightarrow{f} B)$ be a filtered colimit in $(A{\downarrow}\mathbb{S}et^S)$. Let K_i be the kernel of f_i for $i \in I$. By Fact 2.21 $\mu\colon \{B_i\}_{i\in I} \to B$ is a filtered colimit in $\mathbb{S}et^S$, therefore B is $(\coprod_{i\in I} B_i)/_\sim$, where $b \sim b'$ iff b and b' get mapped into the same element by some function in the diagram $\{B_i\}_{i\in I}$. The existence of filtered unions of equivalences means that $ker(f)$ should be $\bigcup_{i\in I} K_i$. Then $\bigcup_{i\in I} K_i = \{(a,a') \mid \exists i \in |I| \text{ such that } f_i(a) = f_i(a')\}$ and $ker(f) = \{(a,a') \mid \forall j \in |I|, f_j(a)/_\sim = f_j(a')/_\sim\}$. By the definition of $\sim$, $ker(f) \subseteq \bigcup_{i\in I} K_i$. But $\bigcup_{i\in I} K_i \subseteq ker(f)$ since each $K_i \subseteq ker(f)$. $\square$

2.2 Finiteness

This section deals with finiteness. The concept of finiteness is essential for proving the completeness of equational deduction, and consequently of the operational semantics.

2.2.1 Finite objects

The link between finiteness and filteredness is now well established in several different branches of mathematics. Although it is hard to trace back its origins, we mention the rôle played by filteredness in explaining some Birkhoff-like axiomatisation results in abstract model theory. Our categorical definition of finiteness corresponds to the definition of "L-small object" in [1] when L is the class of all directed posets, and it also generalises the well known notion of a "finite element" in a partially ordered set.

Definition 2.23 An object k in a category $\mathbb{X}$ is **finite** iff for any map $f\colon k \to d$ to the apex of a colimiting co-cone $\mu\colon D \to d$ in $\mathbb{X}$ over a filtered diagram D, there exists $i \in |D|$ and a map $f_i\colon k \to D(i)$ such that $f_i;\mu_i = f$. $\square$

Example 2.24 In $\mathbb{S}et^S$, the finite objects are exactly those S-sorted sets that are finite on each component in the ordinary sense.

Consider a finite S-sorted set k and a map $f\colon k \to d$ to the apex of a colimiting co-cone $\mu\colon D \to d$ over a filtered diagram D in $\mathbb{S}et^S$.

$$\begin{tikzcd}
D(j) \arrow[rr] \arrow[dr, "\mu_j"] & & D(i) \arrow[dl, "\mu_i"'] \\
& d & \\
& k \arrow[u, "f"] \arrow[uur, "f_i"'] &
\end{tikzcd}$$

Due to the nature of colimits in set theory, $d = \bigcup_{j\in|D|} \mu_j(D(j))$. Therefore, for each element $e \in d$, there exists j such that $e \in \mu_j(D(j))$. The same holds for any subset of d, in particular for $f(k)$. Since d is finite and D is filtered, there exists $i \in |D|$ such that $f(k) \subseteq \mu_i(D(i))$. Now it is easy to construct a map $f_i\colon k \to D(i)$ such that $f_i;\mu_i = f$.

For the converse, assume the hypotheses and suppose k is not finite. Let D be an ω-diagram such that $D(j) \subseteq k$ and $D(j)$ is strictly included in $D(j+1)$ for each $j \in \omega$. Such a diagram exists because k is not finite. Let f be any right inverse to the inclusion $\cup_{j\in\omega} D(j) \subseteq k$. Suppose there exists $i \in \omega$ and $f_i\colon k \to D(i)$ such that $f_i;\mu_i = f$. Let e be an element in $D(i+1)$ that doesn't belong to $D(i)$. If $f_i;\mu_i$ was equal to f, then $e = f(e) = \mu_i(f_i(e)) = e$, which clashes with the fact that e doesn't belong to $D(i)$. □

Example 2.25 In a similar manner to the previous example we can easily see that in the category $\mathbb{V}ect_K$ of vector spaces and linear transformations over some field K, the finite objects are exactly the finite dimensional vector spaces. □

Lemma 2.26 Suppose $\mathbb{X}$ has binary coproducts. Then $k_1 \coprod k_2$ is finite if k_1 and k_2 are finite.

Proof: Consider a colimiting co-cone $\mu\colon D \to d$ over a filtered diagram D in $\mathbb{X}$. Let $f = [f_1, f_2]\colon k_1 \coprod k_2 \to d$ with $f_i\colon k_i \to d$. By the finiteness of k_1 and k_2 and because D is filtered, there is an object j and two maps $g_i\colon k_i \to D(j)$ (for $i = 1, 2$) such that $g_i;\mu_j = f_i$. Define g to be $[g_1, g_2]\colon k_1 \coprod k_2 \to D(j)$. Then $g;\mu_j = f$. □

2.2.2 Finiteness for binary relations

Definition 2.27 A binary relation is **finite** iff at least one of its representations has a finite object of indices. □

Fact 2.28 Any finite binary relation on $a \in |\mathbb{X}|$ is finite as an object of $(\Delta_{\mathbb{X}}{\downarrow}a)$. □

The converse doesn't necessarily hold. However, a natural condition on the base category ensures that finite binary relations on an object a correspond exactly to the finite objects in $(\Delta_{\mathbb{X}}{\downarrow}a)$. The next definition is adapted from [1]:

Definition 2.29 The category $\mathbb{X}$ is **algebroidal** iff each of its objects can be presented as a filtered colimit of finite objects. □

Both $\mathbb{S}et^S$ and $\mathbb{V}ect_K$ are algebroidal categories. In the former case, any S-sorted set is the union of its finite subsets, while in the latter case, each vector space over a field K is the colimit of its finite dimensional subspaces. Another well known example comes from domain theory. A lattice is called algebraic iff each of its elements is a directed union of finite elements.

Fact 2.30 If $\mathbb{X}$ has binary coproducts, then for any binary relation Q on a, $\{Q_0 \text{ finite} \mid Q_0 \subseteq Q\}$ is filtered.

Proof: By Lemma 2.26. □

Corollary 2.31 If $\mathbb{X}$ is algebroidal and has binary coproducts, then for any binary relation Q on a,

$$Q = \bigcup\{Q_0 \text{ finite} \mid Q_0 \subseteq Q\}.$$

Proof: Let Q be $\langle s, t\rangle$ with d the object of indices. Since $\mathbb{X}$ is algebroidal, d is the apex of a colimiting co-cone $\mu\colon D \to d$ of a diagram whose nodes are finite objects in $\mathbb{X}$. For each node i in D, the binary relation $\langle \mu_i; s, \mu_i; t\rangle$ is finite and $\langle s, t\rangle$ is the colimit of $\langle \mu_i; s, \mu_i; t\rangle_{i\in|D|}$ in $(\Delta_{\mathbb{X}}{\downarrow}a)$ since the forgetful functor $(\Delta_{\mathbb{X}}{\downarrow}a) \to \mathbb{X}$ creates colimits. By Lemma 2.8, $\langle s, t\rangle = \bigcup_{i\in|D|}\langle \mu_i; s, \mu_i; t\rangle$. But

$$\bigcup_{i\in|D|} \langle \mu_i; s, \mu_i; t\rangle \subseteq \bigcup\{\langle s_0, t_0\rangle \text{ finite} \mid \langle s_0, t_0\rangle \subseteq \langle s, t\rangle\}.$$

Therefore, $\langle s, t\rangle \subseteq \bigcup\{\langle s_0, t_0\rangle \text{ finite} \mid \langle s_0, t_0\rangle \subseteq \langle s, t\rangle\}$. This proves the corollary since the opposite inclusion is trivial. □

The following corollary motivates Definition 2.27 and shows that the finite binary relations on a correspond exactly to finite objects in $(\Delta_{\mathbb{X}}{\downarrow}a)$.

Corollary 2.32 If $\mathbb{X}$ is algebroidal and has binary coproducts, then for any object a in $\mathbb{X}$, any finite object in $(\Delta_{\mathbb{X}}{\downarrow}a)$ is a finite binary relation on a.

Proof: Assume that $k \xrightarrow{\langle s,t\rangle} a$ is finite as an object in $(\Delta_{\mathbb{X}}{\downarrow}a)$. By Corollary 2.31 $\langle s, t\rangle = \bigcup\{k_0 \xrightarrow{\langle s_0,t_0\rangle} a \text{ finite} \mid \langle s_0, t_0\rangle \subseteq \langle s, t\rangle\}$ and there exists $k_0 \xrightarrow{\langle s_0,t_0\rangle} a \subseteq \langle s, t\rangle$ finite such that $\langle s, t\rangle \subseteq \langle s_0, t_0\rangle$. □

Corollary 2.33 If $\mathbb{X}$ is algebroidal and has binary coproducts, then any atomic relation is finite. □

2.2.3 Reflexive-transitive closures

Throughout this subsection we assume that the category $\mathbb{X}$ is algebroidal and has disjoint[25] binary coproducts.

Lemma 2.34 A binary relation Q on a is symmetric iff $\langle s, t\rangle \subseteq Q$ implies $\langle t, s\rangle \subseteq Q$ for all finite $\langle s, t\rangle$.

Q is transitive iff for any finite relations $\langle s, t\rangle$ and $\langle t, u\rangle$, $\langle s, u\rangle \subseteq Q$ whenever $\langle s, t\rangle \subseteq Q$ and $\langle t, u\rangle \subseteq Q$.

Proof: Let Q be $k \xrightarrow{\langle s,t\rangle} a$. By Corollary 2.31,

$$\langle s, t\rangle = \bigcup\{\langle s_0, t_0\rangle \text{ finite} \mid \langle s_0, t_0\rangle \subseteq \langle s, t\rangle\},$$

in such a way that $\langle s, t\rangle$ could be presented as the colimit of the set in the right-hand side of the previous equality. From this, we deduce that $\langle t, s\rangle = \bigcup\{\langle t_0, s_0\rangle \text{ finite} \mid \langle s_0, t_0\rangle \subseteq \langle s, t\rangle\}$. But each finite $\langle t_0, s_0\rangle$ is included in $\langle s, t\rangle$ by hypothesis, therefore $\langle t, s\rangle \subseteq \langle s, t\rangle$.

For the second part of this lemma, consider $\langle s', t'\rangle, \langle t', u'\rangle \subseteq Q$ and let $\{k_i \xrightarrow{\mu_i} k\}_{i\in I}$ be a representation of k as a filtered colimit of finite objects. Let $s_i = \mu_i; s'$, $t_i = \mu_i; t'$ and $u_i = \mu_i; u'$. Then $\langle s_i, u_i\rangle \subseteq Q$ by hypothesis, and because $\langle s, u\rangle = \bigcup_{i\in|I|}\langle s_i, u_i\rangle$, we have $\langle s, u\rangle \subseteq Q$. □

Definition 2.35 Let Q and R be relations on the same object a. Then their **composition** is

$$Q \circ R = \bigcup\{\langle s, u\rangle \text{ finite} \mid \langle s, t\rangle \subseteq Q, \langle t, u\rangle \subseteq R \text{ for some } t\}.$$

□

Fact 2.36 Let Q and R be relations on the same object a. Then

$$\{\langle s, u\rangle \text{ finite} \mid \langle s, t\rangle \subseteq Q, \langle t, u\rangle \subseteq R \text{ for some } t\}$$

[25] In the sense of Definition 2.12.

is filtered. □

Lemma 2.37 Fix an object a in $\mathbb{X}$. Then

1. the composition of binary relations is monotonic with respect to the inclusions between relations,

2. the composition of binary relations on a is associative, and

3. $(Q_1 \cup Q_2) \circ R = (Q_1 \circ R) \cup (Q_2 \circ R)$ for any binary relations Q_1, Q_2, R on a.

Proof: 1. The proof of this falls out directly from the defintion of inclusions of categorical relations.

2. Consider Q, R, P binary relations on a. Then

$$(Q \circ R) \circ P = \bigcup\{\langle s, u\rangle \text{ finite} \mid \langle s, t\rangle \subseteq Q \circ R, \langle t, u\rangle \subseteq P \text{ for some } t\}.$$

Because of Fact 2.36, for each $\langle s, t\rangle \subseteq Q \circ R$ finite, there exists v such that $\langle s, v\rangle \subseteq Q$ and $\langle v, t\rangle \subseteq R$. Then

$$(Q \circ R) \circ P = \bigcup\{\langle s, u\rangle \text{ finite} \mid \langle s, v\rangle \subseteq Q, \langle v, t\rangle \subseteq R, \langle t, u\rangle \subseteq P \text{ for some } v, t\}.$$

The same holds for $Q \circ (R \circ P)$. Therefore $(Q \circ R) \circ P = Q \circ (R \circ P)$.

3. $(Q_1 \circ R) \cup (Q_2 \circ R) \subseteq (Q_1 \cup Q_2) \circ R$ holds by the monotonicity of $\circ$ with respect to $\subseteq$. For the opposite inclusion, consider $\langle s, u\rangle$ finite such that $\langle s, t\rangle \subseteq Q_1 \cup Q_2$ and $\langle t, u\rangle \subseteq R$ for some t. Let $\langle s_i, t_i\rangle \subseteq Q_i, \langle t_i, u_i\rangle \subseteq R$, $i \in \{1, 2\}$, such that $\langle s, t\rangle = \langle s_1, t_1\rangle \cup \langle s_2, t_2\rangle$. Then $\langle s_i, u_i\rangle \subseteq Q_i \circ R$ and $\langle s, u\rangle = \langle s_1, u_1\rangle \cup \langle s_2, u_2\rangle \subseteq (Q_1 \circ R) \cup (Q_2 \circ R)$. □

Proposition 2.38 Any relation Q on an object a in $\mathbb{X}$ has a reflexive-transitive closure (i.e., the least reflexive-transitive relation containing Q), namely

$$Q^* = \bigcup_{n \in \omega} Q_n$$

where $Q_0 = D_a$ and $Q_{n+1} = Q_n \cup Q \circ Q_n$.

Proof: The reflexivity of Q^* holds because of Q_0. For proving the transitivity of Q^*, we show first by induction on $m \in \omega$ that $Q_m \circ Q_n \subseteq Q_{m+n}$ for any $n \in \omega$. For the induction step,

$$\begin{aligned} Q_{m+1} \circ Q_n &= (Q_m \cup Q \circ Q_m) \circ Q_n \\ &= Q_m \circ Q_n \cup Q \circ Q_m \circ Q_n \qquad \text{(by Lemma 2.37)} \\ &\subseteq Q_{m+n} \cup Q \circ Q_{m+n} \\ &= Q_{m+n+1}. \end{aligned}$$

Now consider $\langle s, t\rangle, \langle t, u\rangle \subseteq Q^*$ finite. Since $Q^* = \bigcup_{n \in \omega} Q_n$ is a filtered colimit, there exists $m, n \in \omega$ such that $\langle s, t\rangle \subseteq Q_n$ and $\langle t, u\rangle \subseteq Q_m$. Therefore $\langle s, u\rangle \subseteq Q_{m+n} \subseteq Q^*$. By Lemma 2.34, Q^* is transitive.

Let R be any reflexive-transitive relation on a and containing Q. By induction on $n \in \omega$, $Q_n \subseteq R$. Therefore $Q^* \subseteq R$. □

2.2.4 Confluent relations

In a set theoretic framework, the following definition represents an extension of the ordinary notion of confluence from elements to finite families (or tuples) of elements. Confluent relations appear in the context of abstract rewriting systems [55].

Definition 2.39 A binary relation Q on an object a of $\mathbb{X}$ is **confluent** iff for any finite $\langle s, t\rangle, \langle s, t'\rangle \subseteq Q$, there exists u such that $\langle t, u\rangle, \langle t', u\rangle \subseteq Q$. $\square$

2.3 Models and Domains

This thesis takes a top-down approach to equational logics, in the spirit of abstract model theory [5, 33], in the sense that all concepts and results are developed at the highest possible level of abstraction. New levels of concreteness, necessary for some concepts and results, are obtained by adding new hypotheses to the previous levels. The basic framework distills the essential ingredients characterising equational logics.

The semantics of any [equational] logical system is given by its *models*. In general, the soundness of the inference rules of a logical system is checked against the models by using a satisfaction relation between models and sentences (in traditional mathematical logic this idea was first formalised in [93]). *Model morphisms* are translations between models. We assume that models and their morphisms form a category. Inspired by the theory of institutions [33], equational logics can be "localised" to signatures. A model is an interpretation of a particular signature into a *domain*. Therefore any model has an underlying domain, and moreover, this correspondence should be functorial. Any two parallel model morphisms identical as maps between the underlying domains are the same. These hypotheses are formulated within the following general assumption:

> **[BasicFramework]:** There is an abstract category of "models" $\mathbb{A}$ and a "forgetful" functor $\mathcal{U} : \mathbb{A} \to \mathbb{X}$ to a category of "domains" $\mathbb{X}$ that is faithful and preserves pullbacks.

In practice, the forgetful functor $\mathcal{U}$ always has a left adjoint $\mathcal{F}$, which means that for every $x \in |\mathbb{X}|$ (which can be thought as a domain of variables) there is a "free model" $x\mathcal{F}$, in the sense that there is a "canonical interpretation" $x\eta : x \to x\mathcal{F}\mathcal{U}$ of "the variables" into the free model satisfying the following universal property: for each $f : x \to A\mathcal{U}$ interpreting variables in a model A, there exists a unique model morphism $f^\sharp : x\mathcal{F} \to A$ extending f, in the sense that $x\eta; f^\sharp\mathcal{U} = f$.

$$\begin{tikzcd} x \arrow[rr, "x\eta"] \arrow[dr, "f"'] & & x\mathcal{F}\mathcal{U} \arrow[dl, "f^\sharp\mathcal{U}"] \\ & A\mathcal{U} & \end{tikzcd} \qquad \begin{tikzcd} & x\mathcal{F} \arrow[dl, "f^\sharp"] \\ A & \end{tikzcd}$$

Notice that $(\mathbb{A},\mathcal{U})$ can be regarded as a **concrete category** (in the sense of [60]) over the category of domains. The condition that $\mathcal{U}$ preserves pullbacks relates to the fact that congruences are equivalences; this will become more transparent later. Notice that $\mathcal{U}$ automatically preserves pullbacks whenever it has a left adjoint (see [64]).

The simplicity of this basic framework is an expression of the simplicity of equational logic in general. This framework supports the internalisation of all concepts and results

in equational logic; this internalisation will be called **category-based equational logic**. The rest of this section is devoted to the presentation of some major equational logical systems used in Computing Science within the framework of our general assumption.

2.3.1 Many sorted algebra

Many sorted algebra (abbreviated *MSA*) seems to have been first studied by Higgins [53] around 1963, and Benabou [8] gave an elegant category theoretic development around 1968, overcoming some of the technical difficulties[26] in [53]. The use of sorted sets (also called indexed families) for MSA was introduced by Goguen in lectures at the University of Chicago in 1968, and first appeared in print in [24]. Sorted sets allow a simpler notation than alternative approaches, and also allow *overloading*; however, overloading only reveals its full potential in order sorted algebra. It was later noted that using sorts in automatic theorem proving can be an advantage, because it can greatly reduce the search space (e.g., see [97]). The basic definitions for overloaded MSA are quite simple:

Definition 2.40 Given a set S, we let S^* denote the set of all finite sequences of elements from S, and we let $[]$ denote the empty sequence of elements from S. Given an S-sorted set A and $w = s_1 ... s_n \in S^*$, let $A^w = A_{s_1} \times \cdots \times A_{s_n}$; in particular, let $A^{[]} = \{\star\}$, some one pointed set.

A **signature** (S, Σ) is an $S^* \times S$-indexed set $\Sigma = \{\Sigma_{w,s} \mid w \in S^*,\ s \in S\}$; we often write just Σ instead of (S, Σ). Notice that this definition permits overloading, in that the sets $\Sigma_{w,s}$ need *not* be disjoint; this can be useful in many applications.

A Σ-**algebra** A consists of an S-sorted set A and a function $\sigma_A : A^w \to A_s$ for each $\sigma \in \Sigma_{w,s}$; the set A_s is called the **carrier** of A of sort s. A Σ-**homomorphism** from a Σ-algebra A to another B is an S-sorted function $f : A \to B$ such that[27]

$$f(\sigma_A(a)) = \sigma_B(f(a))$$

for each $a \in A^w$. $\Box$

Let $\mathbb{A}lg_\Sigma$ denote the category with Σ-algebras as objects and Σ-homomorphisms as morphisms. There is a forgetful functor $\mathcal{U} : \mathbb{A}lg_\Sigma \to \mathbb{S}et^S$ from the category of Σ-algebras to the category of S-sorted sets which forgets the interpretations of the operations in Σ. In this example, $\mathbb{A}lg_\Sigma$ is the category of models and $\mathbb{S}et^S$ is the category of domains.

Given a many sorted signature Σ, an S-sorted set X will be called a set of **variable symbols** if the sets X_s are disjoint from each other and from all the sets $\Sigma_{w,s}$. Given a set X of variable symbols, we let $T_\Sigma(X)$ denote the (S-sorted) **term algebra** with operation symbols from Σ and variable symbols from X; it is the **free** Σ-algebra generated by X, in the sense that if $v : X \to A$ is an **assignment**, i.e., a (many sorted) function to a Σ-algebra A, then there is a unique extension of v to a Σ-homomorphism $v^\sharp : T_\Sigma(X) \to A$. In order to make this construction more precise, we define $(T_\Sigma(X))_s$ to be the least set of strings of symbols such that

1. $\Sigma_{[],s} \cup X_s \subseteq (T_{\Sigma,s}(X))$, and
2. $\sigma \in \Sigma_{s1...sn,s}$ and $ti \in T_{\Sigma,si}(X)$ imply that the string $\sigma(t1, \ldots, tn)$ is in $T_{\Sigma,s}(X)$.

[26] These difficulties are discussed in [37], which gives a more technical survey of work in MSA.
[27] By $f(a)$ we understand $(f(a_1), \ldots, f(a_n))$ where $a = (a_1, \ldots, a_n)$.

The Σ-structure of $T_\Sigma(X)$ is the canonical one. (Strictly speaking, the usual term algebra is not free unless the constant symbols, in $\Sigma_{[],s}$ for $s \in S$, are mutually disjoint; however, even if they are not disjoint, a closely related term algebra, with constants annotated by their sort, is free.) This construction is a left adjoint to the forgetful functor $\mathcal{U} : \mathbb{A}lg_\Sigma \to \mathbb{S}et^S$.

Also, we let T_Σ denote the **initial** term Σ-algebra $T_\Sigma(\emptyset)$, recalling that this means that there is a unique Σ-homomorphism $!_A : T_\Sigma \to A$ for any Σ-algebra A. Call $t \in T_\Sigma$ a **ground Σ-term**. Given a ground Σ-term t, let t_A denote the element $!_A(t)$ in A. Call A **reachable** iff $!_A$ is surjective, i.e., iff each element of A is "named" by some ground term.

2.3.2 Order sorted algebra

The first paper on order sorted algebra (abbreviated *OSA*) [25] says that its main motivation is to provide a better way of treating errors in abstract data types;[28] another motivation is that the use of subsorts can greatly speed up certain theorem proving problems [96]. OSA adds to MSA a partial ordering on the set of sorts, which is interpreted as inclusion among the corresponding carriers; all approaches to OSA share this essential idea. The ideas in [25] were further refined by Goguen and Meseguer, starting around 1983. In [34] the basic OSA definitions are presented in a much more general form than in [41], and we follow that more general approach here.

Definition 2.41 [34] An **order sorted signature** is a triple $(S, \leq, \Sigma)$ such that (S, Σ) is a many sorted signature and $(S, \leq)$ is a partially ordered set. An order sorted signature is **monotone** iff

$$\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2} \text{ and } w_1 \leq w_2 \text{ imply } s_1 \leq s_2.$$

A $(S, \leq, \Sigma)$-**algebra** is a many sorted (S, Σ)-algebra A such that $s \leq s'$ in S implies $A_s \subseteq A_{s'}$. An order sorted Σ-algebra A is **monotone** iff

$$\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2} \text{ and } w_1 \leq w_2 \text{ and } s_1 \leq s_2 \text{ imply that } \sigma_{w_1,s_1} : A_{w_1} \to A_{s_1} \text{ equals } \sigma_{w_2,s_2} : A_{w_2} \to A_{s_2} \text{ on } A_{w_1}.$$

A $(S, \leq, \Sigma)$-**homomorphism** is a many sorted (S, Σ)-homomorphism $h : A \to B$ such that $s \leq s'$ in S implies $h_s(a) = h_{s'}(a)$ for all $a \in A_s$.

A partially ordered set $(S, \leq)$ is (upward) **filtered** iff for any two elements $s, s' \in S$ there is an element $s'' \in S$ such that $s, s' \leq s''$. A partially ordered set S is **locally filtered** iff each of its connected components[29] is filtered. An order sorted signature $(S, \leq, \Sigma)$ is **locally filtered** iff $(S, \leq)$ is locally filtered. □

Notice that there cannot be any overloaded constants if Σ is monotone. Also note that overloaded OSA is a proper generalisation of MSA, because (overloaded) MSA is the special case where the partially ordered set of sorts is discrete; some other approaches do not have (even ordinary non-overloaded) MSA as a special case.

Given a signature Σ in the sense of Definition 2.41, the interpretations of an overloaded operation symbol $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ in an algebra A need not necessarily agree on elements

[28] See [43] for a discussion of the difficulties with handling errors in MSA.

[29] Given a poset $(S, \leq)$, let $\equiv$ denote the transitive and symmetric closure of $\leq$. Then $\equiv$ is an equivalence relation whose equivalence classes are called the **connected components** of $(S, \leq)$.

that belong to the intersection of carriers for w_1 and w_2; thus, a strong form of overloading is supported. For this reason, in [34] this approach is called **overloaded OSA**. Note that Definition 2.41 generalises [41], where both the signatures and algebras are assumed to be monotone. Goguen and Diaconescu introduce in [34] the concept of **signature of non-monotonicities** as a mechanism for saying which operation declarations should be considered non-monotonic.

In [41], overloaded OSA is developed with coherent signatures in a way that closely parallels traditional general algebra; in particular, there are order sorted versions of subalgebra, congruence, term, deduction, initial and free algebras, completeness, etc. Regularity guarantees that every order sorted term has a well defined least sort; this can simplify the implementation of overloaded OSA. Here is the formal definition:

Definition 2.42 An order sorted signature $(S, \leq, \Sigma)$ is **regular** iff it is monotone, and given $\sigma \in \Sigma_{w_1,s_1}$ and $w_0 \leq w_1$, there is a least rank $\langle w, s\rangle$ such that $w_0 \leq w$ and $\sigma \in \Sigma_{w,s}$. Also $(S, \leq, \Sigma)$ is **coherent** iff it is locally filtered and regular. □

A weaker condition that is necessary and sufficient for all terms to have a least sort parse is given in [41]. In essence, the **regular OSA** of [41] allows "multiple universes," one for each connected component of the sort hierarchy, without bothering whether they overlap. However, the programme of general algebra can be carried out in much greater generality than this. In fact, [34] emphasises that overloaded OSA can be developed for arbitrary locally filtered signatures; in particular, initial algebras exist for signatures that are neither regular nor monotone. In fact, all the standard results of general algebra carry through for any locally filtered signature, and this extends to signatures of non-monotonicities as well. An important technical result about the loose semantics of overloaded OSA, which also extends to non-monotonicities, is that any variety of algebras is equivalent (in the categorical sense) to a quasi-variety of many sorted algebras. This result implies that overloaded OSA has all the nice mathematical properties of MSA; for example, it can be used to prove the initiality, Birkhoff variety and quasi-variety theorems.

One of the interesting recent developments in the theory of OSA is by Hubert Comon [17] who showed that OSA specifications can be represented as bottom-up tree automata. The redundancy of the regularity hypothesis follows easily from this representation too. Moreover, the representation of OSA specifications as bottom-up tree automata proves to be very effective as an implementation technique, the regularity condition being redundant at the level of implementation too.

Given an order sorted signature $(S, \leq, \Sigma)$, the Σ-algebras and their homomorphisms form a category $\mathbb{A}lg_\Sigma$. This is the category of models for OSA. The domains are the many sorted sets. We emphasise that the domains for OSA should *not* have an order sorted structure. This idea is supported by the way OSA is implemented; at the theory level, the necessity to work with many sorted domains rather than order sorted domains will become more transparent later. The forgetful functor $\mathcal{U} : \mathbb{A}lg_\Sigma \to \mathbb{S}et^S$ forgets both the algebraic and the order sorted structure.

Other approaches to OSA could be treated in a similar manner. For a recent comparative survey on different approaches on OSA see [34].

2.3.3 Horn clause logics

The model theory of equational logics has an algebraic nature due to the absence of predicates (relational symbols). This is a big advantage over model theories involving

relations, since powerful and elaborate algebraic methods can be used (see [33, 41] for the semantics of programming languages). However, it is well known that Horn clause logics (abbreviated *HCL*), for example, do not lack nice semantical properties like completeness and the existence of initial models. Moreover, the way these properties are obtained has a strong algebraic flavour [39]. This shows that Horn clause logics somehow have an algebraic character.

Theorem 2.43 below describes an embedding of the category of models of any first order signature as a retract of the category of algebras of the algebraic signature obtained from the original first order signature by turning the predicates into operations. The idea of interpreting the predicates as 'boolean valued' operations is hardly new. It has even been used for promoting narrowing as an operational semantics for logic programming [19]. However, our approach is slightly different, because from the very beginning we avoid a full boolean structure on the new sort of truth values. Moreover, our approach emphasises the model theory side (Theorem 2.43). The result is an effective method for applying algebraic techniques to a large class of model theoretic problems in logic programming. For example, the construction of initial models, and more generally of free models of logic programs [39], follows immediately from the well known construction of initial and free algebras (see [41, 45], etc). The same principle applies to free extensions along theory morphisms, which were suggested in [39] as a semantic basis for constraint logic programming.

Recall (e.g., from [33]) that a (**many sorted**) **first order signature** is a triple (S, Σ, Π) such that (S, Σ) is a many sorted signature in the sense of Definition 2.40, and Π is an S^+-indexed family of sets of **predicate** or **relation** symbols. A **morphism** $(f, g, k)\colon (S, \Sigma, \Pi) \to (S', \Sigma', \Pi')$ between two first order signatures consists of an equational signature morphism (f, g) together with an S^+-indexed family of maps $k_w : \Pi_w \to \Pi'_{f^+(w)}$ on predicate symbols.[30] A **model** M of a first order signature (S, Σ, Π) consists of a Σ-algebra structure in the sense of Definition 2.40, together with an interpretation $\pi_M \subseteq M^w$ for each predicate symbol $\pi \in \Pi_w$ as a relation on the carriers. A **morphism** $h\colon M \to M'$ between (S, Σ, Π)-models M and M' is a Σ-homomorphism such that for any predicate symbol $\pi \in \Pi_{s_1 \ldots s_n}$, if $m \in \pi_M$ then $h(m) \in \pi_{M'}$.

For a first order signature (S, Σ, Π), let $\mathbb{M}od_{S,\Sigma,\Pi}$ denote the category of (S, Σ, Π)-models and their morphisms. We will often write (Σ, Π) for (S, Σ, Π), leaving the sort set implicit.

Theorem 2.43 Given a many sorted first order signature (S, Σ, Π), consider an algebraic signature $(S^b, \Sigma^b \cup \Pi^b)$ defined in the following way:

- S^b is S plus a new sort b,
- Π^b is a collection of new operation symbols $\{\pi^b \mid \pi \in \Pi\}$ such that $\pi^b \in \Pi_{s_1 \ldots s_n, b}$ whenever π is an $s_1 \ldots s_n$-ary relational symbol, and
- Σ^b is just Σ plus a new constant $\mathbf{t}$ of sort b.

Then

1. there is a forgetful functor $\mathcal{H}_{\Sigma,\Pi}\colon \mathbb{A}lg_{\Sigma^b \cup \Pi^b} \to \mathbb{M}od_{\Sigma,\Pi}$ such that for all $\pi \in \Pi$, $a \in \pi_{\mathcal{H}_{\Sigma,\Pi}(A)}$ iff $\pi^b_A(a) = \mathbf{t}_A$,

[30] Here f^+ is f^* restricted to non-empty strings.

2. $\mathcal{H}_{\Sigma,\Pi}$ has a left adjoint left inverse[31] $\mathcal{E}_{\Sigma,\Pi}$, and

3. there is a translation $\alpha_{\Sigma,\Pi}$ of (Σ,Π)-Horn clauses to $(\Sigma^b \cup \Pi^b)$-conditional equations that regards every Σ-equation as a Σ^b-equation and maps every atom $\pi(s)$ to the $(\Sigma^b \cup \Pi^b)$-equation $\pi^b(s) = \mathbf{t}$, such that for any Horn clause ϕ and any $(\Sigma^b \cup \Pi^b)$-algebra A,

$$A \models_{\Sigma^b \cup \Pi^b} \alpha_{\Sigma,\Pi}(\phi) \text{ iff } \mathcal{H}_{\Sigma,\Pi}(A) \models_{\Sigma,\Pi} \phi.$$

Proof: We omit the proof of 1. For 2.,it is enough to define $\mathcal{E}_{\Sigma,\Pi}$ on models (its definition on model morphisms is obtained from the general categorical construction of left adjoints from universal arrows; see [64]). Thus given any (Σ,Π)-model $A = ((A_s)_{s\in S}, (\sigma_A)_{\sigma\in\Sigma}, (\pi_A)_{\pi\in\Pi})$, we have to build a $(\Sigma^b \cup \Pi^b)$-algebra $\mathcal{E}_{\Sigma,\Pi}(A)$ which is free with respect to the forgetful functor $\mathcal{H}_{\Sigma,\Pi}$.

$$\begin{tikzcd} A \arrow[rr, equal] \arrow[dr, "h"'] & & \mathcal{H}(\mathcal{E}(A)) \arrow[dl, "\mathcal{H}(h')"] \\ & \mathcal{H}(B) & \end{tikzcd}$$

The carrier of $\mathcal{E}_{\Sigma,\Pi}(A)$ is the same as the carrier of A, except that a new carrier for the sort b is defined by

$$A_b = \{\mathbf{t}_A\} \cup \{\langle \pi, a\rangle \mid \pi \in \Pi \text{ and } a \notin \pi_A\}.$$

The interpretations of the Σ-operation symbols are those of A, and for each $\pi \in \Pi_{s_1 \ldots s_n}$. Define

$$\pi^b_A(a) = \mathbf{t}_A \text{ if } a \in \pi_A, \text{ otherwise } \pi^b_A(a) = \langle \pi, a\rangle.$$

Given any $(\Sigma^b \cup \Pi^b)$-algebra B and any (Σ,Π)-model morphism $h: A \to \mathcal{H}_{\Sigma,\Pi}(B)$, there is exactly one $(\Sigma^b \cup \Pi^b)$-morphism h' from $\mathcal{E}_{\Sigma,\Pi}(A)$ to B extending h (see the above diagram). Of course, $h'_s = h_s$ for every $s \in S$, $h'_b(\mathbf{t}_A) = \mathbf{t}_B$, and $h'_b(\langle \pi, a\rangle)$ is $\pi^b_B(h(a))$ for each $a \notin \pi_A$. This means that $\mathcal{E}_{\Sigma,\Pi}(A)$ is the free $(\Sigma^b \cup \Pi^b)$-algebra over the (Σ,Π)-model A. Notice that $\mathcal{H}_{\Sigma,\Pi}(\mathcal{E}_{\Sigma,\Pi}(A)) = A$.

3. This reduces to showing that for any $(\Sigma^b \cup \Pi^b)$-algebra A, any tuple s of terms in $T_\Sigma(X)$ and any valuation $v: X \to A$, $v^\sharp(s) \subseteq \pi_{\mathcal{H}_{\Sigma,\Pi}(A)}$ iff $v^\sharp(\pi^b(s)) = \mathbf{t}_A$. This holds because $v^\sharp(\pi^b(s)) = \pi^b_A(v^\sharp(s))$. □

Fact 2.44 $\mathcal{H}_{\Sigma,\Pi}$ is natural in (Σ,Π), i.e., $\mathcal{H}$ is a natural transformation. □

Notice that in general the embedding functor $\mathcal{E}_{\Sigma,\Pi}$ is *not* natural in (Σ,Π). However, the naturality of $\mathcal{E}$ can be obtained by slightly modifying the algebraic signature corresponding to a first order signature (Σ,Π) whereby instead of the new sort b we introduce a new sort b_π together with a new constant $\mathbf{t}_\pi$ for each relation symbol π. Theorem 2.43 can be easily translated into this new framework.

[31] When composition is written in the diagrammatic order. In category theory textbooks where the composition of arrows is written in anti-diagrammatic order, e.g., [64], this is referred to as a right inverse.

The following result shows that free models in HCL (more generally, free extensions along HCL theory morphisms) are in fact free algebras regarded as models through the forgetful functor $\mathcal{H}$. This remark includes the important case of Herbrand models, which are in fact term models with the empty interpretation for the relational symbols.

Corollary 2.45 1. Let (S, Σ, Π) be a first-order signature and let Γ be a set of Horn clauses over this signature. Then for every S-sorted set X, the free model $M_\Gamma(X)$ over X in the quasi-variety $\mathbb{M}od_\Gamma$ determined by Γ is the image of the free $(\Sigma^b \cup \Pi^b, \alpha_{\Sigma,\Pi}(\Gamma))$-algebra over X under the forgetful functor $\mathcal{H}$.

2. Let $\Phi\colon (S, \Sigma, \Pi, \Gamma) \to (S', \Sigma', \Pi', \Gamma')$ be a morphism of theories in many-sorted Horn clause logic with equality. Then every Γ-model M has a free extension M' along Φ which can be obtained as the free extension in MSA and translated back to HCL under $\mathcal{H}$.

Proof: 1. First notice that by Theorem 2.43, $\mathcal{H}_{\Sigma,\Pi}$ maps the quasi-variety $\mathbb{A}lg_{\Sigma^b \cup \Pi^b, \alpha(\Gamma)}$ to $\mathbb{M}od_\Gamma$ and that $\mathcal{E}_{\Sigma,\Pi}$ maps $\mathbb{M}od_\Gamma$ to $\mathbb{A}lg_{\Sigma^b \cup \Pi^b, \alpha(\Gamma)}$.

$$\begin{tikzcd}
\mathbb{A}lg_{\Sigma^b \cup \Pi^b, \alpha(\Gamma)} \arrow[r, "\mathcal{H}"] \arrow[d] & \mathbb{M}od_\Gamma \arrow[d] \\
\mathbb{S}et^{S^b} \arrow[r] & \mathbb{S}et^S
\end{tikzcd}$$

Next, the forgetful functor $\mathbb{A}lg_{\Sigma^b \cup \Pi^b, \alpha(\Gamma)} \to \mathbb{S}et^S$ is right adjoint as the composite of the right adjoint forgetful functors $\mathbb{A}lg_{\Sigma^b \cup \Pi^b, \alpha(\Gamma)} \to \mathbb{S}et^{S^b}$ and $\mathbb{S}et^{S^b} \to \mathbb{S}et^S$. The left adjoint to $\mathbb{S}et^{S^b} \to \mathbb{S}et^S$ just adds to the S-sorted sets the empty set as the carrier of sort b.

On the other side of the diagram, the free $(\Sigma^b \cup \Pi^b, \alpha(\Gamma))$-algebra is obtained as $\mathcal{E}_{\Sigma,\Pi}(M_\Gamma(X))$. The conclusion follows from the fact that $\mathcal{E}_{\Sigma,\Pi}; \mathcal{H}_{\Sigma,\Pi} = 1$.

2. This uses the same argument as the proof of the previous part of this corollary, by noticing that Φ induces a morphism of algebraic theories $\Phi^b\colon (\Sigma^b \cup \Pi^b, \alpha_{\Sigma,\Pi}(\Gamma)) \to (\Sigma'^b \cup \Pi'^b, \alpha_{\Sigma',\Pi'}(\Gamma'))$ in the obvious way.

$$\begin{tikzcd}
\mathbb{A}lg_{\Sigma'^b \cup \Pi'^b, \alpha(\Gamma')} \arrow[r, "\mathcal{H}'"] \arrow[d, "\mathbb{A}lg(\Phi^b)"'] & \mathbb{M}od_{\Gamma'} \arrow[d, "\mathbb{M}od(\Phi)"] \\
\mathbb{A}lg_{\Sigma^b \cup \Pi^b, \alpha(\Gamma)} \arrow[r, "\mathcal{H}"'] & \mathbb{M}od_\Gamma
\end{tikzcd}$$

The free extension of M along Φ is the same as $\mathcal{H}_{\Sigma',\Pi'}((\mathcal{E}_{\Sigma,\Pi}(M))^\$)$, where $(\mathcal{E}_{\Sigma,\Pi}(M))^\$$ is the free extension of $\mathcal{E}_{\Sigma,\Pi}(M)$ along Φ^b. $\square$

The final remark of this subsection is that given a first order signature (Σ, Π), the category of models for HCL can be taken as $\mathbb{A}lg_{\Sigma^b \cup \Pi^b}$, and the category of domains should be taken as $\mathbb{S}et^S$. Notice that in HCL, unlike MSA, the forgetful functor from the category of models to the category of domains (i.e., $\mathbb{A}lg_{\Sigma^b \cup \Pi^b} \to \mathbb{S}et^S$) is *not* monadic.

2.3.4 Equational logic modulo axioms

Equational deduction modulo a set of axioms (abbreviated *ELM*) becomes vital when dealing with non-orientable equations in the context of rewriting. A detailed exposition of the subject is given in [30, 56, 20, 63]. Although in practice non-orientable axioms are mostly unconditional[32], there is no theoretical reason to exclude the case of equational deduction modulo a set of conditional equations.

Definition 2.46 [30] Given a MSA signature (S, Σ) and a collection E of Σ-equations, a **Σ-term modulo** E is just an element t of $T_{\Sigma,E}(X)$ (i.e., the quotient of the term algebra $T_\Sigma(X)$ determined by E). □

Equational deduction modulo E is based on a generalisation of the usual concepts of MSA to "concepts modulo E", including the inference rules. In order to have a model theory for equational logic modulo E, we need an adequate notion of model for this type of logic. It is therefore natural to consider $\mathbb{A}lg_{\Sigma,E}$ as the category of models for the equational logic modulo E. This idea is consistent with having "algebras modulo axioms" as models for ELM. The category of domains is the category $\mathbb{S}et^S$ of S-sorted sets and functions. The forgetful functor $\mathcal{U} : \mathbb{A}lg_{\Sigma,E} \to \mathbb{S}et^S$ forgets both the axioms and the algebraic structure of the algebras.

Example 2.47 The logic of Mosses's unified algebras from [75] can be regarded as equational logic modulo a conditional theory. All unified specifications of a given unified signature contain a core essentially consisting of Horn clauses. Unified algebras appear as models of this specification. □

2.3.5 Summary of Examples

The following table gives a summary of how the logical systems presented above fit our abstract model theoretic framework. We also include the case of constraint logics (abbreviated *CL*), which will be presented in detail in Chapter 6.

	$\mathbb{A}$ (cat. of models)	$\mathbb{X}$ (cat. of domains)	$\mathcal{U}$ forgets:
MSA	$\mathbb{A}lg_\Sigma$	$\mathbb{S}et^S$	algebraic structure
OSA	$\mathbb{A}lg_\Sigma$	$\mathbb{S}et^S$	algebraic structure + order sortedness
HCL	$\mathbb{A}lg_{\Sigma^b \cup \Pi^b}$	$\mathbb{S}et^S$	algebraic structure + sort b
ELM	$\mathbb{A}lg_{\Sigma,E}$	$\mathbb{S}et^S$	axioms + algebraic structure
CL	$(A{\downarrow}\mathbb{A}lg(\iota))$	$\mathbb{S}et^{S'}$	comma category structure + algebraic structure

It is possible to have any combination of any of these logical systems, such as order sorted Horn clause logic with equality. An interesting case is given by the logic underlying Eqlog, which combines all of the logical systems presented above; in particular, Eqlog's extensible constraint logic programming also involves CL.

[32] An interesting example of conditional non-orientable axiom is provided by idempotence, sometimes given in its conditional form: `x + y = x if x = y .`

3 Category-based Equational Deduction

In this chapter we develop a categorical proof theory for equational logics and we prove its completeness with respect to the model theory. The following technical assumption underlies the whole chapter:

> **[DeductionFramework]: BasicFramework** + the category $\mathbb{A}$ of models has pullbacks and coequalisers.

The proof theory is based on a categorical abstraction of some basic concepts which constitute the very essence of equational logic and universal algebra. This includes notions like congruence, term algebra, substitution, equation (represented here as parallel pairs of arrows, hardly a new idea, see [51, 52]), and satisfaction. Following the main idea of [18], the quantification of equations is abstracted from variables to models, and as a result, valuations are abstracted from simple assignments of the variables to model morphisms. This new level of abstraction is based on a semantic view of terms as elements of the carrier of a free model, rather than as tree-like syntactical constructs. The fact that equational deduction can be fully extended to this level without any fundamental difficulty illustrates the precedence of semantics over syntax for equational logics. The semantic architecture of a particular equational logic system seems to be the only thing that really matters for its deductive system. A technical consequence is the possibility of developing the main core of the equational proof theory without using freeness.

3.1 Congruences

The construction of quotient models and the formulation of a complete system of inference rules for category-based equational logics both rely upon a notion of congruence.

Definition 3.1 Let A be an arbitrary model. The binary relation Q on the underlying domain of A is a **congruence** iff it is a kernel of a model morphism, i.e., iff there is a model morphism ϕ in $\mathbb{A}$ such that $Q = \mathcal{U}(ker\phi)$. The **quotient** of A by Q is the coequaliser of $ker\phi$. Its target model is denoted $A/_Q$ and is also sometimes called the quotient of A. □

Fact 3.2 Any model congruence is a domain equivalence. □

Lemma 3.3 Let $Q = C\mathcal{U}$ be a congruence on a model A. Then $C = ker(coeq(C))$.

Proof: $C \subseteq ker(coeq(C))$ by the universal property of kernels. Let C be $ker\phi$ for some model morphism ϕ. There exists a [unique] h such that $coeq(ker\phi); h = \phi$. But $ker(coeq(ker\phi)) \subseteq ker(coeq(ker\phi); h)$ and therefore $ker(coeq(C)) \subseteq C = ker\phi$. □

The idea of relating congruences to kernels of model morphisms has a long tradition in general algebra, including MSA and OSA. In the context of Horn clause logics (see Section 2.3.3), the previous definition gives an appropriate notion of congruence for model theories with relational symbols [39].

Definition 3.4 Let Q be a binary relation on the underlying domain of a model A. Then the **congruence closure of** Q is the least congruence on A containing Q; it may be denoted $\mathbf{C}(Q)$. $\square$

Definition 3.5 Suppose the congruence closures of binary relations exist in $\mathbb{A}$ and $\mathbb{X}$ has unions of binary relations. Then the forgetful functor $\mathcal{U}: \mathbb{A} \to \mathbb{X}$ is **finitary** iff

$$\mathbf{C}(Q) = \bigcup\{\mathbf{C}(Q_0) \mid Q_0 \subseteq Q \ \text{ finite}\}$$

for any model A and any binary relation Q on the underlying domain of A. $\square$

All forgetful functors from models to domains presented as examples in Section 2.3 are finitary. This is due to the fact that all operation and relational symbols involved take only a finite number of arguments, as will be seen in Section 3.5.

3.2 Equations, Queries and Satisfaction

Traditionally, equations are pairs of terms constructed from the symbols of a signature plus some variables. In the context of many sorted equational logic the importance of explicit quantification was emphasized for the first time by Goguen and Meseguer [37]. The survey [62] shows that explicit qunatification adds a key syntactic information in the case of constraints and unification. In this way, the quantifier becomes part of the concept of equation.

Although terms are syntactic constructs, from a model theoretic perspective they are just elements of the free term model over the set of quantified variables. Any valuation of the variables into a model extends uniquely to a model morphism evaluating both sides of the equation. Thus a more semantic treatment of quantification regards quantifiers as models rather than as collections of variables, and regards valuations as model morphisms rather than as evaluations of variables into models. This has already been done in [18] in the context of many sorted algebra. This non-trivial generalisation of the notions of sentence and satisfaction in equational logic also supports the extension of the equational proof theory along the same lines without any difficulty. Moreover, this semantic approach to equational logic brings a sense of simplicity and unity to the proof theory, which has somehow been lost in the more traditional syntactical frameworks.

Definition 3.6 Let A be any model. Then a $\mathcal{U}$**-identity** on A is a binary relation $k \xrightarrow{\langle s,t\rangle} A\mathcal{U}$ on the underlying domain of A. An identity $\langle s,t\rangle$ in A is **satisfied** in a model B with respect to a model morphism $h: A \to B$ iff $s;h\mathcal{U} = t;h\mathcal{U}$. This is denoted $B \models \langle s,t\rangle[h]$.

A $\mathcal{U}$**-equation** is a universally quantified expression $(\forall A)\langle s,t\rangle$ where A is a model representing the quantifier and $\langle s,t\rangle$ is an identity in A. A model B satisfies $(\forall A)\langle s,t\rangle$ iff B satisfies the identity $\langle s,t\rangle$ for all model morphisms $h: A \to B$.

A $\mathcal{U}$**-query** is an existentially quantified expression $(\exists A)\langle s,t\rangle$ where A is a model representing the quantifier and $\langle s,t\rangle$ is an identity in A. A **solution** of $(\exists A)\langle s,t\rangle$ in a model B is any model morphism $h: A \to B$ for which $\langle s,t\rangle$ is satisfied in B with respect to h. When B is a free model, h is called an **solution form**. $\square$

The notion of $\mathcal{U}$-equation (query) deals with *families of equations (queries)*, rather than single equations (queries), as sentences. This agrees with Rodenburg's work [82]

showing that equational logic with conjunction satisfies the Craig Interpolation Property[33] whereas normal equational logic does not. Our terminology is influenced by Lassez who replaced the traditional logic programming terminology of *computed answer substitution* by that of *solved form* [67]. The modern terminology has the advantage to allow more flexibility for the representations of solutions (i.e., staying away of from the traditional representations of solution forms as substitutions is very beneficial at the level of operational semantics) and is also more intuitive (i.e., solutions in different models can be obtained by interpreting the solutions forms).

Example 3.7 OSA equations Let $(S, \leq, \Sigma)$ be a coherent (i.e., regular and locally filtered) order sorted signature and let X be an S-sorted set of variables. The collection of all well-formed Σ-terms over X, denoted $\mathcal{T}_\Sigma(X)$, has a canonical structure as an order sorted Σ-algebra.

An **order sorted equation** $(\forall X)t =_s t'$ is an universally quantified pair of terms having the same sort (i.e., $t, t' \in (\mathcal{T}_\Sigma(X))_s$). Any parallel pair of many sorted functions $k \to \mathcal{T}_\Sigma(X)$ defines a many sorted family of such equations.

Given an order sorted Σ-algebra A, any valuation $v : X \to A$ of variables X into A extends uniquely to an order sorted Σ-morphism $v^\sharp : \mathcal{T}_\Sigma(X) \to A$ giving the denotations in A for the terms in $\mathcal{T}_\Sigma(X)$. A **satisfies** the identity $t =_s t'$ with respect to the valuation v iff t and t' have the same denotation, i.e., $v^\sharp(t) = v^\sharp(t')$. When dealing with a many sorted family of equations $k \xrightarrow{\langle t,t'\rangle} \mathcal{T}_\Sigma(X)$, the satisfaction of $\langle t, t'\rangle$ by A with respect to the valuation v means $t; v^\sharp\mathcal{U} = t'; v^\sharp\mathcal{U}$.

It appears that this definition of order sorted equations is more restrictive than the one given by Goguen and Meseguer [41]. However, the two can be shown to agree. In [41], an order sorted equation $(\forall X)t = t'$ is a universally quantified pair of terms having the least sorts $LS(t)$ and $LS(t')$ in the *same connected component*. An order sorted algebra satisfies $t = t'$ with respect to the valuation v iff $v^\sharp_{LS(t)}(t) = v^\sharp_{LS(t')}(t')$. Let's consider w a common supersort of both $LS(t)$ and $LS(t')$. Then for any order sorted algebra A and any valuation $h : X \to A$, we have $A \models t = t'[h]$ iff $A \models t =_w t'[h]$.

This definition of order sorted equations also holds without assuming coherence of the signature by using annotated terms (or parse trees). □

Example 3.8 Let Σ be an algebraic signature and and let E be a collection of Σ-equations. An **equation modulo** E [30], denoted $(\forall X)t =^E t'$, is a universally quantified pair of elements in $T_{\Sigma,E}(X)$ (i.e., t and t' are *terms modulo* E). Any parallel pair of functions $k \to T_{\Sigma,E}(X)$ defines a family of such equations. A (Σ, E)-algebra satisfies $t =^E t'$ for the valuation $v : X \to A$ iff $v^\sharp(t) = v^\sharp(t')$, where $v^\sharp$ is the unique extension of v to a Σ-homomorphism $T_{\Sigma,E}(X) \to A$. □

Definition 3.9 $(\forall A)\langle s', t'\rangle$ **if** $\langle s, t\rangle$ is a $\mathcal{U}$**-conditional equation** quantified by the model A, where $\langle s, t\rangle$ are the hypotheses of the conditional equation. A model B satisfies $(\forall A)\langle s', t'\rangle$ **if** $\langle s, t\rangle$ iff for any morphism $h : A \to B$, $s; h\mathcal{U} = t; h\mathcal{U}$ implies $s'; h\mathcal{U} = t'; h\mathcal{U}$. □

The following definition is a standard extension of the concept of satisfaction between models and sentences to satisfaction between sets of sentences:

Definition 3.10 A set Γ of equations **satisfies** the equation e, written $\Gamma \models e$, iff any model satisfying Γ also satisfies e. □

[33] The Craig Interpolation Property is an important semantic property for logical systems [21].

3.3 Completeness

Our approach to the completeness of category-based equational deduction follows the traditional approach (probably originating with Birkhoff's work on universal algebra [10]), in that the central concept is the congruence determined by an arbitrary collection Γ of [conditional] equations on an arbitrary model A. The key to the completeness result is to regard this congruence in two different ways: the first way is as the collection of unconditional equations quantified by A that can be *syntactically inferred* from Γ, while the second is as the collection of unconditional equations quantified by A that are *semantic consequences* of Γ. Because of the semantic treatment of equation and satisfaction underlying this work, there is no distinction between the congruence determined by Γ on the free models (this case corresponds to the traditional treatments of the completeness of equational logics) and on other models. This is very important in the context of the semantics of constraint logic programming given in Chapter 6, because it involves "built-in models" that are not term models in general.

The rôle of (categorical) projectivity was first pointed out in [18], and in the presence of a left adjoint to the forgetful functor from models to domains, it is directly related to a categorical formulation of the Axiom of Choice for domains. Despite the high level of generality and abstraction, the rules of inference for category-based equational deduction are made gradually more explicit. They can be easily recognised even in the most abstract formulation of completeness. In the case of conditional category-based equational deduction, the most syntactic formulation of the completeness result depends directly on two finiteness conditions. The first one requires that the hypotheses of the equations should be finite, while the second corresponds in practice to finite arities for the operator symbols.

Definition 3.11 Let Γ be a set of conditional equations. A congruence C on A is **closed under Γ-substitutivity** iff for all $(\forall B)\langle s', t'\rangle$ **if** $\langle s, t\rangle$ in Γ and any morphism $h\colon B \to A$, $\langle s; h\mathcal{U}, t; h\mathcal{U}\rangle \subseteq C$ implies $\langle s'; h\mathcal{U}, t'; h\mathcal{U}\rangle \subseteq C$. □

Proposition 3.12 Let $h\colon A \to M$ be a model morphism. Then $M \models \Gamma$ implies $ker(h)$ is closed under Γ-substitutivity.

Proof: Let $(\forall B)\langle s', t'\rangle$ **if** $\langle s, t\rangle$ be a conditional equation in Γ and let $\phi\colon B \to A$ be any model morphism.

$$B \xrightarrow{\phi} A \xrightarrow{h} M$$

Suppose $\langle s; \phi\mathcal{U}, t; \phi\mathcal{U}\rangle \subseteq ker(h)$. Then $s; \phi\mathcal{U}; h\mathcal{U} = t; \phi\mathcal{U}; h\mathcal{U}$. But $\phi; h\colon B \to M$ and M is closed under Γ-substitutivity, therefore $s'; (\phi; h)\mathcal{U} = t'; (\phi; h)\mathcal{U}$. This means that $\langle s'; \phi\mathcal{U}, t'; \phi\mathcal{U}\rangle \subseteq ker(h)$. □

Corollary 3.13 Let C be a congruence on a model A. Then $A/_C \models \Gamma$ implies that C is closed under Γ-substitutivity. □

The following definition is a weakening of the traditional concept of projective object in category theory (see [64]):

Definition 3.14 An object A in a category $\mathbb{C}$ is **coequaliser projective** iff for any coequaliser $e\colon B \to M$ in $\mathbb{C}$ and for any map $g\colon A \to M$ there exists a map $f\colon A \to B$ such that $f; e = g$.

$$\begin{tikzcd} B \arrow[r, "e"] & M \\ & A \arrow[u, "g"'] \arrow[ul, "f"] \end{tikzcd}$$

□

Term models are always coequaliser projective. This will be proved later in connection with a categorical formulation of the Axiom of Choice for the category of domains (see Section 3.5.3).

Proposition 3.15 If all quantifiers in Γ are coequaliser projective, then a congruence C on a model A is closed under Γ-substitutivity iff $A/_C \models \Gamma$.

Proof: Because of Corollary 3.13, we only have to prove that $A/_C \models \Gamma$ if C is closed under Γ-substitutivity. Assume C is closed under Γ-substitutivity. Let $(\forall B)\langle s', t'\rangle$ **if** $\langle s, t\rangle$ be any conditional equation in Γ and let $h\colon B \to A/_C$ be any model morphism. Suppose $s; h\mathcal{U} = t; h\mathcal{U}$. Because B is coequaliser projective, there exists $h'\colon B \to A$ such that $h'; (coeqC) = h$. By Lemma 3.3, $C = ker(coeqC)$, therefore $\langle s; h'\mathcal{U}, t; h'\mathcal{U}\rangle \subseteq C\mathcal{U}$. Since C is closed under Γ-substitutivity, $\langle s'; h'\mathcal{U}, t'; h'\mathcal{U}\rangle \subseteq C\mathcal{U}$. $s'; h\mathcal{U} = t'; h\mathcal{U}$ follows immediately from $h = h'; (coeqC)$. □

Definition 3.16 For any model A, let $\equiv^A_\Gamma$ denote the least congruence on A closed under Γ-substitutivity. □

Corollary 3.17 <u>Completeness Theorem</u>
If $\equiv^A_\Gamma$ exists and the quantifiers in Γ are coequaliser projective, then

1. $A/_{\equiv^A_\Gamma}$ is the free Γ-model over A, and
2. $\Gamma \models (\forall A)\langle s, t\rangle$ iff $s \equiv^A_\Gamma t$.

Proof: 1. Let $A \xrightarrow{f} M$ be a model morphism such that $M \models \Gamma$. By Proposition 3.12, $ker(f)$ is closed under Γ-substitutivity. Because $\equiv^A_\Gamma$ is the least congruence on A closed under Γ-substitutivity, $\equiv^A_\Gamma \subseteq ker(f)$, which means that f equalises $\equiv^A_\Gamma$. We conclude there exists a unique map $A/_{\equiv^A_\Gamma} \xrightarrow{f'} M$ such that $e; f' = f$, where e denotes the coequliser $coeq(\equiv^A_\Gamma)$.

$$\begin{tikzcd} A \arrow[r, "coeq\equiv^A_\Gamma"] \arrow[dr, "f"'] & A/_{\equiv^A_\Gamma} \arrow[d, "f'"] \\ & M \end{tikzcd}$$

2. From Proposition 3.15 we know that $A/_{\equiv^A_\Gamma} \models \Gamma$. Suppose $s \equiv^A_\Gamma t$ and consider a Γ-model M and any model morphism $A \xrightarrow{f} M$. By 1., there is $A/_{\equiv^A_\Gamma} \xrightarrow{f'} M$ such that $e; f' = f$. $s; f\mathcal{U} = t; f\mathcal{U}$ since $s; e\mathcal{U} = t; e\mathcal{U}$. We thus conclude that $\Gamma \models (\forall A)\langle s, t\rangle$.

Conversely, consider $e\colon A \to A/_{\equiv^A_\Gamma}$. Since $\Gamma \models (\forall A)\langle s, t\rangle$, $s; e = t; e$, therefore $\langle s, t\rangle \subseteq ker(e) = \equiv^A_\Gamma$. □

The following two results provide an inference-based version of the completeness theorem for equational logics. This relies upon a syntactic deduction oriented construction of $\equiv_\Gamma^A$. In the case of unconditional equations, $\equiv_\Gamma^A$ has a rather simple representation that shows that any category-based equational deduction is equivalent to a category-based equational deduction in which all applications of the substitutivity rule take place before any application of the congruence rule.[34]

These results are obtained under the following technical assumption:

> **[ConcreteDeductionFramework]: DeductionFramework** + the category $\mathbb{X}$ of domains has unions of binary relations and ω colimits + congruence closures exist in $\mathbb{A}$.

However, the following result doesn't use the existence of ω-colimits in the category of domains:

Proposition 3.18 If Γ contains only *unconditional* equations, then $\equiv_\Gamma^A$ exists and

$$\equiv_\Gamma^A = \mathbf{C}(\bigcup\{\langle s; f\mathcal{U}, t; f\mathcal{U}\rangle \mid (\forall B)\langle s, t\rangle \in \Gamma, f \in \mathbb{A}(B, A)\}).$$

Proof: $\mathbf{C}(\bigcup\{\langle s; f\mathcal{U}, t; f\mathcal{U}\rangle \mid (\forall B)\langle s, t\rangle \in \Gamma, f \in \mathbb{A}(B, A)\})$ is closed under Γ-substitutivity and is a congruence by definition. Consider another congruence C closed under Γ-substitutivity on the model A. Then

$$\bigcup\{\langle s; f\mathcal{U}, t; f\mathcal{U}\rangle \mid (\forall B)\langle s, t\rangle \in \Gamma, f \in \mathbb{A}(B, A)\} \subseteq C$$

since $\langle s; f\mathcal{U}, t; f\mathcal{U}\rangle \subseteq C$ for any $(\forall B)\langle s, t\rangle \in \Gamma$ and any $f \in \mathbb{A}(B, A)$. Therefore

$$\mathbf{C}(\bigcup\{\langle s; f\mathcal{U}, t; f\mathcal{U}\rangle \mid (\forall B)\langle s, t\rangle \in \Gamma, f \in \mathbb{A}(B, A)\}) \subseteq C$$

by taking the congruence closure. □

When Γ contains proper conditional equations, $\equiv_\Gamma^A$ can be constructed in the limit by alternating the applications of the rule of congruence and of the rule of substitutivity:

Proposition 3.19 Assume **ConcreteDeductionFramework**. If the forgetful functor $\mathcal{U} : \mathbb{A} \to \mathbb{X}$ is finitary and the hypotheses of all conditional equations in Γ are finite, then the least congruence on A closed under Γ-substitutivity exists.

Proof: Define $\langle s_0, t_0\rangle$ to be $\bigcup\{\langle s; f\mathcal{U}, t; f\mathcal{U}\rangle \mid (\forall B)\langle s, t\rangle \in \Gamma, f \in \mathbb{A}(B, A)\}$, and for each $n \in \omega$, define

- $\langle s_{2n+1}, t_{2n+1}\rangle$ to be $\mathbf{C}\langle s_{2n}, t_{2n}\rangle$,

  and define

- $\langle s_{2n+2}, t_{2n+2}\rangle$ to be
  $\langle s_{2n+1}, t_{2n+1}\rangle \cup \bigcup\{\langle s'; h\mathcal{U}, t'; h\mathcal{U}\rangle \mid (\forall B)\langle s', t'\rangle$ **if** $\langle s, t\rangle \in \Gamma, B \xrightarrow{h} A, \langle s; h\mathcal{U}, t; h\mathcal{U}\rangle \subseteq \langle s_{2n+1}, t_{2n+1}\rangle\}$.

[34] The rules of congruence and substitutivity are discussed at the end of this subsection.

Observe that for each $n \in \omega$, $\langle s_n, t_n \rangle \subseteq \langle s_{n+1}, t_{n+1} \rangle$. The union $\equiv_\Gamma^A = \bigcup_{n \in \omega} \langle s_n, t_n \rangle$ could be realised as an ω-colimit of the inclusion chain $\langle s_0, t_0 \rangle \subseteq \langle s_1, t_1 \rangle \subseteq \ldots$ in the comma category $(\Delta_{\mathbb{X}} \downarrow A\mathcal{U})$ (the ω-completeness of $\mathbb{X}$ lifts to the comma category $(\Delta_{\mathbb{X}} \downarrow A\mathcal{U})$). We shall prove that $\equiv_\Gamma^A$ is the least congruence on A closed under Γ-substitutivity. Because $\mathcal{U}$ is finitary,

$$\mathbf{C}(\equiv_\Gamma^A) = \bigcup \{\mathbf{C}\langle S, T \rangle \mid \langle S, T \rangle \subseteq \equiv_\Gamma^A \quad \text{finite}\}$$

For each finite $\langle S, T \rangle \subseteq \equiv_\Gamma^A$ there exists $n \in \omega$ such that $\langle S, T \rangle \subseteq \langle s_n, t_n \rangle$. Then $\mathbf{C}\langle S, T \rangle \subseteq \mathbf{C}\langle s_n, t_n \rangle \subseteq \langle s_{n+2}, t_{n+2} \rangle \subseteq \equiv_\Gamma^A$, therefore $\mathbf{C}(\equiv_\Gamma^A) \subseteq \equiv_\Gamma^A$, which means that $\equiv_\Gamma^A$ is a congruence.

For any $(\forall B)\langle s', t' \rangle$ **if** $\langle s, t \rangle$ in Γ and any model morphism $h\colon B \to A$, if $\langle s; h\mathcal{U}, t; h\mathcal{U} \rangle \subseteq \equiv_\Gamma^A$, then because $\langle s, t \rangle$ is finite, there exists $n \in \omega$ such that $\langle s; h\mathcal{U}, t; h\mathcal{U} \rangle \subseteq \langle s_n, t_n \rangle$. By the construction of the chain $\{\langle s_n, t_n \rangle\}_{n \in \omega}$, we have $\langle s'; h\mathcal{U}, t'; h\mathcal{U} \rangle \subseteq \langle s_{n+2}, t_{n+2} \rangle \subseteq \equiv_\Gamma^A$. This shows that $\equiv_\Gamma^A$ is closed under Γ-substitutivity.

Now consider an arbitrary congruence C on A closed under Γ-substitutivity. By induction on n, $\langle s_n, t_n \rangle \subseteq C$ for all $n \in \omega$. Therefore $\equiv_\Gamma^A \subseteq C$.

From all this we conclude that $\equiv_\Gamma^A$ is the least congruence on A closed under Γ-substitutivity. $\square$

Corollary 3.20 Assuming the **ConcreteDeductionFramework**, category-based equational logic is complete under the following two inference rules:

$$[\mathsf{congruence}] \quad \frac{(\forall A)\langle s, t \rangle}{(\forall A)\mathbf{C}\langle s, t \rangle}$$

$$[\mathsf{substitutivity}] \quad \frac{(\forall A)\langle s; h\mathcal{U}, t; h\mathcal{U} \rangle}{(\forall A)\langle s'; h\mathcal{U}, t'; h\mathcal{U} \rangle}$$

where $(\forall B)\langle s', t' \rangle$ **if** $\langle s, t \rangle$ is in Γ and $h\colon B \to A$ is any model morphism. $\square$

3.4 Herbrand's Theorem

Herbrand's Theorem provides mathematical foundations for logic programming. In this section we present a version of Herbrand's Theorem in our category-based framework, based on the categorical characterisation of Herbrand Universes as initial models for equational logic programs. This idea was first exploited in the context of order sorted Horn clause logic with equality by Goguen and Meseguer [39]. The results in this subsection can be seen as a category-based generalisation of the extension of their results to equational logics with projective models as quantifiers.

For this section only, we assume that the category $\mathbb{A}$ has an initial model; we denote it by $0_{\mathbb{A}}$. In the case of many sorted equational logic this is the initial algebra of ground terms.

Corollary 3.21 <u>Herbrand's Theorem</u> Assume the **ConcreteDeductionFramework** and that $\mathcal{U}$ is finitary and consider Γ a collection of conditional equations with finite hypotheses and coequaliser projective quantifiers. Then

1. the initial model of Γ exists, we denote it by 0_Γ, and
2. $\Gamma \models (\exists B)q$ iff $0_\Gamma \models (\exists B)q$ for any $\mathcal{U}$-query $(\exists B)q$ and any model B.

Proof: 1. From Proposition 3.19 and the first part of the Completeness Theorem.

2. Since 0_Γ is a Γ-model, $\Gamma \models (\exists B)q$ implies $0_\Gamma \models (\exists B)q$. For the converse, suppose that $0_\Gamma \models (\exists B)q$ and take any Γ-model M. Let $h\colon B \to 0_\Gamma$ be a solution for $(\exists B)q$ in 0_Γ. Let $!_M$ denote the unique model morphism $0_\Gamma \to M$. Then $h;!_M$ is a solution for $(\exists B)q$ in M. $\square$

At the end of the following section we present another version for Herbrand's Theorem that relies on the present one but provides foundations for solving queries using resolution and paramodulation-like techniques by directly relating the satisfyability of a query by a program to the existence of solution forms to the query. This is formulated in a context corresponding to 'non-empty sorts' in the case of many sorted logics [39]. The next definition gives a category-based formulation of this condition:

Definition 3.22 The forgetful functor $\mathcal{U}$ from the category $\mathbb{A}$ of models to the category $\mathbb{X}$ of domains **has non-empty sorts** iff for each domain $x \in |\mathbb{X}|$ there exists at least one map from x to the domain underlying the initial model $0_{\mathbb{A}}$. $\square$

Example 3.23 Consider an algebraic signature (S, Σ). The initial algebra for this signature is T_Σ, i.e., the algebra of ground terms. There exists at least one S-sorted function from any S-sorted X to T_Σ iff $T_{\Sigma,s} \neq \emptyset$ for all $s \in S$. A sufficient [but not necessary] condition is that for each sort $s \in S$, there is at least one constant of that sort, i.e., $\Sigma_{[],s} = \emptyset$. $\square$

3.5 Consequences of Freeness

So far, our development has avoided the use of freeness, corresponding to the existence of *term models* in the particular cases discussed in the preliminary chapter. By using this concept, we can further explicitate the inference rules for equational deduction by splitting the rule of congruence into equivalence (i.e., reflexivity + symmetry + transitivity) and closure under operations.

Moreover, by assuming freeness, we relate the projectivity condition on quantifiers to a condition on the category of domains corresponding to the Axiom of Choice. We can also see how the finitarity condition on the forgetful functor from models to domains boils down in practice to the finiteness of the arities of the model operations. Finally, in the presence of freeness, we can formulate and prove a more computational version for Herbrand's Theorem.

This section assumes the forgetful functor $\mathcal{U}$ has a left adjoint $\mathcal{F}$.

3.5.1 The existence of congruence closures

The congruence closure of any binary relation can be constructed in two steps strongly reminiscent of the rules of equivalence (i.e., reflexivity, symmetry and transitivity) and congruence (i.e., closure under "model operations") from equational logic [37, 41, 39].

Proposition 3.24 Let $k \xrightarrow{\langle s,t \rangle} A\mathcal{U}$ be a relation on the underlying domain of the model A. Then the congruence closure of $\langle s, t \rangle$ exists and it is constructed by the following steps:

- operations: define $s^\sharp$ and $t^\sharp$ to be the unique extensions of s and t, respectively, to model morphisms $k\mathcal{F} \to A$, and

- equivalence: let $\langle S, T\rangle$ be the kernel of $coeq\langle s^\sharp, t^\sharp\rangle$.

The congruence closure $\mathbf{C}\langle s, t\rangle$ is $\langle S\mathcal{U}, T\mathcal{U}\rangle$.

Proof: $\langle S\mathcal{U}, T\mathcal{U}\rangle$ is a congruence by construction as a kernel pair of a model morphism. Now let $\phi\colon A \to B$ be any model morphism. We have to prove that $\langle s, t\rangle \subseteq ker(\phi\mathcal{U})$ implies $\langle S\mathcal{U}, T\mathcal{U}\rangle \subseteq ker(\phi\mathcal{U})$.

$$\begin{tikzcd}
 & k\mathcal{F}\mathcal{U} \arrow[dr, shift left, "t^\sharp\mathcal{U}"] \arrow[dr, shift right, "s^\sharp\mathcal{U}"'] & \\
k \arrow[ur, "k\eta"] \arrow[rr, shift left, "s"] \arrow[rr, shift right, "t"'] & & A\mathcal{U}
\end{tikzcd}
\qquad
\begin{tikzcd}
k\mathcal{F} \arrow[rr, dashed] \arrow[dr, shift left, "t^\sharp"] \arrow[dr, shift right, "s^\sharp"'] & & ker(e) \arrow[dl, shift left, "S"] \arrow[dl, shift right, "T"'] \\
 & A \arrow[dl, "\phi"'] \arrow[dr, "e"] & \\
B & & B \arrow[ll, dashed, "\phi'"]
\end{tikzcd}$$

Then $\langle s, t\rangle \subseteq ker(\phi\mathcal{U})$ implies $s;\phi\mathcal{U} = t;\phi\mathcal{U}$. $s^\sharp;\phi = t^\sharp;\phi$ because of the universal property of the free model $k\mathcal{F}$. Then there is a model morphism ϕ' such that $coeq\langle s^\sharp, t^\sharp\rangle;\phi' = \phi$. This implies that $S;\phi = T;\phi$, which implies $\langle S, T\rangle \subseteq ker\phi$. □

The operations step stands for the closure of the original relation $\langle s, t\rangle$ under the "model operations". This can be achieved categorically by using the universal property of the free model over the indices of the relation. The equivalence step corresponds to the equivalence generated by the closure under operations. Because this is done at the level of model morphisms, the closure under operations is preserved.

Definition 3.25 Consider a binary relation $\langle s, t\rangle$ on the underlying domain of a model A. Then $\langle s, t\rangle$ is **closed under operations** iff $\langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle \subseteq \langle s, t\rangle$.

The **closure of** $\langle s, t\rangle$ **under operations** is the least relation closed under operations and containing $\langle s, t\rangle$, and is denoted $\mathbf{Op}\langle s, t\rangle$. □

Fact 3.26 Let $\langle s, t\rangle$ be any binary relation on the underlying domain of a model A. Then its closure under operations exists and is given by $\langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle$.

Proof: All we have to show is that $\langle (s^\sharp\mathcal{U})^\sharp, (t^\sharp\mathcal{U})^\sharp\rangle \subseteq \langle s^\sharp, t^\sharp\rangle$. This follows from the co-universal property of the co-unit ϵ of the adjunction between the category of domains and the category of models, or more precisely from $k\mathcal{F}\epsilon; v = (v\mathcal{U})^\sharp$ for any $k\mathcal{F} \xrightarrow{v} A$. □

Example 3.27 Let (S, Σ) be a many sorted signature and let $\langle s, t\rangle$ be an S-sorted binary relation on the carrier of the S-sorted Σ-algebra A. Then

- $\langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle$ is obtained by taking the union of the increasing chain of S-sorted relations $\langle s^n, t^n\rangle_{n\in\omega}$, where $\langle s^0, t^0\rangle = \langle s, t\rangle$ and $\langle s^{n+1}, t^{n+1}\rangle = \langle s^n, t^n\rangle \cup \{\langle \sigma_A(s^n), \sigma_A(t^n)\rangle \mid \sigma \in \Sigma\}$. $\langle \sigma_A(s^n), \sigma_A(t^n)\rangle$ is obtained by relating the results of all the applications of the operation σ_A to all pairs of elements related by $\langle s^n, t^n\rangle$. The union $\bigcup_{n\in\omega}\langle s^n, t^n\rangle$ is the same as relating all the results of the applications of all the derived operators to the pairs of elements related by $\langle s, t\rangle$.

- closing $\langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle$ under equivalence produces the congruence coequalising the S-sorted Σ-morphisms $s^\sharp$ and $t^\sharp$. The congruence is recovered categorically as the kernel of the coequaliser of $s^\sharp$ and $t^\sharp$.

□

The construction of the congruence closure of a binary relation can also be done in most cases by swapping the two steps corresponding to the closure under equivalence and closure under model operations, i.e., closing under equivalence first and under model operations afterwards. This requires coequalisers in the category of domains. Although our category-based framework is too abstract for proving the validity of this alternative construction of the congruence closure half of it still holds at this level:

Lemma 3.28 Further to the **DeductionFramework** assume the category $\mathbb{X}$ of domains has coequalisers. Let $\langle s, t\rangle$ be a relation on the underlying domain of the model A. Then

$$\mathbf{Op}\langle \bar{s}, \bar{t}\rangle \subseteq \mathbf{C}\langle s, t\rangle$$

where $\langle \bar{s}, \bar{t}\rangle$ is the equivalence closure of $\langle s, t\rangle$, i.e., $\langle \bar{s}, \bar{t}\rangle = ker(coeq\langle s, t\rangle)$.

Proof: By the universal property of kernels and Proposition 3.24, it is enough to show that $\bar{s}^\sharp; e = \bar{t}^\sharp; e$ where e is the coequaliser of $s^\sharp$ and $t^\sharp$.

This follows from the fact that $s; e\mathcal{U} = t; e\mathcal{U}$, which implies that $\bar{s}; e\mathcal{U} = \bar{t}; e\mathcal{U}$, and further implies that $\bar{s}^\sharp; e = \bar{t}^\sharp; e$ using the uniqueness part of the universal property corresponding to the adjunction determined by $\mathcal{U}$. □

Definition 3.29 We say that **congruences are concrete** iff any equivalence closed under operations is a congruence. □

Corollary 3.30 If congruences are concrete, then category-based equational logic is complete under the following inference rules:

[reflexivity] $\dfrac{}{(\forall A)\langle s, s\rangle}$

[symmetry] $\dfrac{(\forall A)\langle s, t\rangle}{(\forall A)\langle t, s\rangle}$

[transitivity] $\dfrac{(\forall A)\langle s, t\rangle \quad (\forall A)\langle t, u\rangle}{(\forall A)\langle s, u\rangle}$

[operations] $\dfrac{(\forall A)\langle s, t\rangle}{(\forall A)\langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle}$

substitutivity

□

3.5.2 Finitary model operations

In this subsection we show how the finitarity of $\mathcal{U}$ (Definition 3.5) reduces in practice to the finiteness of the model operations. The category-based formulation of 'finitary model operations' is that the forgetful functor $\mathcal{U}$ from models to domains preserves filtered colimits. We need the following technical condition on the category of domains:

[DomainRegularity]: the category of domains $\mathbb{X}$ is algebroidal and has colimits and filtered unions of equivalences.

Proposition 3.31 Under the **DeductionFramework** and **DomainRegularity** assumptions, $\mathcal{U}$ is finitary if the forgetful functor $\mathcal{U}$ from models to domains preserves filtered colimits.

Proof: Let $k \xrightarrow{\langle s,t \rangle} A\mathcal{U}$ be an arbitrary binary relation on the underlying domain of the model A. Because $\mathbb{X}$ is algebroidal, k is the colimit of a filtered diagram of finite domains $\{k_i\}_{i \in I}$. Let μ be the colimiting co-cone $\{k_i\}_{i \in I} \to k$ and let $s_i = \mu_i; s$ and $t_i = \mu_i; t$ for each $i \in |I|$.

$\mathcal{F}$ preserves colimits because it is a left adjoint, hence $\mu\mathcal{F}$ is still a colimiting co-cone. $\mu_i \mathcal{F}; v^\sharp = v_i^\sharp$ for $v \in \{s, t\}$ by the universal property of $k_i; \eta$, therefore

$$\langle s^\sharp, t^\sharp \rangle = colim_{i \in I} \langle s_i^\sharp, t_i^\sharp \rangle$$

in the comma category $(\Delta_{\mathbb{A}} \downarrow A)$. Then

$$\begin{array}{rcll}
\mathbf{C}\langle s, t \rangle & = & \mathcal{U}(ker(coeq\langle s^\sharp, t^\sharp \rangle)) & \text{(by Proposition 3.24)} \\
& = & ker(\mathcal{U}(coeq\langle s^\sharp, t^\sharp \rangle)) & (\mathcal{U} \text{ preserves kernels}) \\
& = & ker(\mathcal{U}(coeq(colim_{i \in I}\langle s_i^\sharp, t_i^\sharp \rangle))) & \\
& = & ker(\mathcal{U}(colim_{i \in I}(coeq\langle s_i^\sharp, t_i^\sharp \rangle))) & (coeq\colon (\Delta_{\mathbb{A}} \downarrow A) \to (A \downarrow \mathbb{A}) \text{ is left adjoint to} \\
& & & \qquad ker\colon (A \downarrow \mathbb{A}) \to (\Delta_{\mathbb{A}} \downarrow A)) \\
& = & ker(colim_{i \in I}(\mathcal{U}(coeq\langle s_i^\sharp, t_i^\sharp \rangle))) & (\mathcal{U} \text{ preserves filtered colimits}) \\
& = & colim_{i \in I} ker(\mathcal{U}(coeq\langle s_i^\sharp, t_i^\sharp \rangle)) & (\mathbb{X} \text{ has filtered unions of equivalences}) \\
& = & colim_{i \in I} \mathcal{U}(ker(coeq\langle s_i^\sharp, t_i^\sharp \rangle)) & (\mathcal{U} \text{ preserves kernels}) \\
& = & colim_{i \in I} \mathbf{C}\langle s_i, t_i \rangle & \text{(Proposition 3.24)}
\end{array}$$

This means that $\mathbf{C}\langle s, t \rangle = \bigcup_{i \in I} \mathbf{C}\langle s_i, t_i \rangle$. □

Whenever the domain category $\mathbb{X}$ is $\mathbb{S}et$-based, it has filtered unions of equivalences (as shown in Example 2.22). This includes all of the examples discussed in Section 2.3.

Corollary 3.32 All of the forgetful functors from categories of models to categories of domains presented in Section 2.3 are finitary.

Proof: All hypotheses of Proposition 3.31 related to the category of domains are trivially fulfilled by $\mathbb{S}et^S$. The forgetful functors from categories of models to categories of domains preserve filtered colimits because of the finitarity of the model operations.[35] When the model operations are finitary, the forgetful functor from model to domains *creates* filtered colimits, and creation is a stronger property than preservation. □

3.5.3 The Axiom of Choice *versus* projectivity

We use a form of the Axiom of Choice formulated in our category-based framework for proving that free models are always coequaliser projective:

[35] For the case of universal algebra, see Proposition 2, p 208 in Mac Lane's category theory textbook [64]. For all other cases the proof is very similar.

Proposition 3.33 If each coequaliser e in the category of models is a split epi at the domain level, i.e., if $e\mathcal{U}$ has a left inverse, then each free model is coequaliser projective.

Proof: Let $x \in |\mathbb{X}|$ be an arbitrary domain. We have to prove that $x\mathcal{F}$ is coequaliser projective. Let $A \xrightarrow{e} B$ be a model coequaliser and let $x\mathcal{F} \xrightarrow{h} B$ be any model morphism.

$$\begin{tikzcd} A \arrow[r, "e"] & B \\ & x\mathcal{F} \arrow[u, "h"'] \arrow[ul, "h'"] \end{tikzcd}$$

Let m be the left inverse to $e\mathcal{U}$ and let $x\mathcal{F} \xrightarrow{h'} A$ be the unique model morphism such that $x\eta; h'\mathcal{U} = x\eta; h\mathcal{U}; m$.

We now show that $h'; e = h$:

$$\begin{array}{rcll} x\eta; (h'; e)\mathcal{U} & = & x\eta; h'\mathcal{U}; e\mathcal{U} & \\ & = & x\eta; h\mathcal{U}; m; e\mathcal{U} & \text{(by the definition of } h') \\ & = & x\eta; h\mathcal{U} & \text{(by the definition of } m) \end{array}$$

$h'; e = h$ follows because the arrow $x\eta$ is universal from x to $\mathcal{U}$. $\square$

In practice, this form of the Axiom of Choice is always satisfied. In all of the examples previously discussed, model coequalisers are pointwise surjective because they are simply many sorted functions. The usual formulation of the Axiom of Choice asserts that for each element belonging to the image of a function, one can pick an element in the source that gets mapped into the previous one. In terms of functional composition, this is exactly the same as asserting the existence of a left inverse for any surjection, sometimes called a *choice function*. A special remark is needed for the order sorted case, where the fact that the forgetful functor forgets the inclusions between the subsort interpretations is essential.

3.5.4 Herbrand's Theorem revisited

For this paragraph we further assume that the category $\mathbb{A}$ of models has an initial object $0_{\mathbb{A}}$.

As pointed out by Goguen and Meseguer [39], there are definite advantages in the case when models do not have empty sorts. In this context, it is possible to have a more *computational* version of Herbrand's Theorem. The following result instantiated to the institution of order sorted Horn clause logic with equality gives *Herbrand's Theorem for non-empty sorts* as formulated by Goguen and Meseguer in [39].

Theorem 3.34 Herbrand's Theorem Under the **ConcreteDeductionFramework** and **DomainRegularity** assumptions, consider any collection Γ of conditional equations with finite hypotheses and with coequaliser projective quantifiers, and any $\mathcal{U}$-query $(\exists B)q$ where B is any coequaliser projective model. Suppose that $\mathcal{U}$ preserves filtered colimits and has non-empty sorts.

Then $\Gamma \models (\exists B)q$ iff $\Gamma \models (\forall y)q; h\mathcal{U}$ for some domain $y \in |\mathbb{X}|$ and some model morphism $h : B \to y\mathcal{F}$.

Proof: By Herbrand's Theorem 3.21, it is enough to prove that $0_\Gamma \models (\exists B)q$ iff $\Gamma \models (\forall y)q; h\mathcal{U}$ for some domain $y \in |\mathbb{X}|$ and some model morphism $h : B \to y\mathcal{F}$.

Assume that $0_\Gamma \models (\exists B)q$. Let $h\colon B \to 0_\Gamma$ be a solution for $(\exists B)q$ in 0_Γ. Consider $0_{\mathbb{X}}$ the initial domain. Since left adjoint functors preserve colimits, we may assume that $0_{\mathbb{X}}\mathcal{F} = 0_{\mathbb{A}}$, hence the unique model morphism $!_{0_\Gamma}\colon 0_{\mathbb{X}}\mathcal{F} \to 0_\Gamma$ is a coequaliser by virtue of the construction of 0_Γ (see Corollary 3.17). Since B is coequaliser projective, there exists a model morphism $h_0\colon B \to 0_{\mathbb{X}}\mathcal{F}$ such that $h_0; !_{0_\Gamma} = h$. Then $\Gamma \models (\forall 0_{\mathbb{X}})q; h_0\mathcal{U}$.

$$\begin{tikzcd}
B \arrow[r, "h"] \arrow[d, "f"'] \arrow[dr, "h_0"] & 0_\Gamma \\
y\mathcal{F} \arrow[r, "v^\sharp"'] & 0_{\mathbb{X}}\mathcal{F} = 0_{\mathbb{A}} \arrow[u, "!_{0_\Gamma}"']
\end{tikzcd}$$

For the converse, assume that $\Gamma \models (\forall y)q; f\mathcal{U}$ for some domain $y \in |\mathbb{X}|$ and some model morphism $f\colon B \to y\mathcal{F}$. Since $\mathcal{U}$ has non-empty sorts, there exists a domain map $v\colon y \to 0_{\mathbb{A}}\mathcal{U}$. Then $f; v^\sharp; !_{0_\Gamma}$ is a solution for $(\exists B)q$ in 0_Γ. $\square$

The model morphism h in this theorem is a solution form for q under Γ; logic programming deals with the computation of such morphisms.

4 Operational Semantics

By the operational semantics of a computing system one usually means a mathematical definition of *how* programs are executed by the system. For relational programming, most implementations use SLD-resolution as introduced by Prolog, and for equational logic programming most implementations use some refinement of narrowing.

Narrowing is a particular case of paramodulation. Paramodulation was first introduced as an operational inference rule in the context of attempts to integrate equality into resolution-based theorem provers [80]. Narrowing was introduced by Slagle [86]. Later, narrowing was used as a basis for *semantic unification* (i.e., unification modulo a set of rules) algorithms. *Basic narrowing* appeared for the first time in Hullot's work [57]. The completeness result for innermost narrowing in the context of canonical term rewriting systems is originally due to Fribourg [22]. Hölldobler's thesis [54] gives a systematic presentation of the current state of art of this field including also interesting historical references. Our presentation of the completeness of different refinements of paramodulation is influenced by [54].

Equational logic programming systems based on Horn clause logic with equality use a mixture of resolution (applied to relational symbols) and narrowing. However, it is important to notice that in the context of the embedding of Horn clause logics into equational logics described in Section 2.3.3, resolution appears as a refinement of narrowing in the presence of relational symbols.[36] This has mainly theoretical implications rather than practical ones because the use of resolution greatly improves the efficiency of the system, but it is important for an uniform algebraic treatment of the operational semantics of equational logic programming languages based on Horn clause logic with equality.

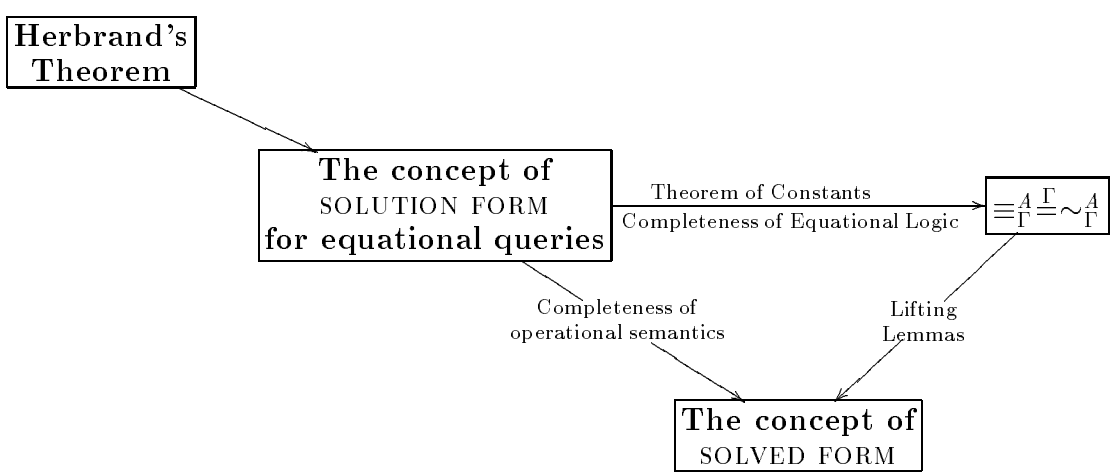
4.0.5 Completeness of Paramodulation: its Architecture

Our approach to the completeness of paramodulation departs fundamentally from previous treatments in that it is based on model theory rather than on combinatorial techniques involving term manipulations. We generalise the concept of paramodulation to **model theoretic paramodulation** by defining paramodulation as an inference rule with respect to an arbitrary fixed model. The ordinary concept of paramodulation is recovered as model theoretic paramodulation with respect to the initial algebra for an algebraic signature. The category-based dimension of our new approach brings out not only the simplicity of the categorical arguments (vis à vis set theoretical arguments), but more importantly, it shows that the core of the paramodulation-based operational semantics for equational logic programming can be developed independently of the details of the particular equational logic involved. In this way, the results of this work can be applied to a variety of equational logic programming systems that are rigorously based on some version of equational logic and whose operational semantics is based on some refinement of paramodulation (some form of narrowing, in general). This includes system based on many sorted or order sorted equational logic, Horn clause logic (with equality), equational logic modulo axioms, etc. These results might be relevant even for constraint programming since constraint logic (i.e., the logic underlying constraint logic

[36] This is explained in Section 4.2.1 below.

programming in the style of Eqlog, see Chapter 6) can be regarded as category-based equational logic. Another important consequence of the model theoretic approach to paramodulation is a direct treatment of computations *modulo axioms*. This is achieved by considering the paramodulation relation induced by the program on the initial model of the respective theory. For example, the programming language Eqlog is based on order sorted Horn clause logic with equality [38, 39] and supports refutations modulo axioms (associativity, commutativity, and their combination).

This chapter proposes a general scheme for the treatment of the completeness of paramodulation-based operational semantics. The core of this scheme is an analysis of the relationship between $\equiv_\Gamma^A$ (given a program Γ and a model A, the least congruence on A closed under Γ-substitutivity) and the relations induced on A by the operational inference rules, mainly paramodulation (this relation is denoted as $\sim_\Gamma^A$. This is technically connected to the concept of *solution* for equational queries through Theorem of Constants and Completeness of Equational Logic and to the concept of *solved form* through Lifting Lemmas. The terminology "solved form" was first introduced by Lassez [67] as a replacement to the traditional logic programming terminology of "computed answer substitution." The new terminology is more adequate to the modern methods of solving queries by system transformations rather than resolution-like techniques (see the survey [62]). Solution and solved forms are respectively the semantic and computational sides of the same concept. The soundness of the operational semantics means that any solved form is a solution and the completeness means that any solution form is an instantiation of a solved form.[37] In other words, the set of solutions of a query is the same as the set of solutions of the solved form. The connection to the model theory of equational logic programming is done via Herbrand's Theorem; this connects directly to the mathematical foundations of logic programming.



This figure visualises the architecture of the completeness of our approach to paramodulation-based operational semantics as discussed abovely. Because of efficiency concerns, equational logic programming systems actually implement various refinements of paramodulation rather than paramodulation itself. Most of these are refinements of narrowing, and one of the most powerful refinements is basic innermost normalised narrowing [54]. The completeness of different narrowing techniques is obtained in the

[37] However, the concept of completeness is usually taken to subsume soundness.

same way as the completeness of plain paramodulation, the only differences occurring at the level of the Lifting Lemmas. As shown in [54], the completeness of different narrowing techniques requires some restrictions on the programs.[38]

One of the most important properties of programs is *confluence*. We show that the completeness of paramodulation and the transitivity of the paramodulation relation are technically equivalent. By approaching confluence from a model theoretic angle, we show that the transitivity of the paramodulation relation is in fact equivalent to the confluence of the program with respect to a given reachable model. In this way, model theoretic paramodulation is complete for oriented application of rules if and only if the program is confluent.

4.1 Preliminaries

The framework for the categroy-based treatment of operational semantics is the general framework of category-based equational logic, i.e., a "forgetful" functor $\mathcal{U} : \mathbb{A} \to \mathbb{X}$ from a category of models to a category of doamins, and satisfying the following technical conditions:

> **[OperationalFramework]: DeductionFramework + DomainRegularity** + the forgetful functor $\mathcal{U}$ has a left adjoint $\mathcal{F}$ and preserves filtered colimits + congruences are concrete.

Definition 4.1 A conditional $\mathcal{U}$**-rule** is an oriented conditional finite $\mathcal{U}$-equation with finite hypotheses, usually written as $(\forall B) l \to r$ **if** $\langle s, t \rangle$ where $\langle s, t \rangle$ is called the hypotheses of the rule and $l \to r$ the **conclusion** (or the **head**) of the rule. The rule is **atomic** if its conclusion (head) is atomic as a binary relation. □

The quantifier B can in general be any model (see Definition 3.6). However, we restrict ourselves here to the case of [coequaliser] projective quantifiers, a condition strongly related to the completeness of the equational deduction (see Theorem 3.17). Recall from Proposition 3.33 that in the presence of a form of the Axiom of Choice, all free models are [coequaliser] projective. As a matter of notation, whenever a model is freely generated by a domain x (which in practice will be a collection of variable symbols), we will write $\forall x$ rather than $\forall x \mathcal{F}$; also for valuations we will use maps $x \to A\mathcal{U}$ rather than model morphisms $x\mathcal{F} \to A$.

4.1.1 Rewriting contexts

The concept of *context* plays a primary rôle in the mathematical formulation of rewriting as an inference rule. This paragraph is concerned with the category-based definition of context. Such a definition is crucial for the category-based treatment of rewriting because the notion of rewriting context ultimately has an algebraic nature; this makes the definition of rewriting independent of the tree-like representations of terms. In this way, rewriting can be defined on algebraic entities that are more abstract than the terms.

This is achieved by abstracting the properties of contexts known from the standard case of many-sorted algebra. One of the most important properties is the *unary* nature of contexts, i.e., rewriting contexts behave as unary functions. The following recalls the definition of context in many-sorted algebra:

[38] However, these restrictions are generally met in practice.

Definition 4.2 Let Σ be a many-sorted algebraic signature. Then a **rewriting Σ-context** is a Σ-term with one variable symbol having a single occurrence of that variable symbol.

Given any Σ-algebra A, a rewriting Σ-context c determines a map $c_A : A \to A$ that evaluates the context for any given value in A of the variable symbol of c. This is represented by the following diagram,

$$\begin{tikzcd} \{z\} \arrow[r, hook] \arrow[dr, "\underline{a}"'] & T_\Sigma(\{z\}) \arrow[d, "\underline{a}^\sharp"] \\ & A \end{tikzcd}$$

where for each $a \in A$, $\underline{a} : \{z\} \to A$ satisfies $\underline{a}(z) = a$. Then $c_A : A \to A$ is defined by $c_A(a) = \underline{a}^\sharp(c)$ where $\underline{a}^\sharp$ is the unique extension of $\underline{a}$ to a Σ-homomorphism. $\square$

Note that in general c_A is *not* an algebraic homomorphism. However, it is easy to notice that the rewriting contexts form a monoid under the composition (i.e., by plugging one context into another), and as shown in the following, the evaluation of contexts commutes with algebraic homomorphisms:

Proposition 4.3 Let c be any rewriting context in an algebraic signature Σ and $h : A \to B$ be a Σ-homomorphism. Then $c_A; h = h; c_B$.

Proof: Using the notation of Definition 4.2, for each $a \in A$ we have:

$$\begin{aligned} (c_A; h)(a) &= h(\underline{a}^\sharp(c)) \\ &= (\underline{a}; h)^\sharp(c) \qquad \text{(by the universal property of } T_\Sigma(\{z\})) \\ &= \underline{h(a)}^\sharp(c) \\ &= c_B(h(a)) \\ &= (h; c_B)(a). \end{aligned}$$

$\square$

This last property suggests the *natural transformation* nature of the rewriting contexts and motivates the following definition:

Definition 4.4 Let $\mathcal{U} : \mathbb{A} \to \mathbb{S}et^S$ be a forgetful functor from a category $\mathbb{A}$ of models. A **$\mathcal{U}$-context** is a natural transformation $c : \mathcal{U} \to \mathcal{U}$. The composition of $\mathcal{U}$-contexts is the usual composition of natural transformations. $\square$

From now on, we will in general use the more intuitive notation $c_A[t]$ instead of $t; c_A$ for the evaluation of a context c in a model A. This notation is closer to the usual notations for contexts in rewriting.

Definition 4.5 A binary relation $\langle s, t \rangle$ on the underlying domain of a model A is **closed under context evaluation** iff $\langle c_A[s], c_A[t] \rangle \subseteq \langle s, t \rangle$ for any context c. The least relation closed under context evaluations and containing $\langle s, t \rangle$ is called the **context closure** of $\langle s, t \rangle$. $\square$

Proposition 4.6 Let $\langle s, t\rangle = \{\langle s_i, t_i\rangle \mid i \in I\}$ be a binary relation on the underlying domain of a model A. If $\langle s, t\rangle$ is closed under operations, then it is also closed under context evaluations.

Proof: Let $\langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle$ be the closure under operations of $\langle s, t\rangle$ by Fact 3.26. By hypothesis, $\langle s, t\rangle = \langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle$. Then for any context c

$$\begin{array}{rcll}
\langle c_A[s], c_A[t]\rangle & = & \langle c_A[s^\sharp\mathcal{U}], c_A[t^\sharp\mathcal{U}]\rangle & \\
& = & \langle s^\sharp\mathcal{U}; c_A, t^\sharp\mathcal{U}; c_A\rangle & \\
& = & \langle c_{I\mathcal{F}}; s^\sharp\mathcal{U}, c_{I\mathcal{F}}; t^\sharp\mathcal{U}\rangle & \text{(by the naturality of } c) \\
& \subseteq & \langle s^\sharp\mathcal{U}, t^\sharp\mathcal{U}\rangle & \\
& = & \langle s, t\rangle. &
\end{array}$$

□

An essential property of rewriting contexts in MSA is that the converse of the previous result holds for transitive relations on reachable algebras:

Proposition 4.7 Let Σ be a many-sorted algebraic signature and A a reachable Σ-algebra. Then a transitive relation on A is closed under operations iff it is closed under rewriting context evaluations.

Proof: Let $\sim$ be a transitive relation on A. In the virtue of Proposition 4.6, it suffices to show that $\sim$ is closed under operations if it is closed under rewriting context evaluations.

Let σ be an arbitrary operation symbol in Σ and let $a = (a_1 \ldots a_n) \sim (b_1 \ldots b_n) = b$. We have to show that $\sigma_A(a) = \sigma_A(b)$. For simplicity (and without restricting generality) we can assume that $n = 2$. Because A is reachable, there exist t and t' ground terms such that $t_A = a_1$ and $t'_A = b_2$. Let $c[z] = \sigma(t, z)$ and $c'[z] = \sigma(z, t')$ be contexts, with variable symbol z. Then

$$\begin{array}{rcll}
\sigma_A(a) & = & c_A[a_2] & \\
& \sim & c_A[b_2] & \text{(since } a_2 \sim b_2) \\
& = & \sigma_A(a_1, b_2) & \\
& = & c'_A[a_1] & \\
& \sim & c'_A[b_1] & \text{(since } a_1 \sim b_1) \\
& = & \sigma_A(b). &
\end{array}$$

Now, $\sigma_A(a) \sim \sigma_A(b)$ because of the transitivity of $\sim$. □

This crucial property is central to the category-based definition of the notion of rewriting context:

Definition 4.8 Let $\mathcal{U} : \mathbb{A} \to \mathbb{X}$ be a forgetful functor from a category of models to a category of domains. A monoid $\mathcal{C}$ of **rewriting contexts** for $\mathcal{U}$ is a submonoid of all $\mathcal{U}$-contexts such that any transitive relation on a reachable model that is closed under rewriting context evaluations is also closed under operations. □

In principle it is possible to have various monoids of rewriting contexts for a fixed category of models and category of domains. Some of these could be very different from from the standard ones, thus generating non-conventional notions of rewriting and paramodulation.

Corollary 4.9 Let $\mathcal{C}$ be a fixed monoid of rewriting contexts. An equivalence on A is a congruence iff it is closed under rewriting context evaluations. □

4.2 Inference Rules

This section presents the inference rules for the operational semantics of equational logic programming as a refinement of paramodulation. Recall from [54] the notion of *occurrence* in a term. For any term t and any occurrence π in t, let $t|_\pi$ denote the subterm of t whose root is positioned at π, and let $t|_{\pi \leftarrow s}$ denote the term obtained from t replacing $t|_\pi$ with s as a subterm in t. An *equational goal* is a pair $\langle t_1, t_2 \rangle$ of terms. The notion of occurrence can be extended from terms to goals by regarding any goal $\langle t_1, t_2 \rangle$ as a term having two subterms t and t'. The instantiation of a term by a substitution θ is denoted $t\theta$, and the composition of substitutions is written simply by concatenation and in diagrammatic order. The empty substitution is denoted ϵ.

The presentation of the rules of inference for the operational semantics of equational logic programming follows the more classical approach of computed anwer substitutions rather than the more modern approach of transformation of system of equations (see [20]).[39] The main reason for this choice is the example nature of this section and also that this presentation of the the inference rules for the operational semantics is faithful to the current implementation of the Eqlog system.

Definition 4.10 Let Σ be an algebraic signature and Γ be a program in Σ, i.e., a collection of Σ-rules. Then the **paramodulation** rule is

$$\frac{\mathbf{G} \cup \{\langle t_1, t_2 \rangle\}}{\mathbf{G}\theta \cup \langle s\theta, t\theta \rangle \cup \{(\langle t_1, t_2 \rangle|_{\pi \leftarrow r})\theta\}}$$

where $(\forall X) l \rightarrow r$ **if** $\langle s, t \rangle$ is a new variant[40] of a rule in Γ, $\mathbf{G}$ is a list of goals, and θ is the most general unifier of l and $\langle t_1, t_2 \rangle|_\pi$. A single inference step of this rule is denoted $\longrightarrow\!\!\!\!\rightarrow_p$.

A **rewriting** step (denoted $\longrightarrow\!\!\!\!\rightarrow_R$) is a paramodulation step such that the domain of the substitution θ doesn't contain any variable from $\langle t_1, t_2 \rangle$.

A **narrowing** step (denoted $\longrightarrow\!\!\!\!\rightarrow_n$) is a paramodulation step such that $\langle t_1, t_2 \rangle|_\pi$ is *not* a variable. □

The elimination of trivial goals is done directly through *syntactic* unification:

Definition 4.11 The **reflection** rule is:

$$\frac{\mathbf{G} \cup \{\langle t_1, t_2 \rangle\}}{\mathbf{G}\theta}$$

where $\mathbf{G}$ is a list of goals and θ is the most general unifier of t_1 and t_2. One step of this rule is denoted $\longrightarrow\!\!\!\!\rightarrow_r$. □

By preventing the application of narrowing steps at occurrences introduced by the computed substitutions, the search space of the narrowing chains is reduced drastically. This restriction is called *basic* narrowing and still preserves the completeness of the operational semantics when the program is a canonical rewriting system [54]. In order to be able to write down the rule of basic narrowing as an inference rule without side conditions, [54] introduces a new representation for goals consisting of a **skeleton** part

[39] Originating from Martelli and Montanari's work on syntactic unification [71].

[40] Obtained by renaming all variables in the rule with new names.

(just goals in the ordinary sense, i.e., pairs of terms) and an **environment** part (the accumulation of the computed substitutions). By also using the rule of **innermost reflection**, it is enough to restrict the application of the narrowing steps to only those occurrences that are leftmost innermost.

Definition 4.12 A **redex** in a goal is an occurrence at which a narrowing step could be applied. An **innermost** redex is a redex such that there doesn't exist any other redex below it.

The rule of **basic innermost narrowing** is:

$$\frac{\langle \mathbf{G} \cup \{\langle t_1, t_2\rangle\}, \sigma\rangle}{\langle \mathbf{G} \cup \langle s, t\rangle \cup \{\langle t_1, t_2\rangle|_{\pi\leftarrow r}\}, \sigma\theta\rangle}$$

where π is a innermost redex in $\langle t_1, t_2\rangle$ for $\langle t_1\sigma, t_2\sigma\rangle$, θ is the most general unifier of $(\langle t_1, t_2\rangle|_\pi)\sigma$ and l and $(\forall X)l{\rightarrow}r$ **if** $\langle s, t\rangle$ is a new variant of a clause in Γ. One step of this rule is denoted $\longrightarrow\!\!\!\!\rightarrow_{in}$.

The rule of **innermost reflection** is:

$$\frac{\langle \mathbf{G} \cup \{\langle t_1, t_2\rangle\}, \sigma\rangle}{\langle \mathbf{G} \cup \{\langle t_1, t_2\rangle|_{\pi\leftarrow x}\}, \sigma\theta\rangle}$$

where π is a innermost redex in $\langle t_1, t_2\rangle$ for $\langle t_1\sigma, t_2\sigma\rangle$ and θ is the substitution replacing a new variable x by $(\langle t_1, t_2\rangle|_\pi)\sigma$. One step being denoted as $\longrightarrow\!\!\!\!\rightarrow_{ir}$. $\Box$

Let $\Box$ denote the empty list of goals. Recall that a chain of inference steps is called a **refutation** iff it ends in $\Box$.

Definition 4.13 A substitution θ is an **instantiation** of another substitution φ (written $\theta \leq \varphi$) iff there exists a substitution γ such that $\theta = \varphi\gamma$. $\Box$

Fact 4.14 The relation $\leq$ on substitutions is a preorder. $\Box$

Definition 4.15 Consider a system of inference rules for equational logic programming operational semantics. A **computed answer substitution**[41] is the accumulation of the substitutions computed by a refutation chain. $\Box$

The inference system is **sound** iff for any list of goals $\mathbf{G}$, any solved form is a solution form for $\mathbf{G}$, and it is **complete** iff any solution form for $\mathbf{G}$ is an instantiation of some solved form.

4.2.1 Resolution as a refinement of paramodulation

In this paragraph we show how resolution can be regarded as paramodulation in the context of the embedding of Horn clause logics into equational logics developed in Section 2.3.3.

Definition 4.16 Let (S, Σ, Π) be a first order signature and Γ a collection of (S, Σ, Π)-clauses. The **resolution** rule is

[41] Called "solved form" in the scheme proposed in the introduction to this chapter.

$$\frac{\mathbf{G} \cup \{p(t)\}}{\mathbf{G}\theta \cup C\theta}$$

where $(\forall X)p(s)$ **if** C is a new variant of a clause in Γ, p is a relational symbol in Π, and θ is the most general unifier of $p(s)$ and $p(t)$. □

Fact 4.17 By using the transformations described in Section 2.3.3, a resolution step can be performed by a narrowing step followed by a reflection step.

Proof: Using the notations of the previous definition, the clause $(\forall X)p(s)$ **if** C becomes a $\Sigma^b \cup \Pi^b$-rule $(\forall X)p^b(s) \rightarrow \mathbf{t}$ **if** C^b where C^b is the transformation of the (Σ, Π)-condition C into the corresponding set of $\Sigma^b \cup \Pi^b$-equations. This rule can be used for performing a narrowing step at the topmost symbol of the selected goal from

$$\mathbf{G}^b \cup \{\langle p^b(t), \mathbf{t}\rangle\}$$

and getting $\mathbf{G}^b\theta \cup C^b\theta$ as a result after eliminating $\langle \mathbf{t}, \mathbf{t}\rangle$ by a reflection step. □

4.3 Model Theoretic Paramodulation

In this section we extend the concept of paramodulation to model theoretic paramodulation within the framework of category-based equational logic, and study the relationship between the paramodulation relation induced by a program Γ on a model A and the least congruence on A closed under Γ-substitutivity. Accordingly to the general scheme proposed in the introduction, this goes at the heart of the category-based treatment of the operational semantics for equational logic programming. The completeness of paramodulation is explained by the identity between these two relations. We show that this identity problem reduces exactly to the transitivity of the paramodulation relation.

For simplicity of notation, we will often omit[42] writing the forgetful functor $\mathcal{U}$ in case of domain maps underlying model morphisms, i.e., we write $s;h$ rather than $s;h\mathcal{U}$.

4.3.1 The paramodulation relation

This paragraph introduces the concept of model theoretic paramodulation in the form of a binary relation induced by a given program on an arbitrary model. We assume a fixed monoid $\mathcal{C}$ of rewriting $\mathcal{U}$-contexts.

Definition 4.18 Let Γ be a collection of conditional $\mathcal{U}$-rules and consider an arbitrary model A. Then a binary relation $\sim$ on A is **closed under Γ-paramodulation** iff for any rule $(\forall B)l \rightarrow r$ **if** $\langle s, t\rangle$ in Γ, for any model morphism $h: B \rightarrow A$, and for any rewriting $\mathcal{U}$-context c,

$$c_A[l;h] \sim b \quad \text{if} \quad s;h \sim t;h \quad \text{and} \quad c_A[r;h] \sim b$$

for any b in the underlying domain of A.

The least binary relation on A closed under reflexivity, symmetry and Γ-paramodulation is denoted as $\sim_\Gamma^A$. □

The concept of the least binary reflexive-symmetric relation closed under paramodulation is an algebraic abstraction of the relation on terms induced by paramodulation as a refutation rule:

[42] Only in this section and the following one.

Fact 4.19 Let T_Σ be the initial Σ-algebra for an algebraic signature Σ, i.e., the algebra of ground terms. For any collection Γ of conditional Σ-rules,

$$\sim_\Gamma^{T_\Sigma} = \{\langle t_1, t_2\rangle \mid \langle t_1, t_2\rangle \xrightarrow{\Gamma}\!\!\!\!\rightarrow_R^* \Box\}$$

i.e., the least relation on T_Σ closed under reflexivity, symmetry and Γ-paramodulation consists exactly of those pairs of terms for which there exists a paramodulation and reflexivity refutation using Γ. $\Box$

Given a program Γ we can define the concept of **(model theoretic) paramodulation** with respect to a model A as an inference rule on A-goals, i.e., symmetrical pairs of elements from A:

$$[\mathsf{mtp}] \quad \frac{\langle s; h, t; h\rangle \quad \langle c_A[r; h], b\rangle}{\langle c_A[l; h], b\rangle}$$

for any rule $(\forall B) l \to r$ **if** $\langle s, t\rangle$ in Γ, for any model morphism $h\colon B \to A$, and for any rewriting $\mathcal{U}$-context c. The symmetry axiom is explained by the fact that the goals in equational logic programming are *not* oriented, i.e., the position of the sides in a goal doesn't matter.

Proposition 4.20 For any model A, the least relation on A closed under reflexivity, symmetry and Γ-paramodulation exists and is given by

$$\sim_\Gamma^A = \bigcup_{n\in\omega} \sim_{\Gamma,n}^A$$

where $\sim_{\Gamma,0}^A = D_{A\mathcal{U}}$ (the diagonal) and

$$\sim_{\Gamma,n+1}^A = \sim_{\Gamma,n}^A \cup sym(\bigcup\{\langle c_A[l; h], b\rangle \mid \langle c_A[r; h], b\rangle, \langle s; h, t; h\rangle \subseteq \sim_{\Gamma,n}^A\})$$

for each $n \in \omega$.

Proof: The reflexivity of $\bigcup_{n\in\omega} \sim_{\Gamma,n}^A$ is given by $\sim_{\Gamma,0}^A$. In order to prove its symmetry, we show by induction on $n \in \omega$ that $\sim_{\Gamma,n}^A$ is symmetric. We use Lemma 2.34. Consider $\langle s', t'\rangle \subseteq \bigcup_{n\in\omega} \sim_{\Gamma,n}^A$ finite. Since $\{\sim_{\Gamma,n}^A \mid n \in \omega\}$ is filtered, there exists $n \in \omega$ such that $\langle s', t'\rangle \subseteq \sim_{\Gamma,n}^A$. The rest follows by the induction hypothesis and by the remark that the union of two symmetric relations is symmetric too.

In order to prove the closure under Γ-paramodulation of $\bigcup_{n\in\omega} \sim_{\Gamma,n}^A$, consider

$$(\forall B) l \to r \text{ **if** } \langle s, t\rangle \in \Gamma, h\colon B \to A \text{ a model morphism and } c \text{ a rewriting context}$$

such that $\langle s; h, t; h\rangle, \langle c_A[r; h], b\rangle \subseteq \bigcup_{n\in\omega} \sim_{\Gamma,n}^A$. Because of the finiteness of both $\langle s; h, t; h\rangle$ and $\langle c_A[r; h], b\rangle$, there exists $m \in \omega$ such that $s; h \sim_{\Gamma,m}^A t; h$ and $c_A[r; h] \sim_{\Gamma,m}^A b$. Therefore, $c_A[l; h] \sim_{\Gamma,m+1}^A b$.

Now, consider any other reflexive-symmetric binary relation Q on A that is closed under Γ-paramodulation. By induction on $n \in \omega$, $\sim_{\Gamma,n}^A \subseteq Q$. Then $\bigcup_{n\in\omega} \sim_{\Gamma,n}^A \subseteq Q$. $\Box$

The intuitive meaning of $\sim_{\Gamma,n}^A$ is the reflexive-symmetric relation generated by applying *at most* n Γ-paramodulation steps.

The soundness of model theoretic paramodulation is given by the following result. Any pair of elements that can be refuted through paramodulation, can be proved using standard equational deduction too.

Proposition 4.21 Let Γ be a collection of conditional $\mathcal{U}$-rules. Then for any model A, $\sim^A_\Gamma \subseteq \equiv^A_\Gamma$.

Proof: Since $\equiv^A_\Gamma$ is closed under reflexivity and symmetry because it is a congruence, all we have to show is that it is also closed under Γ-paramodulation. Let $(\forall B) l \to r$ **if** $\langle s, t\rangle$ be any rule in Γ and $h\colon B \to A$ be a model morphism such that $s;h \equiv t;h$ and such that $c_A[r;h] \equiv b$ for some rewriting context c and some b.

Because $\equiv^A_\Gamma$ is closed under Γ-substitutivity, we have that $l;h \equiv r;h$. Because $\equiv^A_\Gamma$ is closed under operations and by Proposition 4.6, $c_A[l;h] \equiv c_A[r;h]$. Then $c_A[l;h] \equiv b$ by the transitivity of $\equiv^A_\Gamma$. $\square$

The completeness of model theoretic paramodulation is given by the opposite inclusion and works only for the case of reachable models:

Proposition 4.22 Let Γ be a collection of conditional $\mathcal{U}$-rules and let A be a reachable model. Then $\sim^A_\Gamma$ is an equivalence iff $\sim^A_\Gamma = \equiv^A_\Gamma$.

Proof: Since $\equiv^A_\Gamma$ is an equivalence and because of Proposition 4.21, we have to show only that if $\sim^A_\Gamma$ is an equivalence then $\equiv^A_\Gamma \subseteq \sim^A_\Gamma$.

The closure of $\sim^A_\Gamma$ under Γ-substitutivity is obtained directly from the closure under Γ-paramodulation for the particular case when the context c is the identity, and from the reflexivity of $\sim^A_\Gamma$.

Because A is reachable and $\sim^A_\Gamma$ is an equivalence, the closure of $\sim^A_\Gamma$ under operations is the same as its closure under rewriting context evaluations. The closure under rewriting context evaluations is shown by proving by induction on $n \in \omega$ that $\sim^A_{\Gamma,n}; u_A \subseteq \sim^A_\Gamma$ for any rewriting context u. So consider

$$(\forall B) l \to r \textbf{ if } \langle s, t\rangle \in \Gamma, h\colon B \to A \text{ a model morphism and } c \text{ a rewriting context}$$

such that $s;h \sim^A_{\Gamma,n} t;h$ and $c_A[r;h] \sim^A_{\Gamma,n} b$. By applying the induction hypothesis for n, we get that $u_A[c_A[r;h]] \sim^A_\Gamma u_A[b]$ which means that $(c;u)_A[r;h] \sim^A_\Gamma u_A[b]$. Now since $\sim^A_\Gamma$ is closed under Γ-paramodulation, we obtain that $(c;u)_A[l;h] \sim^A_\Gamma u_A[b]$, meaning that $u_A[c_A[l;h]] \sim^A_\Gamma u_A[b]$. Because

$$\sim^A_{\Gamma,n+1} = \sim^A_{\Gamma,n} \cup sym(\bigcup\{\langle c_A[l;h], b\rangle \mid \langle c_A[r;h], b\rangle, \langle s;h, t;h\rangle \subseteq \sim^A_{\Gamma,n}\})$$

we can conclude that $\sim^A_{\Gamma,n}; u_A \subseteq \sim^A_\Gamma$ by using Fact 2.10.

Because congruences are concrete, $\sim^A_\Gamma$ is a congruence (which is closed under Γ-substitutivity as shown above). Since $\equiv^A_\Gamma$ is the least congruence closed under Γ-substitutivity, we have $\equiv^A_\Gamma \subseteq \sim^A_\Gamma$. $\square$

So, the completeness of model theoretic paramodulation reduces to the transitivity of the paramodulation relation:

Completeness of model theoretic paramodulation = transitivity of the paramodulation relation.

4.3.2 Completeness of model theoretic paramodulation

Proposition 4.22 links the completeness of paramodulation to the equivalence property of the paramodulation relation $\sim^A_\Gamma$. In fact, $\sim^A_\Gamma$ is always reflexive and symmetric. In this way, the transitivity of the paramodulation relation is technically equivalent to the completeness of paramodulation.

In this paragraph, $\sim^A_\Gamma$ is shown to be transitive when backward applications of the rules in Γ are allowed. This solution is more on the side of theorem proving rather than logic programming, but the next section deals with this problem in a different way by relating it to confluence.

Definition 4.23 Let Γ be a collection of conditional $\mathcal{U}$-rules. Let $\overline{\Gamma}$ denote the collection of conditional $\mathcal{U}$-rules obtained by reversing the orientation of the rules in Γ, i.e.,

$$\overline{\Gamma} = \{(\forall B) r \to l \textbf{ if } \langle s, t\rangle \mid (\forall B) l \to r \textbf{ if } \langle s, t\rangle \in \Gamma\}$$

□

Fact 4.24 For any model A and any collection Γ of conditional $\mathcal{U}$-rules, $\equiv^A_\Gamma = \equiv^A_{\Gamma\cup\overline{\Gamma}}$. □

For the rest of the section we suppose that all coproducts in the category $\mathbb{X}$ of domains are disjoint.

Proposition 4.25 Let Γ be a collection of conditional atomic $\mathcal{U}$-rules. Then for any model A, $\sim^A_{\Gamma\cup\overline{\Gamma}}$ is transitive.

Proof: Because of Lemma 2.34 it is enough to prove that if $\langle a, b\rangle \subseteq \sim^A_{\Gamma\cup\overline{\Gamma}}$ and $\langle b, d\rangle \subseteq \sim^A_{\Gamma\cup\overline{\Gamma}}$ then $\langle a, d\rangle \subseteq \sim^A_{\Gamma\cup\overline{\Gamma}}$ for a, b, d finite. Since $\langle b, d\rangle$ is finite, there exists $n \in \omega$ such that $\langle b, d\rangle \subseteq \sim^A_{\Gamma\cup\overline{\Gamma},n}$. Therefore, we show by induction on $n \in \omega$ that $a \sim^A_{\Gamma\cup\overline{\Gamma}} b$ and $\langle b, d\rangle \subseteq \sim^A_{\Gamma\cup\overline{\Gamma},n}$ implies $a \sim^A_{\Gamma\cup\overline{\Gamma}} d$, where a, b, d are finite. For the induction step, assume that $a \sim^A_{\Gamma\cup\overline{\Gamma}} b$ and $\langle b, d\rangle \subseteq \sim^A_{\Gamma\cup\overline{\Gamma},n+1}$. In the virtue of Lemma 2.14 and because the rules in Γ are atomic, we may further assume that

$$b = c_A[l; h] \text{ for some } (\forall B) l \to r \textbf{ if } \langle s, t\rangle \in \Gamma \cup \overline{\Gamma}, B \xrightarrow{h} A \text{ and } c \text{ rewriting context}$$

such that $s; h \sim^A_{\Gamma\cup\overline{\Gamma},n} t; h$ and $c_A[r; h] \sim^A_{\Gamma\cup\overline{\Gamma},n} d$. Now, by applying a $\Gamma\cup\overline{\Gamma}$-paramodulation closure step for $(\forall B) r \to l$ **if** $\langle s, t\rangle$ (still in $\Gamma \cup \overline{\Gamma}$) and h, we obtain that $c_A[r; h] \sim^A_{\Gamma\cup\overline{\Gamma}} a$ since $\sim^A_{\Gamma\cup\overline{\Gamma}}$ is closed under $\Gamma \cup \overline{\Gamma}$-paramodulation. Because $a \sim^A_{\Gamma\cup\overline{\Gamma}} c_A[r; h]$ and $c_A[r; h] \sim^A_{\Gamma\cup\overline{\Gamma},n} d$, we can apply the induction hypothesis and conclude by $a \subseteq \sim^A_{\Gamma\cup\overline{\Gamma}} d$. □

The completeness of model theoretic paramodulation when backward applications of rules are alowed is given by the following corollary:

Corollary 4.26 Let Γ be a collection of conditional atomic $\mathcal{U}$-rules. Further assume that any reflexive-symmetric-transitive relation in the category of domains is an equivalence. Then for any reachable model A, we have $\sim^A_{\Gamma\cup\overline{\Gamma}} = \equiv^A_\Gamma$. □

Completeness of many sorted paramodulation. We conclude this section with an example. We illustrate how the general scheme discussed in the introduction to the chapter can be used in conjuction with the previous results on model theoretic paramodulation to prove the completeness of paramodulation as a refutation procedure in the case of many sorted algebra.

We fix an algebraic signature Σ.

Corollary 4.27 Let Γ be a collection of conditional Σ-rules. If $\Gamma \models_\Sigma (\forall X)\langle t_1, t_2\rangle$, then there exists a rewriting refutation of $\langle t_1, t_2\rangle$ using $\Gamma \cup \overline{\Gamma}$, i.e., $\langle t_1, t_2\rangle \xrightarrow{\Gamma\cup\overline{\Gamma}}{}^*_R \Box$.

Proof: By the Theorem of Constants (5.52),

$$\Gamma \models_\Sigma (\forall X)\langle t_1, t_2\rangle \text{ iff } \Gamma \models_{\Sigma_X} (\forall\emptyset)\langle t_1, t_2\rangle$$

where Σ_X is the signature obtained by adjoining X to Σ as new constants. By the Completeness Theorem, $\langle t_1, t_2\rangle$ belongs to $\equiv_\Gamma^{T_{(\Sigma_X)}}$, i.e., the least congruence on $T_{(\Sigma_X)}$ closed under Γ-substitutivity. By Theorem 4.26, $\langle t_1, t_2\rangle$ belongs to $\sim_{\Gamma\cup\overline{\Gamma}}^{T_{(\Sigma_X)}}$, i.e., the least reflexive relation on $T_{(\Sigma_X)}$ closed under $\Gamma \cup \overline{\Gamma}$-paramodulation. The rest follows by Fact 4.19. $\Box$

Definition 4.28 For any algebraic signature Σ, let $F(\Sigma)$ be the collection of all **functional reflexive axioms**, i.e., $F(\Sigma) = \{(\forall x_1 \ldots x_n) f(x_1 \ldots x_n) = f(x_1 \ldots x_n) \mid f \in \Sigma\}$. $\Box$

A similar version of the following Lifting Lemma appears in [54]:

Proposition 4.29 <u>Lifting Lemma</u> Let $\mathbf{G}$ be a finite set of goals. If $\mathbf{G}\theta \xrightarrow{\Gamma}{}^*_{p,r} \Box$ with computed answer substitution σ, then $\mathbf{G} \xrightarrow{\Gamma\cup F(\Gamma)}{}^*_{p,r} \Box$ with computed answer substitution γ such that $\theta\sigma \leq \gamma$.

Proof: We prove by induction on $n \in \omega$ that if $\mathbf{G}\theta \xrightarrow{\Gamma}{}^n_{p,r} \Box$, then $\mathbf{G} \xrightarrow{\Gamma\cup F(\Gamma)}{}^*_{p,r} \Box$ with γ computed answer substitution such that $\theta\sigma \leq \gamma$. For the induction step, there are two cases: when the first step is a reflection, and when it is a paramodulation.

Suppose $\mathbf{G}\theta \longrightarrow\!\!\!\!\rightarrow_r \mathbf{G}'\theta\varphi \xrightarrow{\Gamma}{}^{n-1}_{p,r} \Box$ where $\mathbf{G} = \mathbf{G}' \cup \{\langle t_1, t_2\rangle\}$, $\varphi = mgu(t_1\theta, t_2\theta)$ and σ' is the answer substitution computed by the last $n-1$ refutation steps. Then $\varphi\sigma' = \sigma$.

There exists $\varphi' = mgu(t_1, t_2)$ and a unique substitution ζ such that $\theta\varphi = \varphi'\zeta$. We can do a reflection step $\mathbf{G} \longrightarrow\!\!\!\!\rightarrow_r \mathbf{G}'\varphi'$. Since $(\mathbf{G}'\varphi')\zeta = \mathbf{G}'\theta\varphi$, by the induction hypothesis, there is $\mathbf{G}'\varphi' \xrightarrow{\Gamma\cup F(\Gamma)}{}^*_{p,r} \Box$ with γ' the computed answer substitution such that $\zeta\sigma' \leq \gamma'$. Therefore, there exists a refutation $\mathbf{G} \xrightarrow{\Gamma\cup F(\Gamma)}{}^*_{p,r} \Box$ with $\gamma = \varphi'\gamma'$ computed answer substitution and such that

$$\begin{aligned}
\theta\sigma &= \theta\varphi\sigma' \\
&= \varphi'\zeta\sigma' \\
&\leq \varphi'\gamma' \\
&= \gamma
\end{aligned}$$

Now, suppose that $\mathbf{G}\theta \xrightarrow{\Gamma}\!\!\!\!\rightarrow_p (\mathbf{G}\theta|_{\pi\leftarrow r})\varphi \xrightarrow{\Gamma}\!\!\!\!\rightarrow^{n-1}_{p,r} \Box$ where $(\forall X)l\rightarrow r$ **if** $\langle s,t\rangle$ is a new variant of a clause in Γ and $\varphi = mgu(\mathbf{G}\theta|_\pi, l)$ for some occurrence π in $\mathbf{G}\theta$.

First, assume that π is a basic occurence (i.e., not introduced by θ). In this case, $\mathbf{G}\theta|_\pi = (\mathbf{G}|_\pi)\theta$. Since the variables of the selected clause don't clash with the logical variables, φ is the most general unifier of $(\mathbf{G}|_\pi)\theta$ and $l\theta$. Let φ' be the most general unifier of $\mathbf{G}|_\pi$ and l. Then there exists a unique ζ such that $\theta\varphi = \varphi'\zeta$. We have that $\mathbf{G} \longrightarrow\!\!\!\!\rightarrow_p (\mathbf{G}|_{\pi\leftarrow r})$ and that $((\mathbf{G}|_{\pi\leftarrow r})\varphi')\zeta \xrightarrow{\Gamma}\!\!\!\!\rightarrow^{n-1}_{p,r} \Box$. By applying the same argument as in the previous case when the first refutation step was a reflection, we deduce the existence of a computed answer substitution γ such that $\theta\sigma \leq \gamma$.

The last case occurs when π is not a basic occurence. Then $\pi = \pi_1\pi_2$ where $\mathbf{G}|_{\pi_1}$ is a variable. Let μ be the substitution θ restricted only to the variable $\mathbf{G}|_{\pi_1}$. Then $\theta = \mu + \theta'$ where $dom\mu \cap dom\theta' = \emptyset$. Then $\mathbf{G} \xrightarrow{F(\Gamma)}\!\!\!\!\rightarrow^*_p \mathbf{G}\mu \longrightarrow\!\!\!\!\rightarrow_p (\mathbf{G}\mu|_{\pi\leftarrow r})\varphi$. Because of the renaming of the variables we may also assume that $dom\varphi \cap dom\theta' = \emptyset$. Then we have that $\mu\varphi\theta' = (\mu + \theta')\varphi = \theta\varphi$, which implies that $(\mathbf{G}\mu|_{\pi\leftarrow r})\varphi\theta' \xrightarrow{\Gamma}\!\!\!\!\rightarrow^{n-1}_{p,r} \Box$ because $(\mathbf{G}\mu|_{\pi\leftarrow r})\varphi\theta' = (\mathbf{G}\theta|_{\pi\leftarrow r})\varphi$. By the induction hypothesis there exists a refutation $(\mathbf{G}\mu|_{\pi\leftarrow r})\varphi \xrightarrow{F(\Gamma)\cup\Gamma}\!\!\!\!\rightarrow^*_{p,r} \Box$ with γ' computed answer substitution such that $\theta'\sigma' \leq \gamma'$ where σ' is the substitution computed by the refutation $(\mathbf{G}\theta|_{\pi\leftarrow r})\varphi \xrightarrow{\Gamma}\!\!\!\!\rightarrow^{n-1}_{p,r} \Box$. Then $\gamma = \mu\varphi\gamma'$ and by a similar argument as in the previous cases we can prove that $\theta\sigma \leq \gamma$. $\Box$

Corollary 4.30 Let Γ be a collection of conditional Σ-rules. Then the refutation procedure through reflexivity and paramodulation via $\Gamma \cup \overline{\Gamma} \cup F(\Sigma)$ is complete. $\Box$

4.4 Paramodulation modulo a Model Morphism

This section proposes an abstract treatment for computations modulo axioms. Each theory determines a quotienting morphism for each model A (see Theorem 3.17) constructing the free model over A satisfying that theory. This quotienting can be considered as the model theoretic expression of the (logical) theory. In this way, the study of computations modulo a model morphism generalises the study of computations modulo axioms. We study the relationship between provability by paramodulation in a model and provability by paramodulation in the quotient model. A standard example is given by the quotient of an initial model (i.e., model of ground terms) modulo axioms.[43]

The following result shows that any model morphism preserves provability under paramodulation:

Proposition 4.31 Let Γ be a collection of conditional $\mathcal{U}$-rules and $f: A \rightarrow A'$ be an arbitrary model morphism. Then

$$\sim^A_\Gamma ; f \subseteq \sim^{A'}_\Gamma$$

Proof: It is enough to show by induction on $n \in \omega$ that

$$\sim^A_{\Gamma,n} ; f \subseteq \sim^{A'}_{\Gamma,n}$$

For the induction step consider

[43] Section 4.5.3 elaborates on this example.

$(\forall B) l \to r$ **if** $\langle s,t\rangle \in \Gamma, h\colon B \to A$ a model morphism and c a rewriting context

such that $\langle s;h, t;h\rangle, \langle c_A[r;h], b\rangle \subseteq \sim^A_{\Gamma,n}$. By induction hypothesis, $\langle s;h;f, t;h;f\rangle, \langle c_A[r;h;f], b;f\rangle \subseteq \sim^{A'}_{\Gamma,n}$. Hence $\langle c_A[l;h], b\rangle;f = \langle c_A[l;h;f], b;f\rangle \subseteq \sim^{A'}_{\Gamma,n+1}$, which proves that $\sim^A_{\Gamma,n+1};f \subseteq \sim^{A'}_{\Gamma,n+1}$. □

The equality

$$\sim^A_\Gamma;f = \sim^{A'}_\Gamma$$

doesn't hold in general because in its present form it dismisses the rôle played in proofs by the quotienting. A way to integrate the quotienting into the proof theory is given by introducing a new inference rule:

Definition 4.32 Let $\sim^{A,f}_\Gamma$ be the least reflexive-symmetric relation closed under Γ-paramodulation and under

$$[\mathsf{modf}] \quad \frac{Q}{ker(f) \circ Q \circ ker(f)}$$

where Q is any binary relation on the underlying domain of A. □

Fact 4.33 The relation $\sim^{A,f}_\Gamma$ exists and can be obtained in the manner of Proposition 4.20 by an alternation of Γ-paramodulation steps with modf. □

Proposition 4.34 Let $f\colon A \to A'$ be a model morphism and Γ be a collection of conditional $\mathcal{U}$-rules. Then

$$\sim^{A,f}_\Gamma;f \subseteq \sim^{A'}_\Gamma .$$

Proof: By similarity to the proof of Proposition 4.31. □

The following theorem is the main result of this section:

Theorem 4.35 Let $f\colon A \to A'$ be a coequaliser in the category of models and Γ be a collection of conditional $\mathcal{U}$-rules. Then

$$\sim^{A,f}_\Gamma;f = \sim^{A'}_\Gamma .$$

Proof: By Proposition 4.34 it is enough to prove the inclusion $\sim^{A'}_\Gamma \subseteq \sim^{A,f}_\Gamma;f$. We show by induction on $n \in \omega$ that

$$\sim^{A'}_{\Gamma,n} \subseteq \sim^{A,f}_\Gamma;f.$$

For $n = 0$ it is enough to prove that $D_{A'\mathcal{U}} = D_{A\mathcal{U}};f$, since $D_{A\mathcal{U}} \subseteq ker(f)$. The inclusion $D_{A\mathcal{U}};f \subseteq D_{A'\mathcal{U}}$ is obvious. Consider $\langle s', s'\rangle \subseteq D_{A'\mathcal{U}}$. Because $f\mathcal{U}$ is split-epi, there exists s such that $s;f = s'$. Hence $\langle s,s\rangle;f = \langle s', s'\rangle$ and $D_{A'\mathcal{U}} \subseteq D_{A\mathcal{U}};f$.

For the induction step, consider $c_{A'}[l;h'] \sim^{A'}_{\Gamma,n+1} b'$ with

$(\forall B) l \to r$ **if** $\langle s,t\rangle \in \Gamma, h'\colon B \to A'$ a model morphism and c a rewriting context

such that $\langle s;h',t;h'\rangle, \langle c_{A'}[r;h'],b'\rangle \subseteq \sim^{A'}_{\Gamma,n}$.
Because B is coequaliser projective, there exists $h\colon B \to A$ such that $h;f = h'$ and b such that $b;f = b'$.

$$\begin{tikzcd} B \arrow[r, dashed, "h"] \arrow[dr, "h'"'] & A \arrow[d, "f"] \\ & A' \end{tikzcd}$$

$$\begin{aligned} c_{A'}[l;h'] &= c_{A'}[l;h;f] \\ &= l;h;f;c_{A'} \\ &= l;h;c_A;f \qquad \text{(naturality of } c) \\ &= c_A[l;h];f. \end{aligned}$$

Similarly $c_{A'}[r;h'] = c_A[r;h];f$. By the induction hypothesis $\langle s;h,t;h\rangle;f, \langle c_A[r;h],b\rangle;f \subseteq \sim^{A,f}_{\Gamma};f$. Because $\sim^{A,f}_{\Gamma}$ is closed under the rule modf, $\langle s;h,t;h\rangle, \langle c_A[r;h],b\rangle \subseteq \sim^{A,f}_{\Gamma}$. Because $\sim^{A,f}_{\Gamma}$ is closed under Γ-paramodulation , we have $\langle c_A[l;h],b\rangle \subseteq \sim^{A,f}_{\Gamma}$. Hence $c_{A'}[l;h'] = c_A[l;h];f \subseteq \sim^{A,f}_{\Gamma};f$. $\square$

Model theoretic paramodulation together with modf define the concept of **paramodulation modulo a model morphism**. The previous theorem shows that

> Paramodulation modulo a model morphism = paramodulation in the quotient model.

As already mentioned, paramodulation modulo axioms can be regarded as a particular case of paramodulation modulo a model morphism. Actually, by taking the semantic approach on equational theories expressed by Definition 3.6, these two notions appear to be two sides of the same concept. This point of view is supported by regarding the kernel of a model morphism as a theory, or better as the consequences of a theory in the source of the model morphism.

4.5 Confluence

Using the rules of a program as non-oriented equations can lead to very inefficient search within the space of paramodulation chains. A first crucial point in reducing the size of the space of inference chains is to make use of the orientation of the rules. This also adds direction to the refutation, bringing it closer to the true meaning of *computation*. The completeness of paramodulation with oriented rules depends essentially on the confluence of the program. This section explains the relationship between the transitivity of the paramodulation relation determined by a program Γ on a model A and the confluence of Γ as a collection of [oriented] rules.

Confluence (also called the Church-Rosser property[44]) is central to the theory of rewriting. Confluence and termination are essential properties of rewriting systems as models of computation. Confluent and terminating rewriting systems can be used as decision procedures for equality (see [30]). Our concept of confluence for a program generalises the traditional one in the sense that it depends on a given model rather than being fixed (to the model of ground terms).

[44]More precisely, Church-Rosser and confluence are different properties that can be easily proved equivalent in most cases. However, there are some situations when there is a subtle difference between these two properties (see [20]).

4.5.1 Model theoretic rewriting

Any program determines a rewriting relation on the underlying domain of any model:

Definition 4.36 Let Γ be a collection of conditional $\mathcal{U}$-rules. Then a binary relation $\gg$ on a model A is closed under Γ-**rewriting** iff for any rule $(\forall B)l \to r$ **if** $\langle s, t\rangle$ in Γ and for any morphism $h\colon B \to A$,

$$c_A[l;h] \gg c_A[r;h] \text{ if } s;h \sim_\Gamma^A t;h$$

for any rewriting $\mathcal{U}$-context c. The least relation on A closed under reflexivity, transitivity and Γ-rewriting is denoted as $\gg_\Gamma^A$. □

Fact 4.37 Let Γ be a collection of conditional $\mathcal{U}$-rules. For any model A, $\gg_\Gamma^A$ exists and is given by

$$\gg_\Gamma^A = (\rho_\Gamma^A)^* \text{ where } \rho_\Gamma^A = \bigcup\{\langle c_A[l;h], c_A[r;h]\rangle \mid s;h \sim_\Gamma^A t;h\},$$

i.e., $\gg_\Gamma^A$ is the transitive-reflexive closure of the least relation closed under Γ-rewriting. □

In Definition 4.36, h plays the rôle of the *matcher* for the left-hand side of a rule to an element of the algebra. For example, in the case of the OBJ system, the algebra A is the initial algebra of ground terms (or the initial algebra of a theory for the case of rewriting modulo axioms). In this case, h matches the left-hand side of a rule in the program with a subterm of the term to be rewritten. But the rewriting is done *only after* the system *proves* the validity of the hypotheses instantiated by the matcher h. The algebraic formulation of this last condition is given by $s;h \sim_\Gamma^A t;h$, since $\sim_\Gamma^A$ contains exactly all identities in A that can be proved from Γ by paramodulation.

The following result shows that the rewriting relation is preserved under model morphisms:

Proposition 4.38 Let Γ be a collection of conditional $\mathcal{U}$-rules. For any model morphism $f\colon A \to A'$

$$\gg_\Gamma^A; f \subseteq \gg_\Gamma^{A'} .$$

Proof: Consider

$$(\forall B)l \to r \text{ if } \langle s, t\rangle \in \Gamma, h\colon B \to A \text{ a model morphism and } c \text{ a rewriting context}$$

such that $s;h \sim_\Gamma^A t;h$. By Proposition 4.31, $s;h;f \sim_\Gamma^{A'} t;h;f$, hence $c_A[l;h];f = c_{A'}[l;h;f] \gg_\Gamma^{A'} c_{A'}[r;h;f] = c_A[r;h];f$. □

Definition 4.39 Let Γ be a collection of conditional $\mathcal{U}$-rules and $f\colon A \to A'$ be a model morphism. The binary relation $\gg$ on A is closed under Γ-**rewriting modulo** f iff for any rule $(\forall B)l \to r$ **if** $\langle s, t\rangle$

$$c_A[l;h] \gg c_A[r;h] \text{ iff } s;h \sim_\Gamma^{A,f} t;h.$$

The least relation on A closed under reflexivity, transitivity, Γ-rewriting modulo f, and modf is denoted $\gg_\Gamma^{A,f}$. □

The following result is in the spirit of Section 4.4 and it shows that rewriting modulo a model morphism[45] is the same as rewriting in the quotient model.

Theorem 4.40 Let Γ be a collection of conditional $\mathcal{U}$-rules and f be a coequaliser in the category of models. Then

$$\gg_\Gamma^{A,f}; f = \gg_\Gamma^{A'} .$$

Proof: By similarity to the proof of Theorem 4.35 and by using this theorem for $\sim_\Gamma^{A,f}; f = \sim_\Gamma^{A'}$. □

4.5.2 Transitivity *versus* confluence

For this section we assume the category $\mathbb{X}$ of domains has disjoint coproducts (see Definition 2.12).

Lemma 4.41 Let Γ be a collection of conditional atomic $\mathcal{U}$-rules and A be any model. For any a, b, b' finite, if $a \gg_\Gamma^A b$ and $b \sim_\Gamma^A b'$, then $a \sim_\Gamma^A b'$.

Proof: If $a \gg_\Gamma^A b$, then because $\langle a, b\rangle$ is finite and by Fact 4.37 and Proposition 2.38, there exists $n \in \omega$ such that $\langle a, b\rangle \subseteq (\rho_\Gamma^A)_n$. We prove by induction on $n \in \omega$ that if $\langle a, b\rangle \subseteq (\rho_\Gamma^A)_n$ and $b \sim_\Gamma^A b'$, then $a \sim_\Gamma^A b'$. For the induction step, suppose that $\langle a, b\rangle \subseteq (\rho_\Gamma^A)_{n+1}$. By Lemma 2.14 we may assume that $\langle a, b\rangle \subseteq \rho_\Gamma^A \circ (\rho_\Gamma^A)_n$. In the virtue of Fact 2.36, we may further assume that there exists d such that $\langle a, d\rangle \subseteq \rho_\Gamma^A$ and $\langle d, b\rangle \subseteq (\rho_\Gamma^A)_n$. By Lemma 2.14 and because of the atomicity of the rules in Γ, we may assume that

$$a = c_A[l; h] \quad \text{with} \quad \langle c_A[r; h], b\rangle \subseteq (\rho_\Gamma^A)_n \quad \text{and} \quad s; h \sim_\Gamma^A t; h$$

for some rule $(\forall B) l \to r$ **if** $\langle s, t\rangle$ in Γ, for some model morphism $h : B \to A$ and for some rewriting $\mathcal{U}$-context c, and such that $b \sim_\Gamma^A b'$. By the induction hypothesis, $c_A[r; h] \sim_\Gamma^A b'$. Because $\sim_\Gamma^A$ is closed under Γ-paramodulation, $a \sim_\Gamma^A b'$. □

The following result describes the paramodulation relation $\sim_\Gamma^A$ as the "set" of pairs of elements that can be rewritten to the same "element". This intuition constitutes the basis for using term rewriting systems as a decision procedure for equality.

Proposition 4.42 Let Γ be a collection of conditional atomic $\mathcal{U}$-rules. Then for any model A

$$\sim_\Gamma^A = \bigcup\{\langle a, a'\rangle \text{ finite} \mid a \gg_\Gamma^A d \text{ and } a' \gg_\Gamma^A d \text{ for some } d\}.$$

Proof: We first show by induction on $n \in \omega$ that

$$\sim_{\Gamma,n}^A \subseteq \bigcup\{\langle a, a'\rangle \text{ finite} \mid a \gg_\Gamma^A d \text{ and } a' \gg_\Gamma^A d \text{ for some } d\}$$

For the induction step, let's suppose that $a \sim_{n+1} b$ with

$$a = c_A[l; h] \text{ for some } (\forall B) l \to r \textbf{ if } \langle s, t\rangle \in \Gamma, h : B \to A \text{ and } c \text{ rewriting context}$$

[45] Or modulo axioms; see the discussion ending Section 4.4.

such that $s;h \sim^A_{\Gamma,n} t;h$ and $c_A[r;h] \sim^A_{\Gamma,n} b$. By the induction hypothesis, $c_A[r;h] \gg d$ and $b \gg d$ for some d because $\langle c_A[r;h], b\rangle$ is finite and $\{\langle a, a'\rangle \text{ finite} \mid a \gg d, a' \gg d\}$ is filtered (by using Lemma 2.26). Because $s;h \sim^A_\Gamma t;h$, we also have that $c_A[l;h] \gg d$. Then $\langle a, b\rangle \subseteq \bigcup\{\langle a, a'\rangle \mid a \gg^A_\Gamma d \text{ and } a' \gg^A_\Gamma d \text{ for some } d\}$.

For the opposite inclusion, we apply Lemma 4.41. Consider a, a' finite such that $a \gg^A_\Gamma d$ and $a' \gg^A_\Gamma d$ for some d. Then $a \sim^A_\Gamma d$ and, consequently, $a \sim^A_\Gamma a'$. □

Definition 4.43 Consider a model A and Γ a collection of conditional $\mathcal{U}$-rules. Γ is A-**confluent** iff the rewriting relation $\gg^A_\Gamma$ is confluent. □

The notion of A-confluence represents a generalisation of the traditional notions of confluence in the theory of term rewriting systems.[46] The simplest and best known one corresponds to the case when A is the [initial] algebra of ground terms T_Σ for an MSA signature Σ. In Section 4.5.3 we explain the relationship between A-confluence and the notion of *confluence modulo an equivalence* as presented in [55, 61, 20, 30].

The following establishes the crucial link between the confluence of Γ and the transitivity of the paramodulation relation induced by Γ:

Proposition 4.44 Consider a model A and Γ a collection of conditional atomic $\mathcal{U}$-rules. Then Γ is A-confluent iff $\sim^A_\Gamma$ is transitive.

Proof: Assume Γ is A-confluent and consider $a \sim b \sim c$ finite. In the virtue of Proposition 4.42, there exists d, d' such that $a \gg d$, $b \gg d$, $b \gg d'$ and $c \gg d'$. By the confluence of $\gg$, there exists d'' such that $d \gg d''$ and $d' \gg d''$. Thus, $a \gg d''$ and $c \gg d''$. By applying again Proposition 4.42, $a \sim c$.

For the converse, let's assume that $\sim^A_\Gamma$ is transitive and consider $a \gg b$ and $a \gg c$ with a, b, c finite. By Lemma 4.41, $a \sim b$ and $a \sim c$. Therefore, $b \sim c$, and $b \gg d$, $c \gg d$ for some d by Proposition 4.42. □

The following corollary shows that in the case of confluence, the refutation procedure using paramodulation and reflexivity is complete for oriented rules. In the presence of confluence the application of the rules in $\overline{\Gamma}$ (i.e., the backward application of the rules in Γ) is no longer necessary.

Corollary 4.45 Let A be a reachable model and Γ be a collection of A-confluent conditional atomic $\mathcal{U}$-rules. Further assume that any reflexive-symmetric-transitive relation in the category of domains is an equivalence. Then $\sim^A_\Gamma = \equiv^A_\Gamma$.

Proof: By applying Proposition 4.22. □

Through the Lifting Lemma 4.29 we can apply the previous result to the case of paramodulation in MSA:

Corollary 4.46 Let Σ be an algebraic signature. If Γ is a confluent collection of conditional Σ-rules, then the refutation procedure through reflexivity and paramodulation using $\Gamma \cup F(\Sigma)$ is complete. □

[46] See [30] for a detailed exposition of the concept of confluence for term rewriting systems. Other important surveys are [20, 56].

4.5.3 Confluence modulo a Model Morphism

In this section we argue that the notion of A-confluence (Definition 4.43) corresponds to confluence of rewriting on equivalence classes in the case of a quotienting morphism defined by a theory.

The notion of rewriting on the congruence classes (called **class-rewriting** in the survey [20]) was introduced by Lankford and Ballantyne [65] for *permutative* congruences, that is congruences for which each congruence class is finite. For example, associativity and/or commutativity gives rise to permutative congruences.

Let Σ be an algebriac signature and let E be a collection of Σ-equations. In the context of the definitions introduced by Section 4.4, let A be the algebra of ground terms T_Σ, A' be the initial Σ, E-algebra $T_{\Sigma,E}$, and $f: T_\Sigma \to T_{\Sigma,E}$ be the quotienting morphism. Rewriting (paramodulation) modulo f is the same as rewriting (paramodulation) modulo E. Given a collection Γ of conditional Σ-rules, class-rewriting relation defined by Γ and E (denoted Γ/E in [20, 61] is $\gg_\Gamma^{T_\Sigma,f}$. By Theorem 4.40 we have the following:

Corollary 4.47 A term rewriting system Γ is confluent modulo axioms E iff it is $T_{\Sigma,E}$-confluent. □

The term rewriting literature contains several papers [55, 61] and surveys [56, 20] studyign alternative notions of confluence modulo axioms and their relationship with confluence of rewriting on congruence classes.

4.6 Narrowing in MSA

This section is entirely devoted to the application of the general theory developed in Sections 4.3 and 4.5 to the particular case of many sorted narrowing including some of its refinements. Although all results of this section had been established before, the way they fall as a direct consequence of the general category-based results on the completeness of paramodulation is new and can be taken as an example for applying the theory developed in Sections 4.3 and 4.5 to other cases of interest.

The structure of this section is influenced by the gradual development of the completeness results for different refinements of narrowing given in [54]. We fix an MSA signature Σ. The rôle of the model A is now played by the [initial] algebra of the ground terms.

Definition 4.48 A Σ-rule $(\forall X)l \to r$ **if** $\langle s, t\rangle$ is a **rewrite rule** iff $var^{47}(r) \cup var\langle s, t\rangle \subseteq var(l) = X$. It is **collapse free**[48] iff l is not a variable. □

The main difference between rewrite rules and oriented equations (or simply rules) is that in the case of the former the system doesn't have to "invent" values for the variables that might occur in the right hand side or in the condition of a rule but not in its left hand side. This makes rewriting systems appropriate for computations by giving direction to rewriting. An important consequence is the fact that the presence of the functional reflexive axioms is no longer necessary:

[47] By $var(t)$ we mean the set of all variables occurring in the term t. More formally, $var(t)$ is the least set X such that $t \in T_\Sigma(X)$ (see [30]).

[48] An interesting discussion on the rôle played by this concept for the completeness of paramodulation can be found in [54].

Lemma 4.49 Let Γ be a rewriting system and $\mathbf{G}$ be a list of goals. Then $\mathbf{G} \xrightarrow{\Gamma \cup F(\Gamma)}\!\!\!\!\rightarrow^*_R \Box$ implies that $\mathbf{G} \xrightarrow{\Gamma}\!\!\!\!\rightarrow^*_R \Box$.

Proof: Any application of a paramodulation step with a clause from $F(\Gamma)$ would produce a non-empty answer substitution, therefore they are not used in the refutation. $\Box$

The application of Corollary 4.45 requires a new version of Lifting Lemma adequate to the new context. Recall (see [30], for example) that a term t is said to be in **normal form** when no rewriting can be applied to t anymore. A substitution is said to be in normal form iff all its terms are in normal form.

Proposition 4.50 Lifting Lemma Let Γ be a collapse free rewriting system and θ be a substitution in normal form. If $\mathbf{G}\theta \xrightarrow{\Gamma}\!\!\!\!\rightarrow^n_R \Box$ then $\mathbf{G} \xrightarrow{\Gamma}\!\!\!\!\rightarrow^n_{n,r} \Box$ with the computed answer substitution σ such that $\theta \leq \sigma$.

Proof: We prove this lifting lemma by induction on $n \in \omega$.

The first case occurs when the first step of the refutation chain is the removal of an identity $t\theta = t'\theta$ with $\langle t_1, t_2 \rangle \in \mathbf{G}$. Let φ be the most general unifier of t_1 and t_2. Then there exists θ' such that $\varphi\theta' = \theta$. θ' is in normal form since θ is in normal form. If $\mathbf{G} = \mathbf{G}' \cup \{\langle t_1, t_2 \rangle\}$, then since $(\mathbf{G}'\varphi)\theta' \xrightarrow{\Gamma}\!\!\!\!\rightarrow^{n-1}_R \Box$, the induction hypothesis implies that $\mathbf{G}'\varphi \xrightarrow{\Gamma}\!\!\!\!\rightarrow^{n-1}_{n,r} \Box$ with the computed answer substitution σ' such that $\theta' \leq \sigma'$. But $\mathbf{G} \longrightarrow\!\!\!\!\rightarrow_r \mathbf{G}'\varphi$ with φ the computed answer substitution. The answer substitution computed by the refutation $\mathbf{G} \xrightarrow{\Gamma}\!\!\!\!\rightarrow^n_{n,r} \Box$ is $\varphi\sigma' \geq \varphi\theta' = \theta$.

The second case occurs when the first step of the refutation is a proper rewriting step. Then $\mathbf{G}\theta \longrightarrow\!\!\!\!\rightarrow_R \mathbf{G}\theta|_{\pi \leftarrow r\varphi} \cup \langle s, t \rangle\varphi$ for $(\forall X) l \rightarrow r$ **if** $\langle s, t \rangle$ a new variant of a clause in Γ and $l\varphi = \mathbf{G}\theta|_\pi$.

- θ in normal form implies that $(\mathbf{G}\theta)|_\pi = (\mathbf{G}|_\pi)\theta$, i.e., π is a basic occurence,
- Γ collapse free implies that $\mathbf{G}|_\pi$ is *not* a variable, and
- $dom\varphi \cap dom\theta = \emptyset$ implies the existence of θ' in normal form such that $\varphi'\theta' = \theta + \varphi$ where $\varphi' = mgu(l, \mathbf{G}|_\pi)$. This works because $l(\theta+\varphi) = l\varphi = (\mathbf{G}|_\pi)\theta = (\mathbf{G}|_\pi)(\theta+\varphi)$. θ' is in normal form because both θ and φ are in normal form.

Then $\mathbf{G} \longrightarrow\!\!\!\!\rightarrow_n \mathbf{G}|_{\pi \leftarrow r\varphi'} \cup \langle s, t \rangle\varphi'$ and $(\mathbf{G}|_{\pi \leftarrow r\varphi'} \cup \langle s, t \rangle\varphi')\theta' = \mathbf{G}\theta|_{\pi \leftarrow r\varphi} \cup \langle s, t \rangle\varphi \longrightarrow\!\!\!\!\rightarrow^{n-1}_R \Box$. Now, we can apply the induction hypothesis in the same manner with the former case (when the first refutation step was a reflection) and deduce the conclusion of this lemma. $\Box$

Corollary 4.51 Let Γ be a confluent collapse free rewriting system. The refutation procedure through reflexivity and narrowing is complete. $\Box$

4.6.1 Canonical term rewriting systems

This paragraph reviews the completeness of basic innermost narrowing from [54]. This works under the further assumption of the termination of the term rewriting system involved. A rewriting system that is both confluent and terminating is called **canonical**.

The completeness of basic innermost narrowing is obtained directly from Corollary 4.51 via the following[49]:

Proposition 4.52 [54] Let Γ be a canonical collapse free term rewriting system and $\mathbf{G}$ be a list of goals. Then $\mathbf{G} \xrightarrow{\Gamma} \twoheadrightarrow^*_{n,r} \Box$ implies that $\langle \mathbf{G}, \epsilon \rangle \xrightarrow{\Gamma} \twoheadrightarrow^*_{r,in,ir} \Box$ with the same computed answer substitution.

Sketch of Proof: When the substitution θ is in normal form, in $\mathbf{G}\theta \xrightarrow{\Gamma} \twoheadrightarrow^*_R \Box$ rewriting is applied only at basic occurrences. The canonicity of Γ implies the we can select a chain of innermost rewrites. Innermost reflection is needed to move to redeces above an innermost redex with respect to $\longrightarrow\!\!\!\!\rightarrow_R$ because innermost redeces with respect to $\longrightarrow\!\!\!\!\rightarrow_n$ might not correspond to innermost redeces with respect to $\longrightarrow\!\!\!\!\rightarrow_R$. $\Box$

Basic innermost narrowing can be combined with rewriting on the goals. As discussed in [54], this can be very beneficial in cutting off non-terminating narrowing chains. In some cases it also adds to the efficiency of the computation. The completeness of basic innermost narrowing combined with rewriting follows directly from Corollary 4.51 via the following:

Proposition 4.53 [54] Let Γ be a canonical collapse free term rewriting system and let $\mathbf{G}$ be a list of goals. If $\mathbf{G} \xrightarrow{\Gamma} \twoheadrightarrow^*_{n,r} \Box$ then $\langle \mathbf{G}, \epsilon \rangle \xrightarrow{\Gamma} \twoheadrightarrow^*_{R,r,in,ir} \Box$ with the same computed answer substitution and narrowing applied only to normalised goals. $\Box$

4.7 Computing in Eqlog

The Oxford prototype implementation of Eqlog is an extension of the OBJ3 system (developed at SRI International; its user manual is [46]). The Eqlog system adds an implementation of order sorted basic leftmost innermost narrowing. The current goal to be processed is selected to be the leftmost one from the goal list. The goals are represented in the manner of Definition 4.12, i.e., having a skeleton part and an environment part representing the accumulation of the computed answer substitutions. The main narrowing loop implements a depth-first search on the space of all narrowing chains regarded as a search tree.

4.7.1 OS unification in Eqlog

The implementation of unification follows the order sorted version of Martelli-Montanari algorithm described in [74]. It is known (see [74]) that an order sorted unification problem may fail because of the sort structure. In some cases, this can dramatically speed up the whole computation because most of the computation time is spent on failing unifications. On the other hand, a successful unification problem might have a finite *most general solution set* (see [74, 27]) rather than a single most general unifier. However, the following property of OSA signatures assures the existence of a most general unifier for any successful unification problem:

Definition 4.54 A monotonic OSA signature $(S, \leq, \Sigma)$ is **coregular** (called unitary in [74]) iff

[49] Using the environment-skeleton representation of goals described in Section 4.2.

1. for any two sorts $s, s' \in S$ there is at most one maximal common subsort, and

2. for any operator symbol $\sigma \in \Sigma$ and any sort $s \in S$, the set $\{w \in S^* \mid \sigma \in \Sigma_{w,s'}$ and $s' \leq s\}$ has at most one maximal element.

□

Although the Eqlog system assumes that all signatures of modules are coregular,[50] it also has a facility for showing the eventual non-coregularities of a signature of the current module. One types

```
set show noncoreg on .
```

to turn it on and,

```
set show noncoreg off .
```

to turn it off.

4.7.2 Examples with narrowing

Consider the following module defining an ADT of lists over a set of elements (represented here by the sort `Elt`). The non-empty lists form a subsort `NList` of the sort of all lists (i.e., `List`). The empty list and the usual list selectors have the same name as their Lisp counterparts, while the constructor function (`cons` in Lisp) is simply denoted by concatenation. In order to get a purely logical inference procedure for this example we have to use an ADT definition for the natural numbers rather than import them as built-ins.[51] The function giving the length of a list is denoted by `#`.

`0`, `s`, `nil`, and `__` are declared as constructors.

```
obj LIST is
  sorts Elt Nat NList List .
  subsort  NList < List .
  op 0 : -> Nat                  [cons] .
  op s : Nat -> Nat              [cons] .
  op a : -> Elt .
  op nil : -> List               [cons] .
  op __ : Elt List -> NList      [cons] .
  op car : NList -> Elt .
  op cdr : NList -> List .
  op #_ : List -> Nat .
  var E : Elt .
  var L : List .
  eq car(E L) = E .
  eq cdr(E L) = L .
  eq # nil = 0 .
  eq #(E L) = s(# L) .
endo
```

[50] Experiments made in Oxford showed that the vast majority of OBJ modules are coregular.
[51] The Eqlog system inherits the built-in natural numbers from the OBJ system.

By typing

```
set show narrowing on .
```

the user can see the actual inference steps performed by the Eqlog system which alternates reflection and basic leftmost innermost narrowing steps. Successful reflection steps are omitted. The meaning of all fields is obvious except for `next-position`, which refers to the occurrence at which the redex of the next narrowing step has to be found. This occurrence is a list of natural numbers representing the path to the redex within the tree underlying the term if the search process backtracks, otherwise is still unknown.

For example, the query

```
find Lst : List such that # Lst = s(s(0)) ; car(Lst) = a .
```

produces the following output:

```
##########################################
solve in % :
car(Lst) = a
# Lst = s(s(0))

reflection failed
------------------------------------------
depth in the narrowing chain: 1
current goal list (skeleton):
E_978 = a
# Lst = s(s(0))
current answer substitution:
E_978: Elt -> UNBOUND
L_977: List -> UNBOUND
Lst: NList -> E_978 L_977
next-position: unknown
------------------------------------------
depth in the narrowing chain: 2
current goal list (skeleton):
s(# L_983) = s(s(0))
current answer substitution:
E_984: Elt -> a
E_978: Elt -> a
L_977: List -> L_983
Lst: NList -> a L_983
next-position: unknown

reflection failed
------------------------------------------
depth in the narrowing chain: 3
current goal list (skeleton):
s(0) = s(s(0))
current answer substitution:
```

```
E_984: Elt -> a
L_983: List -> nil
E_978: Elt -> a
L_977: List -> nil
Lst: NList -> a nil
next-position: unknown

reflection failed

constructor clash
-----------------------------------------
depth in the narrowing chain: 3
current goal list (skeleton):
s(s(# L_989)) = s(s(0))
current answer substitution:
E_984: Elt -> a
L_983: List -> E_990 L_989
E_978: Elt -> a
E_990: Elt -> UNBOUND
L_977: List -> E_990 L_989
Lst: NList -> a (E_990 L_989)
L_989: List -> UNBOUND
next-position: unknown

reflection failed
-----------------------------------------
depth in the narrowing chain: 4
current goal list (skeleton):
s(s(0)) = s(s(0))
current answer substitution:
E_984: Elt -> a
L_983: List -> E_990 nil
E_978: Elt -> a
E_990: Elt -> UNBOUND
L_977: List -> E_990 nil
Lst: NList -> a (E_990 nil)
L_989: List -> nil
next-position: unknown

A solution is:
Lst: NList -> a (E_990 nil)
```

This example also shows how the sorts of the logical variables are dynamically changed during the computation process. The Eqlog system accepts a certain class of badly typed terms in queries which are treated by using the method of retracts,[52] but this is hidden to the user. In our example, accordingly to the original declaration of the type of the logical variable `Lst`, the term `car(Lst)` is not well typed because `car` is defined only on

[52]Inherited from the OBJ3 system; for a detailed discussion on retracts and their semantics see [34].

the subsort `NList` of the non-empty lists. However, during the computation process the order sorted unification function changes the sort of `Lst` to `NList`. This could be easily noticed in the first narrowing step performed by the system, and also shows up in the final result.

4.7.3 Constructor discipline

Consider the following query:

```
find Lst : NList such that # Lst = 0 .
```

Because the rule `# nil = 0` would never be selected due to the type constraint on `Lst`, the system proceeds into endless applications of the rule `#(E L) = s(# L)`.

However, such a situation could be easily avoided by noticing that there is no possible refutation from goals of the form `s(...) = 0`. This suggests a **constructor discipline** as a way to stop non-terminating computations and also as a way to reduce the search within the space of narrowing chains. Although the constructor discipline is used in equational logic programming as a control facility (the programmer has the full option to declare some operations as constructors), the concept of constructor has a precise mathematical meaning at the level of algebraic specifications. In [73], Meseguer and Goguen showed that only order sorted algebra solves the constructor-selector problem.

Definition 4.55 [30] A subsignature $\Omega \subseteq \Sigma$ is a **subsignature of constructors** for a specification (Σ, E) iff $T_{\Sigma,E}\restriction_{\Omega}$ is a reachable Ω-algebra. A **subsignature of unique constructors** is a subsignature of constructors Ω such that $T_{\Sigma,E}$ is the initial (i.e., ground terms) Ω-algebra. □

The main principle underlying any constructor discipline for equational logic programming can be concisely formulated as follows:

Constructors cannot be narrowed.

The Eqlog system implements this principle in two different ways.[53] The first one occurs when the topmost operators of the sides of a goal are different constructors.[54] In this case, since it is impossible to develop the narrowing chain into a refutation, the computation backtracks[55]. The second way to apply the constructor discipline is to banish from narrowing the positions where the corresponding operator is a constructor. The main consequence in this case is to speed up of the computation of innermost redeces.

[53] Many other implementations of narrowing embed some sort of constructor discipline, notably the ALF system [49].

[54] Actually, the Eqlog system implements a stronger version of this: before a narrowing step is performed, the system tries to find the outermost occurrence at which the corresponding operators are different constructors, and such that all outer positions are occupied by constructors within both sides of the goal. If such a position is found, then the computation backtracks without trying to perform the narrowing step.

[55] In the previous example of an Eqlog run, this corresponds to the message `constructor clash`.

5 Modularisation

A promising approach to developing large and complex systems (which may be software, hardware, or both) is to start from a description of the system as an interconnection of some specification modules. This permits the verification of many properties to be carried out at the level of design rather than code, and thus should improve reliability. With suitable mechanical support, it might also improve the efficiency of the development process. In addition, it promotes reuse, because some modules may be taken directly from a library, or else may be modifications of library modules. For this reason, many modern programming and specification languages support some form of modularisation, and most mathematical results about modules have appeared in the context of formal software engineering, particularly specification languages. There has been much recent interest in module composition systems under the name of "megaprogramming" [98, 94].

Modularisation for equational logic programming has been studied less. Two basic problems are the *soundness* and *completeness* of the translation of queries and their solutions along module imports. It is important to notice that in ELP the notion of module is very similar to that in equational (i.e., functional) programming[56] and, although each query is related to a certain module, the query is not part of the module. Given a module import $P \xrightarrow{\psi} P'$ (technically regarded as a morphism of theories), ψ is sound iff for any query q in P, any of its solutions is translated to an solution of $\psi(q)$. The completeness of ψ means that any solution of $\psi(q)$ corresponds to a solution of q.[57] Our notion of module import is *not* restricted only to inclusion of theories, a module import could be any morphism of theories. In this context, we prove the soundness property for arbitrary module imports.

A particularly important relation between theories is that of *conservative extension*, which says that any model of a subtheory can be expanded to a model of the supertheory. This semantic property can be important for the reuse of modules. Other semantic properties of extensions arise in connection with parameterised (i.e., generic) modules. The completeness property is proven to hold for the case of essentially persistent module imports.[58]

The theory of institutions [33] provides an abstract mathematical formulation of the concept of 'logical system' very adequate for the study of modularisation in declarative programming languages rigorously based on logical systems. In order to use the machinery provided by the theory of institutions to modularisation problems specific to equational logic programming, we have to integrate the framework of category-based equational logics with institutions. The institution of category-based equational logics provides the most abstract framework which is still concrete enough to deal with concepts like queries and solutions. The primary mathematical structure in this approach is the notion of Kleisli category. Translations of queries along module imports appear as functors between Kleisli categories. The more general case of quantifiers as models (rather than collections of variables) reveals that the translations of the quantifiers along

[56] For example, there are only very small diferences between Eqlog and OBJ modules.

[57] Section 5.3.2 shows how soundness and completeness of module imports relates to the traditional concept of soundness and completeness for logical systems.

[58] A property stronger than conservative extension.

module imports are simply free constructions.

The institution of category-based equational logics is abstract enough to encode equational logic programming modules as signatures and module imports as morphisms of signatures. This different level of use of the institution of category-based equational logics is the basis for a category-based semantics for equational logic programming queries and solutions. The institution of category-based equational logics also supports a category-based version of the Theorem of Constants. We place this result here exactly because of its connection to the basic mathematical structures of this chapter, although in principle it is not related to modularisation issues. The model-theoretic dimension of our more general version of the Theorem of Constants is also related to the so-called "method of diagrams" from classical first-order model theory.

The soundness and completeness problem for translations of queries and their solutions along module imports is shown to be an instantiation of the soundness and completeness at the level of institutions with an entailment system. This fact resorts to a special and rather eccentric institution having collections of logical variables as signatures, queries as models, and substitutions as sentences. A substitution is an answer for a query iff the query satisfies the substitution. The only inference rule defining the entailment relation encodes the translation of substitutions along module imports.

5.0.4 Some History

The earliest work on software modules with which we are familiar is by Parnas [77, 78, 79]. Program modules differ from earlier program structuring mechanisms such as subroutines, procedures and blocks, in that they may include a number of procedure and data definitions, may be parameterised, may import other modules, and may hide certain elements. A major motivation for modules in this sense is to facilitate the modification of software, by localizing the representation of data and the operations that depend upon it; this is called *information hiding*. Such modules support software reuse because they can be specified, verified, and compiled separately. Note that this notion of module is essentially *syntactic*: it concerns texts that describe systems.

The earliest work that we know on specification modules is by Goguen and Burstall, for their specification language Clear [12, 13], the semantics of which is based on institutions.[59] This approach to modules has been applied to various logic-based languages, particularly OBJ [46], Eqlog [38], FOOPS [40, 47] (which combines the functional and object paradigms), and FOOPlog [40] (which combines functional, logic and object paradigms); it could also be applied to any pure logic-based programming language, such as (pure) Lisp and (pure) Prolog. In [26], this is even extended to imperative programming. The module system of (Standard) ML [50] has also been strongly influenced by this work on Clear.

Clear introduced the ideas that a specification module determines a theory, and that such theories can be put together using colimits; these ideas have their origin in some earlier work by Goguen on General Systems Theory [23, 36]. Clear provided operations for summing, renaming, extending, hiding, importing and (in the case of generics) applying theories. Theories in turn denote classes of models. The earliest work that we know giving a calculus of modules is also due to Goguen and Burstall [31]. Building on Clear, they studied laws for *horizontal* structuring relationships, and *vertical* implementing (also called "refinement") relationships, concluding that the axioms of a 2-category should be

[59] Other early work on modules for specification languages was by Liskov on the language CLU [3].

satisfied.[60] Some general laws for the module operations of Clear appear in [23], and others occur in the proofs in [13]. Some recent results on the formal properties of module composition over institutions appear in [29].

The *module algebra* of Bergstra, Heering and Klint [9] attempts to capture the horizontal structure of modules with equations among certain basic operations on modules, including sum, renaming, and information hiding. These equations, together with constructors for signatures and sentences, give a many sorted equational presentation, about which some interesting results can be proved, including a normal form theorem. Unfortunately, this work has first order logic built into its choice of the constructors for signatures and sentences. However, Bergstra *et al.* abstract some interesting general principles from this special case. [21] develops a module algebra in the context of the theory of institutions. In [21] it is shown that all reasonable institutions support certain simple operations on theories; what properties ensure that these operations have various desirable properties is also explored. A new categorical axiomatisation of the notion of inclusion permits simple definitions for these operations on theories.

Much interesting work using institutions has been done by Tarlecki [89, 90, 91, 92] and by Sannella and Tarlecki [83, 84, 85].

5.1 Institutions and Modularisation

Institutions are much more abstract than Tarski's model theory, and they also add another basic ingredient, namely signatures and the possibility of translating sentences and models from one signature to another. A special case of this translation may be familiar from first order model theory: if $\Sigma \to \Sigma'$ is an inclusion of first order signatures, and if M is a Σ'-model, then we can form $M{\restriction}_\Sigma$, called the *reduct* of M to Σ. Similarly, if e is a Σ-sentence, then we can always view it as a Σ'-sentence (but there is no standard notation for this). The key axiom, called the Satisfaction Condition, says that *truth is invariant under change of notation*, which is surely a very basic intuition for traditional logic.

Definition 5.1 An **institution** $\Im = (\mathbb{S}ign, Sen, \textsc{Mod}, \models)$ consists of

1. a category $\mathbb{S}ign$, whose objects are called **signatures**,
2. a functor $Sen : \mathbb{S}ign \to \mathbb{S}et$, giving for each signature a set whose elements are called **sentences** over that signature,
3. a functor $\textsc{Mod} : \mathbb{S}ign^{op} \to \mathbb{C}at$ giving for each signature Σ a category whose objects are called Σ-**models**, and whose arrows are called Σ-(**model**) **morphisms**, and
4. a relation $\models_\Sigma \subseteq |\textsc{Mod}(\Sigma)| \times Sen(\Sigma)$ for each $\Sigma \in |\mathbb{S}ign|$, called Σ-**satisfaction**,

such that for each morphism $\phi : \Sigma \to \Sigma'$ in $\mathbb{S}ign$, the **Satisfaction Condition**

$$M' \models_{\Sigma'} Sen(\phi)(e) \quad \text{iff} \quad \textsc{Mod}(\phi)(M') \models_\Sigma e$$

holds for each $M' \in |\textsc{Mod}(\Sigma')|$ and $e \in Sen(\Sigma)$. $\square$

[60] This is consistent with the fact that in our category-based semantics for queries and solutions, the category of modules and module imports comes equipped with a 2-categorical structure induced by the 2-categorical structure of the category-based equational signatures.

We will often denote the reduct functor $\textsc{Mod}(\phi)$ by $\_\restriction_\phi$, and the sentence translations $Sen(\phi)$ simply by $\phi(\_)$ or even $\_\phi$.

All logics presented in Section 2.3 are institutions. Once a logic is proved to be an institution, OBJ-like modularisation principles can be applied to any programming language rigorously based on that logic. [21] contains a series of results obtained at the level of institution theory and supporting OBJ-like protecting module imports and parameterised (generic) programming.

Definition 5.2 A **theory** (Σ, E) in an institution $\Im = (\mathbb{S}ign, Sen, \textsc{Mod}, \models)$ consists of

- a signature Σ, and
- a set E of Σ-sentences closed under semantical deduction, i.e., $e \in E$ if $E \models_\Sigma e$.[61]

A **theory morphism** $\phi\colon (\Sigma, E) \to (\Sigma', E')$ is just a morphism of signatures $\phi\colon \Sigma \to \Sigma'$ such that $Sen(\phi)(E) \subseteq E'$. Let $\mathbb{T}h(\Im)$ denote the subcategory of theories in $\Im$. □

The principle of "initial algebra semantics" is formalised at the level of institutions (see [33]) by the concept of liberality:

Definition 5.3 Let $\Im = (\mathbb{S}ign, Sen, \textsc{Mod}, \models)$ be an institution. A theory morphism ϕ is **liberal** iff the reduct functor $\textsc{Mod}(\phi)$ has a left-adjoint.

The institution $\Im$ is liberal iff all theory morphisms in $\mathbb{T}h(\Im)$ are liberal. □

In general, equational logics tend to be liberal, while first order logics are not liberal. In [89], Tarlecki relates the liberality of an institution to the quasi-variety property which must be fulfilled by the class of models of any theory in that institution, meaning that the models of any theory must be closed under products and submodels.[62]

5.1.1 Exactness

An important model theoretic property of many logical systems is that finite colimits are preserved by the model functor. Thus, if we combine some theories T_i in a diagram $T\colon I \to \mathbb{T}h(\Im)$ having colimit (i.e., result of combination) C, then the denotations of the T_i and C behave in the way one would hope: $\textsc{Mod}(C)$ is the limit of the diagram $T; \textsc{Mod}^{op}\colon I \to \mathbb{C}at$. In particular (and assuming that the categories of Σ-models are concrete), our intuition would lead us to hope that a model of $T_1 \bigoplus T_2$ (the co-product) would consist of a pair of models, one of T_1 and the other of T_2; i.e., we intuitively expect $\textsc{Mod}(T_1 \bigoplus T_2)$ to be $\textsc{Mod}(T_1) \times \textsc{Mod}(T_2)$. The situation is similar for a pushout of theory morphisms $T_0 \to T_1$ and $T_0 \to T_2$, which for simplicity we assume are theory inclusions, so that T_0 is shared between T_1 and T_2: we expect that a model of $T_1 \bigoplus_{T_0} T_2$ (the pushout) can be constructed from a pair of models, one of T_1 and the other of T_2, by identifying their reducts to T_0; that is, we expect $\textsc{Mod}(T_1 \bigoplus_{T_0} T_2)$ to be the pullback of $\textsc{Mod}(T_1) \to \textsc{Mod}(T_0)$ and $\textsc{Mod}(T_2) \to \textsc{Mod}(T_0)$. This property, which we call *exactness*, seems to have first arisen in [85], and is also used in the pioneering work of Tarlecki [91] on abstract algebraic institutions, and of Meseguer [72] on categorical logics[63].

[61] Meaning that $M \models_\Sigma e$ for any Σ-model M that satisfies all sentences in E.

[62] In the case of the usual logical systems, this corresponds exactly to the power of Horn clause axiomatisations.

[63] Meseguer [72] introduced the term exactness, but used it for the concept that we call semiexactness here.

Definition 5.4 An institution is **exact** iff the model functor $\textsc{Mod}\colon \mathbb{S}ign \to \mathbb{C}at^{op}$ preserves finite colimits, and is **semiexact** iff $\textsc{Mod}$ preserves pushouts. $\square$

Although many sorted logics tend to be exact, their unsorted variants tend to be only semiexact. In particular, the model functor does not preserve coproducts for either unsorted first order logic or unsorted equational logic. This is undesirable from the point of view of modularisation. Combining this with the well known fact that the coproduct of unsorted terminating term rewriting systems need not be terminating, although it is terminating in the many sorted case, we might conclude that unsorted logics are unnatural for many applications in Computing Science.

It is not hard to see that any chartered institution is exact.[64] Charters were introduced by Goguen and Burstall [32] as a general way to produce institutions. The basic intuition is that the syntax of a logical system is an initial algebra. Because it appears that most institutions of interest in Computing Science can be chartered, it follows that most institutions of interest in Computing are exact. In particular, both many sorted first order logic and many sorted equational logic are exact. On the other hand, unsorted equational logic is not exact.

Notice that, for any institution $\Im$, the model functor $\textsc{Mod}$ extends to $\mathbb{T}h(\Im)$, by mapping a theory (Σ, E) to the full subcategory $\textsc{Mod}(\Sigma, E)$ of $\textsc{Mod}(\Sigma)$ formed by the Σ-models that satisfy E. The following result shows that one can lift exactness from signatures to theories, so that exactness depends only on the behavior of signatures, and is independent of what happens with sentences. Semiexactness for theories plays an important rôle in the "categorical logics" described by Meseguer in [72]. In [21] it is shown that this follows from the corresponding property for signatures:

Proposition 5.5 If an institution is semiexact, then $\textsc{Mod}\colon \mathbb{T}h \to \mathbb{C}at^{op}$ preserves pushouts.

Proof: Let $\phi_1\colon (\Sigma', E') \to (\Sigma_1, E_1)$ and $\phi_2\colon (\Sigma', E') \to (\Sigma_2, E_2)$ be morphisms of theories and let $\phi_1'\colon (\Sigma_2, E_2) \to (\Sigma, E)$ and $\phi_2'\colon (\Sigma_1, E_1) \to (\Sigma, E)$ be their pushout. Recall from [33] that (ϕ_1', ϕ_2') is the pushout of (ϕ_1, ϕ_2) in Sign and E is the deductive closure of $\phi_2'(E_1) \cup \phi_1'(E_2)$.

$$\begin{tikzcd}
 & (\Sigma_1, E_1) \arrow[dr, "\phi_2'"] & \\
(\Sigma', E') \arrow[ur, "\phi_1"] \arrow[dr, "\phi_2"'] & & (\Sigma, E) \\
 & (\Sigma_2, E_2) \arrow[ur, "\phi_1'"'] &
\end{tikzcd}$$

Let M_1 be a Σ_1-model of E_1 and M_2 a Σ_2-model of E_2 such that $M_1{\restriction}_{\phi_1} = M_2{\restriction}_{\phi_2}$; now let M' denote this Σ'-model. Then by the Satisfaction Condition, M' satisfies E'. By semiexactness and the construction of pullbacks in $\mathbb{C}at$, there is a Σ-model M such that $M{\restriction}_{\phi_2'} = M_1$ and $M{\restriction}_{\phi_1'} = M_2$. By the Satisfaction Condition again, M satisfies the translations of both E_1 and E_2, and thus satisfies E. We have now shown that any pair of

[64] Using the facts that $\textsc{Mod}$ is 2-representable for chartered institutions, and that 2-representable functors preserve colimits.

models (M_1, M_2) with $M_1 \in |\text{MOD}(\Sigma_1, E_1)|$ and $M_2 \in |\text{MOD}(\Sigma_2, E_2)|$ and $M_1 \restriction_{\phi_1} = M_2 \restriction_{\phi_2}$ determines a (Σ, E)-model M.

Conversely, any (Σ, E)-model M is determined in this way by its translations $M_1 = M \restriction_{\phi'_2}$ and $M_2 = M \restriction_{\phi'_1}$ which, by the Satisfaction Condition, satisfy E_1 and E_2, respectively.

Because the models of a theory form a full subcategory of the models of its signature, we can extend this argument to model morphisms. Therefore, $\_\restriction_{\phi'_2} : \text{MOD}(\Sigma, E) \to \text{MOD}(\Sigma_1, E_1)$ and $\_\restriction_{\phi'_1} : \text{MOD}(\Sigma, E) \to \text{MOD}(\Sigma_2, E_2)$ are the pullback of $\_\restriction_{\phi_1} : \text{MOD}(\Sigma_1, E_1) \to \text{MOD}(\Sigma', E')$ and $\_\restriction_{\phi_2} : \text{MOD}(\Sigma_2, E_2) \to \text{MOD}(\Sigma', E')$. □

A proof of the following result was sketched in [85] and given in [21]:

Corollary 5.6 If an institution is exact, then $\text{MOD} : \mathbb{T}h \to \mathbb{C}at^{op}$ preserves finite colimits.

Proof: By exactness, MOD maps the initial object of Sign to the terminal (singleton) category. Because the only model of this category satisfies the empty theory (i.e., the tautologies over the initial signature) we conclude that the model functor maps the initial theory to the terminal category. Now we are done, because all finite colimits can be constructed from pushouts and an initial object. □

5.1.2 Parametric modules and views

Definition 5.7 A theory morphism $\phi : P \to T$ is **conservative** iff for any model $M \in |\text{MOD}(P)|$ there exists a model $N \in |\text{MOD}(T)|$ such that $N \restriction_\phi = M$. □

Persistence is a stronger notion than conservative extension, and is important for the semantics of parameterised data types (e.g., see [33]).

Definition 5.8 A theory morphism $\phi : P \to T$ is **persistent** iff its associated reduct functor $\_\restriction_\phi : \text{MOD}(T) \to \text{MOD}(P)$ has a left adjoint such that each component of the unit of the adjunction is an equality. □

Fact 5.9 A persistent theory morphism is conservative. □

Example 5.10 Consider the following classical example of generic lists over elements of monoids. The monoid operations are abstract and they can be used as generic operations for computations involving all elements of a list.

```
th MON is
  sort Mon .
  op e : -> Mon .
  op _*_ : Mon Mon -> Mon [assoc] .

  var x : Mon .
  eq e * x = x .
  eq x * e = x .
endth
```

```
th LIST*[X :: MON] is
  sort List .
  subsort Mon < List .
  op __ : List List -> List [assoc] .
  op nil : -> List .
  op # : List -> Mon .

  vars L L´ : List .
  eq L nil = L .
  eq nil L = L .
  eq #(nil) = e .
  eq #(L L´) = #(L) * #(L´) .
endth
```

The module `LIST*` imports the module `MON` without introducing any new elements or identifying any old elements. This means that the module import `MON` $\hookrightarrow$ `LIST*` is persistent. This is so because for any monoid M the free `LIST*`-model over M consists of lists with elements from M and its reduct to `MON` gives exactly the original monoid M. $\square$

The following result (from [21]) is related to the semantics of applying a generic module to an actual parameter module using a "view," as proposed in Clear and implemented in OBJ3 and Eqlog:

Proposition 5.11 Given a semiexact institution with pushouts of signatures, let (ϕ', ψ') be the pushout of theory morphisms $\phi\colon P \to T$ and $\psi\colon P \to P'$. Then:

1. If the functor $\_\restriction_\phi\colon \textsc{Mod}(T) \to \textsc{Mod}(P)$ has a left inverse $\phi^\$\colon \textsc{Mod}(P) \to \textsc{Mod}(T)$, then there is a left inverse $\phi'^\$$ of $\_\restriction_{\phi'}$ such that the following diagram commutes:

$$\begin{tikzcd}
\textsc{Mod}(P) \arrow[r, "\phi^\$"] & \textsc{Mod}(T) \\
\textsc{Mod}(P') \arrow[u, "_\restriction_\psi"] \arrow[r, "\phi'^\$"'] & \textsc{Mod}(T') \arrow[u, "_\restriction_{\psi'}"']
\end{tikzcd}$$

2. ϕ' is persistent if ϕ is persistent.

Proof: To show the first assertion, pick an arbitrary model N' of P'. Then $N = N'\restriction_\psi$ is a model of P by the Satisfaction Condition. Let M be $\phi^\$(N)$. Then $M\restriction_\phi = N'\restriction_\psi = N$. By Proposition 5.5, there is a model M' of T' such that $M'\restriction_{\psi'} = M$ and $M'\restriction_{\phi'} = N'$. The mapping $N' \mapsto M'$ defines the functor $\phi'^\$$ on objects, and its definition on arrows is similar. Next, $\phi'^\$$ preserves identities because $1_{M'}\restriction_{\psi'} = \phi'^\$(1_{N'})\restriction_{\psi'}$ and $1_{M'}\restriction_{\phi'} = \phi'^\$(1_{N'})\restriction_{\phi'}$ for any P'-model N'. By Proposition 5.5, $1_{M'} = \phi'^\$(1_{N'})$. The same argument gives the preservation of composition by $\phi'^\$$.

For the second assertion, we will show that $\phi'^\$$ is left-adjoint to $\_\restriction_{\phi'}$ if $\phi^\$$ is left-adjoint to $\_\restriction_\phi$, that is (using the above notations), M' is a free T'-model over N' if M is a free T-model over N. Pick an arbitrary T'-model M'_1 and an arbitrary model morphism

$h\colon N' \to M_1'\restriction_{\phi'}$. We have to prove that there is a unique model morphism $h^\sharp\colon M' \to M_1'$ such that $h^\sharp\restriction_{\phi'} = h$. Notice that by Proposition 5.5, any $h^\sharp\colon M' \to M_1'$ is uniquely determined by its reducts $h = h^\sharp\restriction_{\phi'}\colon N' \to M_1'\restriction_{\phi'}$ and $f = h^\sharp\restriction_{\psi'}\colon M \to M_1'\restriction_{\psi'}$ and by the condition $h\restriction_\psi = f\restriction_\phi$.

$$\begin{tikzcd} N' \arrow[r, equal] \arrow[dr, "h"'] & M'\restriction_{\phi'} \arrow[d, "h^\sharp\restriction_{\phi'}"] \\ & M_1'\restriction_{\phi'} \end{tikzcd} \qquad \begin{tikzcd} N \arrow[r, equal] \arrow[dr, "h\restriction_\psi"'] & M\restriction_\phi \arrow[d, "f\restriction_\phi"] \\ & (M_1'\restriction_{\psi'})\restriction_\phi \end{tikzcd}$$

Now let f be the unique model morphism $M \to M_1'\restriction_{\psi'}$ such that $h\restriction_\psi = f\restriction_\phi$ (since M is free over N). Then the morphism $h^\sharp\colon M' \to M_1'$ determined by (f, h) is the desired extension of h to a model morphism $M' \to M_1'$. □

Example 5.12 Based on Example 5.10, consider the following specification of lists:

```
th List is
  sorts Elt List .
  subsort Elt < List .
  op empty : -> List .
  op append : List List -> List [assoc] .

  var L : List .
  eq append(L , empty) = L .
  eq append(empty , L) = L .
endth
```

The operation `append` is associative and has the empty list as an identity. In this way, `List` is a refinement of the theory of monoids. There is a view from `MON` to `List`:

```
view list from MON to List is
  sort Mon to List .
  op (_*_) to append .
  op e to empty .
endv
```

The instantiation `LIST*[list]` of the generic module `LIST*` via `list` is the pushout of `MON` $\hookrightarrow$ `LIST*` with `list`. In this example, the operation `#` appends all lists from a list of lists. By the previous theorem, `LIST*[list]` protects `List`. This fact can be checked directly as well. □

In this example, `LIST*[list]` is a simple module expression involving essentially only one instantiation of a generic module. The evaluation of this module expression was obtained as a pushout in the category of theories. In the case of more complicated module expressions[65] the evaluation is done by taking the colimit of the corresponding diagram in the category of theories.

[65] Possibly involving combinations between various kinds of module imports and instantiations of generic modules via views.

5.2 Satisfaction Condition for Category-based Equational Logic

In order to apply the theory of institutions to our framework, we have to answer the following questions:

1. What is a morphism of signatures in the case of category-based equational logicsΓ
2. What are the translations of models and sentences along signature morphisms, and, in particular, what is the translation of the quantifiers along signature morphismsΓ
3. Does the satisfaction relation between models and sentences in category-based equational logics given by Definition 3.6 verify the Satisfaction ConditionΓ

The answers to these questions would be helped by taking a closer look at the typical case of many sorted equational logic:

Definition 5.13 A signature morphism $\phi\colon (S,\Sigma) \to (S',\Sigma')$ in MSA is a pair $\langle f, g\rangle$ consisting of a map $f : S \to S'$ on sorts and an $S^* \times S$-indexed family of maps $g_{u,s} : \Sigma_{u,s} \to \Sigma'_{f^*(u),f(s)}$ on operator symbols. □

Example 5.14 ϕ of the previous definition determines a forgetful functor $\mathbb{A}lg(\phi) : \mathbb{A}lg_{\Sigma'} \to \mathbb{A}lg_{\Sigma}$ on models and another forgetful functor $\mathbb{S}et^f : \mathbb{S}et^{S'} \to \mathbb{S}et^S$ on domains. Notice the commutativity of the following diagram:

$$\begin{tikzcd}
\mathbb{A}lg_{\Sigma'} \arrow[r, "\mathcal{U}'"] \arrow[d, "\mathbb{A}lg(\phi)"'] & \mathbb{S}et^{S'} \arrow[d, "\mathbb{S}et^f"] \\
\mathbb{A}lg_{\Sigma} \arrow[r, "\mathcal{U}"] & \mathbb{S}et^{S}
\end{tikzcd}$$

where $\mathcal{U}$ and $\mathcal{U}'$ are the corresponding forgetful functors from many sorted algebras to many sorted sets. □

To resume, any signature morphism determines a pair of forgetful functors, one on models ($\mathbb{A}lg(\phi)$ in the previous example), and one on domains ($\mathbb{S}et^f$ in the previous example). Each of them has a left adjoint, meaning that any model has a free extension along a signature morphism (while free extensions along theory morphisms is problematic in many logical systems, most of them still support free extensions along signature morphisms; a typical example being first order logic). Finally, forgetting model structure first along a signature morphism and afterwards to domains is the same as forgetting to domains first and domain structure afterwards.

All these ideas are formalised by the following definition:

Definition 5.15 A **category-based equational signature** is a functor $\mathcal{U} : \mathbb{A} \to \mathbb{X}$. A **morphism of category-based equational signatures** is a couple $\langle \mathcal{M}, \mathcal{D}\rangle : \mathcal{U} \to \mathcal{U}'$ of functors such that $\mathcal{M};\mathcal{U} = \mathcal{U}';\mathcal{D}$ and $\mathcal{D}$ has a left adjoint.

$$\begin{tikzcd}
\mathbb{A}' \arrow[r, "\mathcal{U}'"] \arrow[d, "\mathcal{M}"'] & \mathbb{X}' \arrow[d, "\mathcal{D}"] \\
\mathbb{A} \arrow[r, "\mathcal{U}"] & \mathbb{X}
\end{tikzcd}$$

□

Notice that consequently to Definition 5.3, a morphism of category-based equational signatures is **liberal** iff $\mathcal{M}$ has a left adjoint.

The following array shows how some concepts from many sorted equational logic are reflected at the level category-based equational logics:

MSA	category − based equational logics
signature	functor
(S, Σ)	$\mathcal{U} : \mathbb{A} \to \mathbb{X}$
S	$\mathbb{X}$
Σ	$\mathbb{A}$
$\phi = \langle f, g \rangle$	$\langle \mathcal{M}, \mathcal{D} \rangle$
f	$\mathcal{D}$
g	$\mathcal{M}$
$\mathbb{S}et^f$	$\mathcal{D}$
$\mathbb{A}lg(\phi)$	$\mathcal{M}$
$\Sigma-$equation	$\mathcal{U}-$equation

5.2.1 Many-sorted institutions

This section introduces a class of institutions for which the signature morphisms can be regarded as morphisms of category-based equational signatures. In this was, these institutions admit an internalisation of category-based equational logic.

In any institution that has "sorted" signatures, the category of domains for a theory is in fact the category of models for the simple signature containing only the "sorts" of the signature of the theory. Assuming a certain degree of liberality of the respective institution, the forgetful functor from the category of the models of the theory to the category of the domains has a left-adjoint. The following definition makes the notion of sorted signature precise and is generic for all examples of Section 2.3:

Definition 5.16 A **many-sorted institution** is a tuple $\Im = (\mathbb{S}ign, \mathcal{S}ort, \textsc{Mod}, Sen, \models)$ such that

- $(\mathbb{S}ign, \textsc{Mod}, Sen, \models)$ is an institution,
- $\mathcal{S}ort : \mathbb{S}ign \to \mathbb{S}et$ is a functor that has a left-adjoint left-inverse $\mathcal{Q}$, and
- $\Im$ is liberal on signature morphisms.

A **domain** in $\Im$ is a model for a signature of the form $\mathcal{Q}(S)$ for S an arbitrary set. □

Now, we are in the situation to internalise the category-based equational logics in many-sorted institutions:

Proposition 5.17 Let $\Im = (\mathbb{S}ign, \mathcal{S}ort, \textsc{Mod}, Sen, \models)$ be a many sorted institution with ε the co-unit of the persistent adjunction $\mathcal{Q} \dashv \mathcal{S}ort : \mathbb{S}et \rightharpoonup \mathbb{S}ign$.

Any signature morphism $\Phi : \Sigma \to \Sigma'$ determines a liberal morphism of category-based equational signatures

$$\langle \textsc{Mod}(\Phi), \textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Phi)) \rangle : \mathcal{U}_\Sigma \to \mathcal{U}_{\Sigma'}$$

where $\mathcal{U}_\Sigma = \textsc{Mod}(\varepsilon_\Sigma)$ is the forgetful functor form the category $\textsc{Mod}(\Sigma)$ of Σ-models to the category of domains $\textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Sigma))$ for any signature Σ of $\Im$.

Proof: Any signature morphism $\Phi : \Sigma \to \Sigma'$ induces a translation of sorts $\mathcal{S}ort(\Phi) : \mathcal{S}ort\Sigma \to \mathcal{S}ort\Sigma'$ which determines a domain reduct functor $\textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Phi)) : \textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Sigma')) \to \textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Sigma))$ having a left adjoint $\mathcal{Q}(\mathcal{S}ort\Phi)^\$$ in the virtue of the liberality of the institution $\Im$ on signature morphisms. $\textsc{Mod}(\Phi)$ has a left adjoint by the liberality of Φ.

$$\begin{tikzcd}
\textsc{Mod}(\Sigma) \arrow[d, "\textsc{Mod}(\varepsilon_\Sigma)"'] & \textsc{Mod}(\Sigma') \arrow[l, "\textsc{Mod}(\Phi)"'] \arrow[d, "\textsc{Mod}(\varepsilon_{\Sigma'})"] \\
\textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Sigma)) \arrow[r, shift right, "\mathcal{Q}(\mathcal{S}ort\Phi)^\$"'] & \textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Sigma')) \arrow[l, shift right, "\textsc{Mod}(\mathcal{Q}(\mathcal{S}ort\Phi))"']
\end{tikzcd}$$

The diagram commutes on right adjoints because of the naturality of ε, i.e., $\varepsilon_\Sigma ; \Phi = \mathcal{Q}(\mathcal{S}ort\Phi) ; \varepsilon_{\Sigma'}$, and by the application of the model functor to this identity. $\Box$

The liberality condition of Definition 5.16 is avery mild condition in practice. Even instituions notorious for not being liberal, like first order logic, are still liberal on signature morphisms.

Corollary 5.18 The signature morphisms in MSA, OSA, HCL, ELM are morphisms of category-based equational signatures.

Proof: In all cases this holds by the liberality of the repsective insdtituion on signature morphisms. A special mention is necessary for ELM. In this institution the signature morphisms are MSA theory morphisms, and we use the liberality of the institution of MSA. $\Box$

5.2.2 Sentence translations along morphisms of category-based equational signatures

Before defining the translations of equations along morphisms of category-based equational signatures, we have another look at the example of many sorted equational logic:

Example 5.19 Each function $f : S \to S'$ translates any S-sorted set X into the S'-sorted set $X^\sim$ by taking the [pointwise] left Kan extension of f along X:

$$X^\sim_{s'} = \coprod_{f(s)=s'} X_s \quad \text{for any sort} \quad s' \in S'.$$

$$\begin{tikzcd}
S \arrow[r, "f"] \arrow[dr, "X"'] & S' \\
& \mathbb{S}et
\end{tikzcd}
\qquad
\begin{tikzcd}
S' \arrow[d, "X^\sim"] \\
\mathbb{S}et
\end{tikzcd}$$

Any MSA signature morphism $\phi = \langle f, g \rangle : (S, \Sigma) \to (S', \Sigma')$ defines an S-sorted map $\phi^*_X : T_\Sigma(X) \to T_{\Sigma'}(X^\sim)\restriction_\phi$:

$$\begin{tikzcd}
X \arrow[r, "X\eta"] \arrow[dr, "j"'] & \mathcal{U}(T_\Sigma(X)) \arrow[d, "j^\sharp\mathcal{U} = \phi^*_X"] \\
& \mathcal{U}(T_{\Sigma'}(X^\sim)\restriction_\phi)
\end{tikzcd}$$

First, note that $X \subseteq \mathcal{U}(T_{\Sigma'}(X^{\sim})\restriction_{\phi})$ because if $x \in X_s$ then $x \in X^{\sim}_{f(s)}$ and $X^{\sim}_{f(s)} \subseteq T_{\Sigma'}(X^{\sim})_{f(s)} = (T_{\Sigma'}(X^{\sim})\restriction_{\phi})_s$; let $j : X \to \mathcal{U}(T_{\Sigma'}(X^{\sim})\restriction_{\phi})$ denote this inclusion. Then we simply define $\phi^*_X = j^\sharp$, where $j^\sharp$ is the unique extension of j to a Σ-homomorphism $T_\Sigma(X) \to T_{\Sigma'}(X^{\sim})\restriction_{\phi}$. Any Σ-equation $(\forall X)\langle s, t\rangle$ is translated to the Σ'-equation $(\forall X^{\sim})\langle \phi^*_X(l), \phi^*_X(r)\rangle$. $\square$

Notice that, in the previous example, the term algebra $T_{\Sigma'}(X^{\sim})$ is exactly the free extension of $T_\Sigma(X)$ along ϕ. From this, we may conclude that:

Translations of quantifiers are free extensions along signature morphisms.

This generalisation also covers the case when quantifiers are not free models. The translation of equations along signature morphisms in MSA is a particular case of the following abstract definition:

Definition 5.20 Let $\langle \mathcal{M}, \mathcal{D}\rangle$ be a liberal morphism of category-based equational signatures $(\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$. Then the $\mathcal{U}$-equation $(\forall A)\langle s, t\rangle$ is translated to the $\mathcal{U}'$-equation $(\forall A^{\$\$})\langle s^*, t^*\rangle$,

$$\begin{tikzcd} k \arrow[r, "k\theta"] \arrow[d, "t"'] & k^{\$}\mathcal{D} \arrow[d, "t^*\mathcal{D}"] \\ A\mathcal{U} \arrow[r, "A\alpha\mathcal{U}"'] & A^{\$\$}\mathcal{M}\mathcal{U} = A^{\$\$}\mathcal{U}'\mathcal{D} \end{tikzcd}$$

where $\_^{\$}$ denotes the left adjoint to $\mathcal{D}$, $\_^{\$\$}$ denotes the left adjoint to $\mathcal{M}$, α and θ denote the units of the adjunctions determined by $\mathcal{M}$ and $\mathcal{D}$, and s^* and t^* denote the unique "extensions" of $l; A\alpha\mathcal{U}$ and $r; A\alpha\mathcal{U}$ to maps in $\mathbb{X}'$.

Similarly, a $\mathcal{U}$-query $(\exists A)\langle s, t\rangle$ is translated to the $\mathcal{U}'$-query $(\exists A^{\$\$})\langle s^*, t^*\rangle$. $\square$

Since translations of quantifiers along liberal signature morphisms are free expansions of models and coequaliser projectivity is a property of the quantifiers essential for the completeness of the deduction system, we need to investigate the preservation of coequaliser projectivity under free expansions of models. The following lemma[66] gives a sufficient condition for the preservation of coequaliser projectivity under free expansions:

Lemma 5.21 Let $\mathcal{N} : \mathbb{A} \to \mathbb{B}$ be a left adjoint to a coequaliser preserving functor $\mathcal{M} : \mathbb{B} \to \mathbb{A}$. Then $\mathcal{N}$ preserves coequaliser projective objects.

Proof: Consider $A \in |\mathbb{A}|$ a coequaliser projective object. We have to prove that $A\mathcal{N}$ is coequaliser projective in $\mathbb{B}$.

$$\begin{tikzcd} B'\mathcal{M} \arrow[r, "e\mathcal{M}"] & B\mathcal{N} \\ A \arrow[u, "h'"] \arrow[r, "A\eta"'] & A\mathcal{N}\mathcal{M} \arrow[u, "h\mathcal{M}"'] \arrow[ul, "h'^\sharp\mathcal{M}"'] \end{tikzcd} \qquad \begin{tikzcd} B' \arrow[r, "e"] & B \\ & A\mathcal{N} \arrow[u, "h"'] \arrow[ul, "h'^\sharp"] \end{tikzcd}$$

Let $e : B' \to B$ be a coequaliser $\mathbb{B}$, and take an arbitrary $h : A\mathcal{N} \to B$. Because $e\mathcal{M}$ is a coequaliser in $\mathbb{A}$ (by hypothesis), there exists $h' : A \to B'\mathcal{M}$ such that $h'; e\mathcal{M} = A\eta; h\mathcal{M}$,

[66] It is used only in Chapter 6 in the context of the category-based semantics for constraint logic programming.

where $A\eta : A \to A\mathcal{N}\mathcal{M}$ is the universal arrow from A to $\mathcal{M}$. Let $h'^\sharp : A\mathcal{N} \to B'$ be the unique map such that $A\eta; h'^\sharp\mathcal{M} = h'$. Then

$$\begin{aligned} A\eta; (h'^\sharp; e)\mathcal{M} &= A\eta; h'^\sharp\mathcal{M}; e\mathcal{M} \\ &= h'; e\mathcal{M} \\ &= A\eta; h\mathcal{M} \qquad \text{(by definition of } h') \end{aligned}$$

By the universal property of $A\eta$ we have that $h'^\sharp; e = h$. □

Kleisli translations In this paragraph we study the particular case when the sentences, either equations or queries, are quantified by "variables". This technically corresponds to the existence of "term" models, i.e., the existence of left adjoints to the forgetful functors from models to domains.

In this case, the translation described by Definition 5.20 could be characterised as a morphism (i.e., functor) of Kleisli categories satisfying a certain universal property. This result together with the Satisfaction Condition for category-based equational logics constitute the technical basis for the development of the category-based semantics of equational logic programming queries and their solutions in the context of modularisation in the style of Eqlog.

By using the same notations as in Definition 5.20, further assume that $\mathcal{U}$ and $\mathcal{U}'$ have left adjoints $\mathcal{F}$ and, $\mathcal{F}'$ respectively, with η and ε and, η' and ε' respectively, the units and the co-units of the respective adjunctions. Fix a domain $x \in |\mathbb{X}|$. We may assume that $(x\mathcal{F})^{\$\$} = (x^\$)\mathcal{F}'$ in the virtue of the general principle of composition of adjunctions.

Fact 5.22 The diagram of Definition 5.20 defining the translations of equations and queries reads as:

$$\begin{tikzcd} k \arrow[r, "k\theta"] \arrow[d, "t"'] & k^\$\mathcal{D} \arrow[d, "t^*\mathcal{D}"] \\ x\mathcal{F}\mathcal{U} \arrow[r, "x\mathcal{F}\alpha\mathcal{U}"'] & (x\mathcal{F})^{\$\$}\mathcal{M}\mathcal{U} = x^\$\mathcal{F}'\mathcal{U}'\mathcal{D} \end{tikzcd}$$

□

Lemma 5.23 There exists a unique natural transformation $\gamma : \mathcal{D}; \mathcal{F} \to \mathcal{F}'; \mathcal{M}$ such that $\eta'\mathcal{D} = \mathcal{D}\eta; \gamma\mathcal{U}$. Moreover, $\mathcal{M}\varepsilon = \mathcal{U}'\gamma; \varepsilon'\mathcal{M}$ and $\mathcal{F}\alpha = \theta\mathcal{F}; \_^\$\gamma$.

Proof: The natural transformation γ is uniquely defined by the formula $\eta'\mathcal{D} = \mathcal{D}\eta; \gamma\mathcal{U}$ by using the universal property of the unit η.

Now, by the triangular laws for adjunctions, we have $\mathcal{U}'\mathcal{D}\eta; \mathcal{M}\varepsilon\mathcal{U} = \mathcal{M}\mathcal{U}\eta; \mathcal{M}\varepsilon\mathcal{U} = 1_{\mathcal{M};\mathcal{U}}$, and by the previous formula and the triangular laws for adjunctions we have $\mathcal{U}'\mathcal{D}\eta; \mathcal{U}'\gamma\mathcal{U}; \varepsilon'\mathcal{M}\mathcal{U} = \mathcal{U}'\eta'\mathcal{D}; \varepsilon'\mathcal{U}'\mathcal{D} = 1_{\mathcal{U}';\mathcal{D}} = 1_{\mathcal{M};\mathcal{U}}$. Then $\mathcal{U}'\mathcal{D}\eta; \mathcal{M}\varepsilon\mathcal{U} =$
$\mathcal{U}'\mathcal{D}\eta; \mathcal{U}'\gamma\mathcal{U}; \varepsilon'\mathcal{M}\mathcal{U}$. By the universal property of the unit η, we deduce $\mathcal{M}\varepsilon = \mathcal{U}'\gamma; \varepsilon'\mathcal{M}$.

$$\begin{tikzcd} x \arrow[r, "x\theta"] \arrow[d, "x\eta"'] & x^\$\mathcal{D} \arrow[r, "x^\$\eta'\mathcal{D}"] \arrow[d, "x^\$\mathcal{D}\eta"] & x^\$\mathcal{F}'\mathcal{U}'\mathcal{D} \arrow[d, equal] \\ x\mathcal{F} \arrow[r, "x\theta\mathcal{F}\mathcal{U}"'] & x^\$\mathcal{D}\mathcal{F}\mathcal{U} \arrow[r, "x^\$\gamma\mathcal{U}"'] & x^\$\mathcal{F}'\mathcal{M}\mathcal{U} \end{tikzcd}$$

For the last identity, fix $x \in |\mathbb{X}|$. Then

$$
\begin{array}{rcll}
x\eta; x\theta\mathcal{F}\mathcal{U}; x^{\$}\gamma\mathcal{U} & = & x\theta; x^{\$}\mathcal{D}\eta; x^{\$}\gamma\mathcal{U} & \text{(by the naturality of } \eta) \\
 & = & x\theta; x^{\$}\eta'\mathcal{D} & \text{(by the Definition of } \gamma) \\
 & = & x\eta; x\mathcal{F}\alpha\mathcal{U} & \text{(as unit of the composite of adjunctions} \\
 & & & \text{in two different ways)}
\end{array}
$$

By the universal property of $x\eta$ we deduce that $x\theta\mathcal{F}; x^{\$}\gamma = x\mathcal{F}\alpha$. □

Corollary 5.24 When the category-based equational signatures have left adjoint, we can define the translation of sentences along morphisms of category-based equational signatures that are not necessarily liberal.

Proof: By replacing $\mathcal{F}\alpha$ from Fact 5.22 with $\theta\mathcal{F}; \_^{\$}\gamma$. □

In order to give the universal characterization of this translation as a morphism of Kleisli categories we have to resort to the (rather sophisticated) theory of monads in 2-categories developed by Street in [88]:

Definition 5.25 Let $\mathbb{C}$ be a 2-category.

A **monad** $\langle X, S\rangle$ consists of an object X, a 1-cell $X \xrightarrow{S} X$ and a pair of 2-cells $1 \xrightarrow{\eta} S$, $S; S \xrightarrow{\mu} S$ (called the **unit** and the **multiplication**) satisfying the commutative diagrams

$$
\begin{tikzcd}
S \arrow[r, "S\eta"] \arrow[dr, equal] & SS \arrow[d, "\mu"] & S \arrow[l, "\eta S"'] \arrow[dl, equal] \\
 & S &
\end{tikzcd}
\qquad
\begin{tikzcd}
SSS \arrow[r, "S\mu"] \arrow[d, "\mu S"'] & SS \arrow[d, "\mu"] \\
SS \arrow[r, "\mu"'] & S
\end{tikzcd}
$$

A **monad functor** $\langle U, \phi\rangle\colon \langle X, S\rangle \to \langle Y, T\rangle$ consists of a 1-cell $X \xrightarrow{U} Y$ and a 2-cell $U; T \xrightarrow{\phi} S; U$ satisfying the commutative diagrams

$$
\begin{tikzcd}
UT \arrow[dr, "\phi"] & \\
U \arrow[u, "U\eta"] \arrow[r, "\eta U"'] & SU
\end{tikzcd}
\qquad
\begin{tikzcd}
UTT \arrow[r, "\phi T"] \arrow[d, "U\mu"'] & SUT \arrow[r, "S\phi"] & SSU \arrow[d, "\mu U"] \\
UT \arrow[rr, "\phi"'] & & SU
\end{tikzcd}
$$

A **monad functor transformation** $\langle U, \phi\rangle \xrightarrow{\sigma} \langle U', \phi'\rangle$ is a 2-cell $U \xrightarrow{\sigma} U'$ satisfying the commutative diagram

$$
\begin{tikzcd}
UT \arrow[r, "\sigma T"] \arrow[d, "\phi"'] & U'T \arrow[d, "\phi'"] \\
SU \arrow[r, "S\sigma"'] & SU'
\end{tikzcd}
$$

The 2-category $\mathbb{M}nd(\mathbb{C})$ has monads as objects, monad functors as 1-cells, and monad functor transformations as 2-cells. □

Definition 5.26 For any 2-category $\mathbb{C}$, let $\mathbb{C}^*$ denote the 2-category obtained from $\mathbb{C}$ by reversing all 1-cells (so that $\mathbb{C}^*(x, y) = \mathbb{C}(y, x)$). $\mathbb{M}nd(\mathbb{C}^*)^*$ has the monads of $\mathbb{C}$ as objects, **monad opfunctors** of $\mathbb{C}$ as 1-cells and **monad opfunctor transformations** as 2-cells. □

Theorem 5.27 (from [88]) In a 2-category $\mathbb{C}$ suppose $\langle X, T\rangle$ and $\langle X', T'\rangle$ are monads. Any adjunction $H \dashv D\colon X \rightharpoonup X'$ sets up a natural bijection between the monad functors $\langle D, \gamma\rangle\colon \langle X', T'\rangle \to \langle X, T\rangle$ and the monad opfunctors $\langle H, \delta\rangle\colon \langle X, T\rangle \to \langle X', T'\rangle$. □

Also, any category-based equational signature canonically determines a monad. However, category-based equational signatures are more general than monads because some adjunctions fail to be monadic. As already mentioned, an important class of examples in this sense is given by the order sorted theories.

Definition 5.28 Category-based equational signatures form a 2-category $\mathbb{E}qSig$ such that

- objects are category-based equational signatures,
- 1-cells are morphisms of category-based equational signatures, and
- 2-cells $\langle \sigma, \tau\rangle\colon \langle \mathcal{M}, \mathcal{D}\rangle \to \langle \mathcal{M}', \mathcal{D}'\rangle$ are pairs of natural transformations $\sigma\colon \mathcal{M} \to \mathcal{M}'$, $\tau\colon \mathcal{D} \to \mathcal{D}'$ such that $\sigma\mathcal{U} = \mathcal{U}'\tau$.

□

Corollary 5.29 There exists a forgetful 2-functor $\mathcal{M}nd\colon \mathbb{E}qSig^* \to \mathbb{M}nd(\mathbb{C}at)$ which determines (by Theorem 5.27) a canonical 2-functor $\mathcal{M}ndop\colon \mathbb{E}qSig \to \mathbb{M}nd(\mathbb{C}at^*)^*$ mapping morphisms of category-based equational signatures to monad opfunctors.

Proof: $\mathcal{M}nd$ maps a category-based equational signature $\mathcal{U}\colon \mathbb{A} \to \mathbb{X}$ to its attached monad $\langle \mathbb{X}, \mathcal{T}\rangle$ of $\mathbb{C}at$, morphisms of equational logics $\langle \mathcal{M}, \mathcal{D}\rangle$ to monad functors $\langle \mathcal{D}, \gamma\mathcal{U}\rangle\colon \langle \mathbb{X}', \mathcal{T}'\rangle \to \langle \mathbb{X}, \mathcal{T}\rangle$ (γ defined by Lemma 5.23) and maps 2-cells $\langle \sigma, \tau\rangle$ to monad functor transformations τ. Straightforward calculations assure the correctness of these definitions.

$\mathcal{M}ndop$ maps morphisms of category-based equational signatures $\langle \mathcal{M}, \mathcal{D}\rangle$ to the monad opfunctors $\langle \_^{\$}, \delta\rangle\colon \langle \mathbb{X}, \mathcal{T}\rangle \to \langle \mathbb{X}', \mathcal{T}'\rangle$ corresponding to the monad functor $\langle \mathcal{D}, \gamma\mathcal{U}\rangle$, where $\_^{\$}$ is the left-adjoint to $\mathcal{D}$. □

Recall (from [64]) that any monad $\langle \mathbb{X}, \mathcal{T}\rangle$ in $\mathbb{C}at$ determines a **Kleisli category** $\mathbb{X}_{\mathcal{T}}$ having the same objects as $\mathbb{X}$ but "substitutions" as arrows, i.e.,

$$\mathbb{X}_{\mathcal{T}}(x, y) = \{h^\flat \mid h \in \mathbb{X}(x, y\mathcal{T})\}$$

The composition of arrows in $\mathbb{X}_{\mathcal{T}}$ is given by $h^\flat; h'^\flat = (h; h'\mathcal{T}; z\mu)^\flat$:

$$x \xrightarrow{h} y\mathcal{T} \xrightarrow{h'\mathcal{T}} z\mathcal{T}\mathcal{T} \xrightarrow{z\mu} z\mathcal{T}$$

When the monad is determined by a category-based signature $\mathcal{U}$, the Kleisli category $\mathbb{X}_{\mathcal{T}}$ is in fact the *substitution system* determined by $\mathcal{U}$. In this case, a simple calculation shows that the composition in $\mathbb{X}_{\mathcal{T}}$ corresponds exactly to the composition of substitutions:

$$\begin{tikzcd} x \arrow[r, "h"] & y\mathcal{F}\mathcal{U} \arrow[r, "h'^\sharp\mathcal{U}"] & z\mathcal{F}\mathcal{U} \\ & y \arrow[u, "y\eta"] \arrow[ur, "h'"'] & \end{tikzcd}$$

When there is no danger of confusion we identify $\mathbb{X}(x, y\mathcal{FU})$ with $\mathbb{X}_{\mathcal{T}}(x, y)$ via the bijection $\_^{\flat}$.

Following [88], for any 2-category $\mathbb{C}$, there is an "inclusion" 2-functor $\mathcal{I}nc_{\mathbb{C}} \colon \mathbb{C} \to \mathbb{M}nd(\mathbb{C})$ mapping each object X to the trivial monad $\langle X, 1\rangle$. The well-known construction of the Eilenberg-Moore algebras categories appears a right 2-adjoint to $\mathcal{I}nc_{\mathbb{C}at}$ [88]. The following definition is the basis in [88] for recovering the theory of monadicity in the abstract framework of an arbitrary 2-category $\mathbb{C}$:

Definition 5.30 The 2-category $\mathbb{C}$ **admits construction of algebras** iff $\mathcal{I}nc_{\mathbb{C}}$ has a right 2-adjoint. □

Theorem 5.31 (from [88]) $\mathbb{C}at^*$ admits construction of algebras. The left 2-adjoint to $\mathcal{I}nc^*_{\mathbb{C}at^*} \colon \mathbb{C}at \to \mathbb{M}nd(\mathbb{C}at^*)^*$ is the Kleisli construction, which evaluated at $\langle \mathbb{X}, \mathcal{T}\rangle$ is $\mathbb{X}_{\mathcal{T}}$ and the unit $\langle J_{\mathcal{T}}, \omega\rangle \colon \langle \mathbb{X}, \mathcal{T}\rangle \to \langle \mathbb{X}_{\mathcal{T}}, 1\rangle$ is given by

- $J_{\mathcal{T}} \colon \mathbb{X} \to \mathbb{X}_{\mathcal{T}}$ with $xJ_{\mathcal{T}} = x$ for any $x \in |\mathbb{X}|$, and $fJ_{\mathcal{T}} = (f; x'\eta)^{\flat}$ for any $f \in \mathbb{X}(x, x')$, and
- $\omega \colon \mathcal{T}; J_{\mathcal{T}} \to J_{\mathcal{T}}$ with $x\omega = (1_{x\mathcal{T}})^{\flat}$ for any $x \in |\mathbb{X}|$.

$$\begin{tikzcd}
\langle \mathbb{X}, \mathcal{T}\rangle \arrow[r, "\langle J_{\mathcal{T}}, \omega\rangle"] \arrow[dr, "\langle K, \delta\rangle"'] & \langle \mathbb{X}_{\mathcal{T}}, 1\rangle \arrow[d, "\langle \mathcal{K}, 1\rangle"] & \mathbb{X}_{\mathcal{T}} \arrow[d, "\mathcal{K}"] \\
& \langle \mathbb{Y}, 1\rangle & \mathbb{Y}
\end{tikzcd}$$

□

From Theorem 5.31 and Corollary 5.29 we deduce the main result of this paragraph:

Corollary 5.32 For any morphism of category-based equational signatures $\langle \mathcal{M}, \mathcal{D}\rangle \colon (\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ there exists a unique functor $\mathcal{K} \colon \mathbb{X}_{\mathcal{T}} \to \mathbb{X}'_{\mathcal{T}'}$ such that

- $J_{\mathcal{T}}; \mathcal{K} = \_^{\$}; J_{\mathcal{T}'}$, and
- $(1_{x\mathcal{T}})^{\flat}\mathcal{K} = (x\delta)^{\flat}$, where $\delta \colon \mathcal{T}; \_^{\$} \to \_^{\$}; J_{\mathcal{T}'}$ is the natural transformation part of $\mathcal{M}ndop\langle \mathcal{M}, \mathcal{D}\rangle$.

$$\begin{tikzcd}
\mathcal{M}ndop(\mathcal{U}) = \langle \mathbb{X}, \mathcal{T}\rangle \arrow[r, "\langle J_{\mathcal{T}}, \omega\rangle"] \arrow[d, "\mathcal{M}ndop\langle \mathcal{M}, \mathcal{D}\rangle = \langle _^{\$}, \delta\rangle"'] & \langle \mathbb{X}_{\mathcal{T}}, 1\rangle \arrow[d, "\langle \mathcal{K}, 1\rangle"] \\
\mathcal{M}ndop(\mathcal{U}') = \langle \mathbb{X}', \mathcal{T}'\rangle \arrow[r, "\langle J_{\mathcal{T}'}, \omega'\rangle"'] & \langle \mathbb{X}'_{\mathcal{T}'}, 1\rangle
\end{tikzcd}$$

□

By spelling out the two properties of $\mathcal{K}$ we get exactly the translation described by the version of Definition 5.20 presented at the beginning of this paragraph (see Fact 5.22).

5.2.3 The Satisfaction Condition

The following result can be regarded as a generic proof of the Satisfaction Condition for any equational logic. All examples in Section 2.3 generate [equational] institutions by following the same pattern. The equational version of this theorem can be extended to conditional equations without any problem.

Theorem 5.33 Let $\langle \mathcal{M}, \mathcal{D} \rangle$ be a liberal morphism of category-based equational signatures $(\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$. Then for any model $B \in |\mathbb{A}'|$ and for any sentence $(\lambda A)\langle s, t\rangle$, with $\lambda \in \{\forall, \exists\}$,

$$B \models_{\mathcal{U}'} (\lambda A^{\$\$})\langle s^*, t^*\rangle \quad \text{iff} \quad B\mathcal{M} \models_{\mathcal{U}} (\lambda A)\langle s, t\rangle$$

Proof: The right adjoint $\mathcal{M}$ determines a natural bijection $\mathbb{A}(A, B\mathcal{M}) \simeq \mathbb{A}'(A^{\$\$}, B)$ mapping each model morphism $A \xrightarrow{h} B\mathcal{M}$ to the model morphism $A^{\$\$} \xrightarrow{\bar{h}} B$ such that $h = A\alpha; \bar{h}\mathcal{M}$.

$$\begin{tikzcd} A \arrow[r, "A\alpha"] \arrow[dr, "h"'] & A^{\$\$}\mathcal{M} \arrow[d, "\bar{h}\mathcal{M}"] \\ & B\mathcal{M} \end{tikzcd}$$

For each $v : k \to A\mathcal{U}$, we have:

$$\begin{aligned} k\theta; (v^*; \bar{h}\mathcal{U}')\mathcal{D} &= k\theta; v^*\mathcal{D}; \bar{h}\mathcal{U}'\mathcal{D} \\ &= v; A\alpha\mathcal{U}; \bar{h}\mathcal{U}'\mathcal{D} \qquad \text{(by Definition 5.20)} \\ &= v; A\alpha\mathcal{U}; \bar{h}\mathcal{M}\mathcal{U} \\ &= v; h\mathcal{U} \end{aligned}$$

Therefore,

$$\begin{aligned} B \models_{\mathcal{U}'} (\forall A^{\$\$})\langle s^*, t^*\rangle \quad & \text{iff} \quad s^*; \bar{h}\mathcal{U}' = t^*; \bar{h}\mathcal{U}' && \text{for all } A^{\$\$} \xrightarrow{\bar{h}} B \\ & \text{iff} \quad k\theta; (s^*; \bar{h}\mathcal{U}')\mathcal{D} = k\theta; (t^*; \bar{h}\mathcal{U}')\mathcal{D} \\ & \text{iff} \quad s; h\mathcal{U} = t; h\mathcal{U} && \text{for all } A \xrightarrow{h} B\mathcal{M} \\ & \text{iff} \quad B\mathcal{M} \models_{\mathcal{U}} (\forall A)\langle s, t\rangle \end{aligned}$$

A similar argument works for the case of queries. □

In the case when the sentences are quantified by variables, rather than models, we have the following corollary:

Corollary 5.34 Let $\langle \mathcal{M}, \mathcal{D} \rangle$ be a morphism of category-based equational signatures $(\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ such that $\mathcal{F}$ and $\mathcal{F}'$ are left adjoints to $\mathcal{U}$ and $\mathcal{U}'$, respectively. Then for any model $B \in |\mathbb{A}'|$ and for any sentence $(\lambda x)\langle s, t\rangle$, with $\lambda \in \{\forall, \exists\}$ and x a domain in $\mathbb{X}$,

$$B \models_{\mathcal{U}'} (\lambda x^{\$})\langle s^*, t^*\rangle \quad \text{iff} \quad B\mathcal{M} \models_{\mathcal{U}} (\lambda x)\langle s, t\rangle$$

Proof: By using the last equation Lemma 5.23, the existence of a left adjoint ot $\mathcal{M}$ is no longer necessary. □

The fact that $\mathbb{E}qSig$ comes naturally equipped with a 2-categorical structure reinforces the argument of Goguen and Burstall [32] that the signatures of any chartable institution form a 2-category. The presentation of the sentence functor as a Kleisli translation projects a new light on the duality between syntax and semantics in category based equational logic: the sentence functor is a model functor when reversing the 1-cells in $\mathbb{C}at$!

5.3 Queries and Solutions *versus* Modularisation

In this section we give a categorical semantics for equational logic programming queries and their solutions in the context of modularisation in the style of the programming language Eqlog, and we discuss the crucial problem of the soundness and completeness for module imports. We take here the point of view of [21] that modules are presentations (theories) and that module imports are morphisms of presentations (theories). In [39], Goguen and Meseguer give a denotational semantics for equational logic programming based on initial algebra semantics. Due to the presence of logical variables, the denotation of an equational logic programming module is given by an adjunction rather than an initial model. This is in fact the adjunction determined by the forgetful functor from the category of models of the given module to the category of domains representing the mathematical structure for the collections of logical variables. This idea exploits the fact that the notion of category-based equational signature is abstract enough to contain the concept of equational logic programming module in the manner described in Section 2.3.4. The principle underlying our category-based semantics for equational logic programming queries and their solutions is formulated as

> The denotation of modules is abstracted to category-based equational signatures that have left adjoints.

Definition 5.35 Let P be an equational logic programming module. Its **denotation** $[\![P]\!]$ is the forgetful functor $[\![P]\!]\colon \textsc{Mod}(P) \to \textsc{Dom}(P)$ from the category of its models, $\textsc{Mod}(P)$, to the category of its domains, $\textsc{Dom}(P)$.

The denotation of a module import $P \xrightarrow{\psi} P'$ is a morphism of category-based equational signatures $[\![\psi]\!]\colon [\![P]\!] \to [\![P']\!]$. □

Definition 5.36 A **query** for the equational logic programming module P is a $[\![P]\!]$-query. A **solution** for a query $q = (\exists B)\langle t_1, t_2\rangle$ in a P-model A is a morphism $h\colon B \to A$ such that $t_1; h[\![P]\!] = t_2; h[\![P]\!]$.

Let $P \xrightarrow{\psi} P'$ be a module import. The translation of queries along ψ (i.e., from P-queries to P'-queries) is given by the translation along the morphism of category-based equational signatures $[\![\psi]\!]$ accordingly to Definition 5.20.[67] □

The interpretation of the Satisfaction Condition (Theorem 5.33) in this context is that for any P-query q, any module import $\psi\colon P \to P'$, and any P'-model A, there is a canonical one-one correspondence between the solutions of $q\psi$ in A and the solutions of q in $A\mathcal{M}$, where $\mathcal{M}$ is the model reduct component of $[\![\psi]\!]$.

[67] We denote this translation by $\_\psi$.

5.3.1 The institution of queries and substitutions

Computations in equational logic programming systems produce answers to queries in form of substitutions. As known, solutions for queries can be regarded as unifiers. The next fact is consistent to Goguen's approach on unifiers as co-cones in Kleisli categories as expressed in [27]:

Fact 5.37 Let $q = (\exists X)\langle t_1, t_2\rangle$ be a query for the program P whose quantification is given by variables, i.e., $X \in |\textsc{Dom}(P)|$. A **solution form** for q is a co-cone for the parallel pair $\langle t_1^\flat, t_2^\flat\rangle$ in $\textsc{Dom}(P)_{\mathcal{T}_P}$, where $\mathcal{T}_P$ is the monad determined by the [right-adjoint] forgetful functor $[\![P]\!]\colon \textsc{Mod}(P) \to \textsc{Dom}(P)$. □

The relationship between queries and substitutions can be formalised as a Satisfaction Relation in a particular institution in which queries play the rôle of models and substitutions play the rôle of sentences. The source of a certain substitution has to match the quantifier of a certain query in the same way the sentences and models of logical systems have to be based within the same language (i.e., signature). This suggests that the signatures for the institution of queries as models and of substitutions as sentences should be given by collections of [logical] variables.

Definition 5.38 Assume a fixed module P. We define an institution $\Im_P$ consisting of the following data:

- $\mathbb{S}ign = \textsc{Dom}(P)^{op}_{\mathcal{T}_P}$,[68] i.e., signatures are domains and signature morphisms are substitutions,
- $\textsc{Mod}(X) = \{(\exists X)\langle t_1, t_2\rangle \mid t_1, t_2 \text{ in } \mathcal{T}_P(X)\}$ for each domain X in $\textsc{Dom}(P)$, where $\mathcal{T}_P$ is the monad determined by the right adjoint forgetful functor $[\![P]\!]$. Each map $f^\flat$ in $\textsc{Dom}(P)^{op}_{\mathcal{T}_P}(X, X') = \textsc{Dom}(P)_{\mathcal{T}_P}(X', X)$ determines a reduct functor $\textsc{Mod}(f)\colon \textsc{Mod}(X') \to \textsc{Mod}(X)$ such that

  $$\textsc{Mod}(f)(q') = q'; f^\sharp$$

  for any query q' in $\textsc{Mod}(X')$,[69]
- $Sen(X) = \{\langle \psi, s\rangle \mid P \xrightarrow{\psi} P', s \text{ is a } P'\text{-substitution of the logical variables } X\psi\}$. Each map $f^\flat$ in $\textsc{Dom}(P)^{op}_{\mathcal{T}_P}(X, X')$ determines a sentence translation $Sen(f)\colon Sen(X) \to Sen(X')$ by

  $$Sen(f)\langle \psi, s\rangle = \langle \psi, f\psi; s^\sharp\rangle$$

  for any P'-substitution s and any module import ψ, and
- $q \models_X \langle \psi, s\rangle$ iff s is a solution form for the query $q\psi$.

□

[68] The opposite of the Kleisli category $\textsc{Dom}(P)_{\mathcal{T}_P}$.

[69] This translation corresponds to a translation of the logical variables of a query. This might also include identifications of variables.

Proposition 5.39 Given any module P, the previous construction $\Im_P$ defines an institution.

Proof: All we have to prove is the Satisfaction Condition for the institution $\Im_P$. Consider a domain map $X' \xrightarrow{f} \mathcal{T}_P(X)$, an arbitrary P-query $q' = (\exists X')\langle t_1, t_2\rangle$, and an arbitrary sentence $\langle \psi, s\rangle \in Sen(X)$. Then

$$\begin{array}{lll} q' \models_{X'} \langle \psi, f\psi; s^\sharp\rangle & \text{iff} & t_1\psi; (f\psi; s^\sharp)^\sharp = t_2\psi; (f\psi; s^\sharp)^\sharp \quad \text{(by Definition 5.38)} \\ & \text{iff} & t_1\psi; f^\sharp\psi; s^\sharp = t_2\psi; f^\sharp\psi; s^\sharp \\ & \text{iff} & (t_1; f^\sharp)\psi; s^\sharp = (t_2; f^\sharp)\psi; s^\sharp \quad \text{(by Corollary 5.32)} \\ & \text{iff} & q'; f^\sharp \models_X \langle \psi, s\rangle \quad \text{(by Definition 5.38)} \end{array}$$

□

5.3.2 Soundness and completeness for module imports

Definition 5.40 Let $\psi : P \to P'$ be a module import. ψ is **sound** iff for any P-query q and any solution form s for q, $s\psi$ is a solution form for $q\psi$.

ψ is **complete** iff for any P-query q and any solution form s' for $q\psi$ there exists a solution form s for q such that $s' = s\psi$. □

A sound and complete module import $P \to P'$ *protects* the solution forms, i.e., any P-query has the same solutions in P' as in P.

Fact 5.41 The composition of sound/complete module imports is sound/complete. □

There is a strong flavour of conceptual similarity between the soundness and completeness for module imports and the soundness and completeness for logical systems. In fact, both of them are instantiations of the category-based formulation of the concepts of soundness and at the level of institutions, as shown by the following result:

Proposition 5.42 In the institution $\Im_P$ introduced by Definition 5.38, consider the entailment relation $\vdash_X$ (parameterised by signatures, i.e., P-domains)[70] defined by the following inference rule encoding the translation of solution forms along imports of P:

$$P \xrightarrow{\psi} P' : \quad \frac{\langle 1_P, s\rangle}{\langle \psi, s\psi\rangle}$$

Consider an arbitrary P-query $q = (\exists X)\langle t_1, t_2\rangle$. Let q^* denote the set of all consequences of q of the form $\langle 1_P, s\rangle$, i.e., the set of all solution forms for q. Then

1. $q^* \vdash_X \langle \psi, s\rangle$ implies $q \models_X \langle \psi, s\rangle$ for all s iff ψ is sound, and
2. $q \models_X \langle \psi, s\rangle$ implies $q^* \vdash_X \langle \psi, s\rangle$ for all s iff ψ is complete.

Proof: The correctness of the definition of the entailment relation can be easily verified by checking all conditions from the definition of an entailment system (see [21] or [72]).

The proof of this proposition is essentially based on the observation that $q^* \vdash_X \langle \psi, s\rangle$ means that there exists s_0 a P-substitution that is a solution form for q and such that $s = s_0\psi$. The rest is given by Definition 5.40. □

[70] See [21, 72] for the definition of entailment relations in institutions.

Definition 5.43 A morphism of category-based equational signatures $\langle \mathcal{M}, \mathcal{D} \rangle \colon (\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ is **essentially persistent** iff it is liberal and the adjunctions corresponding to both $\mathcal{M}$ and $\mathcal{D}$ are persistent.

A module import ψ is essentially persistent iff its denotation $[\![\psi]\!]$ is an essentially persistent morphism of category-based equational signatures. □

When domains are many sorted sets, the persistency of the adjunction on domains corresponds exactly to the injectivity on sorts of the module import; this relates to Goguen-Meseguer use of persistency in the context of *protecting extensions* for built-ins in Eqlog [39].

Lemma 5.44 Let $\langle \mathcal{M}, \mathcal{D} \rangle \colon (\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ be an essentially persistent morphism of category-based equational signatures. Consider q a $\mathcal{U}$-query. Then:

- $\_^{\$}$ embeds $\mathbb{X}$ as a full subcategory of $\mathbb{X}'$, and
- $\hat{q}$ has exactly the same solution forms in $\mathbb{X}$ as q,

where $\hat{q}$ denotes the $\mathcal{U}'$-query obtained by translating q along $\langle \mathcal{M}, \mathcal{D} \rangle$.

Proof: For any query q and model A denote its solutions in the model A by $Sol(q, A)$. The image of $\_^{\$}$ in $\mathbb{X}'$ is a full subcategory as a consequence of the persistency of the adjunction determined by $\mathcal{D}$. Since $\_^{\$}$ is also injective on objects, it embeds $\mathbb{X}$ as a full subcategory of $\mathbb{X}'$. For the rest of the proof we identify $\mathbb{X}$ with the image of $\_^{\$}$.

Let $\mathcal{F}$ and $\mathcal{F}'$ be left adjoints to $\mathcal{U}$ and $\mathcal{U}'$, respectively. For any $y \in |\mathbb{X}|$, we have:

$$\begin{array}{rcll} Sol(q, y\mathcal{F}) & = & Sol(q, (y\mathcal{F})^{\$\$}\mathcal{M}) & \text{(persistency)} \\ & = & Sol(\hat{q}, (y\mathcal{F})^{\$\$}) & \text{(Theorem 5.33, Satisfaction Condition for queries)} \\ & = & Sol(\hat{q}, y\mathcal{F}') & \text{(composition of adjoints)} \end{array}$$

The conclusion of the lemma follows now by applying Corollary 5.34. □

Theorem 5.45 <u>Completeness</u> Let $P \xrightarrow{\psi} P'$ be a module import. Then

1. ψ is sound, and
2. ψ is complete whenever it is essentially persistent.

Proof: Let q be a query in P.

1. Assume s is a solution form for q. Then $s^\flat$ coequalises $q^\flat$, where $q^\flat$ is the parallel pair of arrows in the Kleisli category $\textsc{Dom}(P)_{\mathcal{T}_P}$ corresponding to the P-query q, and $s^\flat$ is the arrow in $\textsc{Dom}(P)_{\mathcal{T}_P}$ corresponding to the substitution s.

By Corollary 5.32, $[\![\psi]\!]$ determines a functor $\mathcal{K} \colon \textsc{Dom}(P)_{\mathcal{T}_P} \to \textsc{Dom}(P')_{\mathcal{T}_{P'}}$. This means that $(s\psi)^\flat = s^\flat\mathcal{K}$ coequalises $(q\psi)^\flat = q^\flat\mathcal{K}$, which means that $s\psi$ is an solution form for $q\psi$.

2. By applying the previous lemma to the case of the essentially persistent morphism of category-based equational signatures $[\![\psi]\!] \colon [\![P]\!] \to [\![P']\!]$. □

Example 5.46 Consider the generic module `LIST*` from Example 5.10. Notice that `MON` $\hookrightarrow$ `LIST*` is an essentially persistent module import. The query

```
select LIST* .
find X Y : Mon such that X * Y = Y * X .
```

has exactly the same solution forms in MON as in LIST*. □

The lack of persistency might destroy the completeness of module imports as in the following:

Example 5.47 Consider the following theories:

```
th SOURCE is
  sorts S1 S2 .
  op a : -> S1 .
  op b : -> S2 .
  op f : S1 -> S2 .
endth

th TARGET is
  sort S .
  op a´ : -> S .
  op b´ : -> S .
  op f : S -> S .

  eq f(b´) = b´ .
endth
```

and the following view:

```
view V from SOURCE to TARGET is
  sort S1 to S . sort S2 to S .
  op a to a´ .
  op b to b´ .
  op f to f .
endv
```

The TARGET-query

```
find X : S such that f(X) = b´ .
```

has a solution form (i.e., X:S->b´) although the SOURCE-query

```
select SOURCE .
find X : S1 such that f(X) = b .
```

does not have any solution form. □

5.4 Theorem of Constants

Theorem of Constants supports the treatment of universally quantified variables as temporary constants [30]. Although such treatments are used on a large scale in the context of term rewriting, the importance of a mathematical result providing foundations to equational theorem proving using ground rewriting was emphasised for the first time in the context of the OBJ system [30]. A similar application appeared in Chapter 4 (see Corollary 4.27) when dealing with the lifting of the completeness of paramodulation from the case of ground terms to the case of terms with variables.

The classical formulation of the Theorem of Constants establishes an equivalence between $(\forall X)\langle s, t\rangle$ being a consequence of a theory Γ in a signature Σ, and $(\forall\emptyset)\langle s, t\rangle$ being a consequence of Γ in the larger signature Σ_X which is obtained by adjoining the variables X to Σ as new constants.

5.4.1 The level of institutions

The Theorem of Constants admits a category-based version at the level of the theory of institutions which captures the essence of the model theoretic phenomenon underlying it. This is based on the internalisation of the notion of universal sentence (i.e., universal quantified formula) in any institution by following a category-based formulation of universal quantification.[71]

Definition 5.48 Let $\Im = (\mathbb{S}ign, \textsc{Mod}, Sen, \models)$ be any institution. $(\forall\iota)\rho$ is a $\Im$-**universal** Σ-sentence if

- $\Sigma \xrightarrow{\iota} \Sigma'$ is any signature morphism, and
- ρ is a Σ'-sentence.

A Σ-model M **satisfies** $(\forall\iota)\rho$ iff all its expansions to a Σ'-model satisfy ρ, i.e., $M' \models_{\Sigma'} \rho$ for all M' with $M'\restriction_\iota = M$. $\square$

The main idea here is that the symbols from Σ' that are not in Σ play the rôle of the variables. The previous definition includes also the case of second order universal quantification corresponding to the situation when some symbols from $\Sigma' - \Sigma$ are function or relation symbols. The classical situation of first order universal quantification occurs when all symbols from $\Sigma' - \Sigma$ are constants.

The Theorem of Constants admits the following generic institutional version:

Theorem 5.49 For any set Γ of Σ-sentences,

$$\Gamma \models_\Sigma (\forall\iota)\rho \quad \text{iff} \quad \iota(\Gamma) \models_{\Sigma'} \rho$$

Proof:

$$\begin{array}{lll} \textsc{Mod}(\Gamma) \models_\Sigma (\forall\iota)\rho & \text{iff} & \{N \mid N\restriction_\iota \in |\textsc{Mod}(\Gamma)|\} \models_{\Sigma'} \rho \\ & \text{iff} & \textsc{Mod}(\iota\Gamma) \models_{\Sigma'} \rho \qquad \text{(by the Sat. Cond. in } \Im\text{).} \end{array}$$

$\square$

[71] This was first introduced by Barwise and later used by Tarlecki in the context of "abstract algebraic institutions" [91].

Apart of applications to second order logic and category-based equational logic (next section), this abstract version of Theorem of Constants can be applied to hidden sorted logics, thus giving support to proofs for the object paradigm [35] based on ground rewriting.

5.4.2 The level of category-based equational logic

The previous generic version of the Theorem of Constants can be instantiated to the institution of category-based equational logics. When $(\forall A)\langle s, t\rangle$ is an equation having a model as a quantifier, the expanded signature Σ_A is obtained by adjoining the whole model A to Σ. This is reminiscent of the so-called *method of diagrams* in classical model theory [14], and is naturally encoded at the categorical level by using comma categories [89, 91].

If $\mathbb{A}$ is a category of models and A is any model, a morphism $A \to B$ interprets the elements of A as new constants in B. The evaluation of the model operations on these constants respects the model structure of A. The inclusion of signatures $\Sigma \hookrightarrow \Sigma_A$ is defined at the level of category-based equational signatures as follows:

Lemma 5.50 Let $\mathcal{U}\colon \mathbb{A} \to \mathbb{X}$ be a category-based equational signature such that $\mathbb{A}$ has binary coproducts. For any model A in $\mathbb{A}$, $\langle \mathbb{A}_A, 1_{\mathbb{X}}\rangle : \mathcal{U} \to \mathcal{U}_A = \mathbb{A}_A;\mathcal{U}$ is a liberal morphism of category-based equational signatures.

$$\begin{tikzcd}
\mathbb{A} \arrow[d, "\mathcal{U}"'] & (A{\downarrow}\mathbb{A}) \arrow[l, "\mathbb{A}_A"'] \arrow[d, "\mathcal{U}_A"] \\
\mathbb{X} \arrow[r, equal] & \mathbb{X}
\end{tikzcd}$$

Proof: All we have to prove is that $\mathbb{A}_A$ has a left adjoint. This is in fact $(A\coprod\_)\colon \mathbb{A} \to (A{\downarrow}\mathbb{A})$ mapping any model B to $A \xrightarrow{j_A} A\coprod B$ (j are the co-cone arrows of the coproduct of B and A). The unit of this adjunction at B is j_B. $\square$

The following corollary shows that the translation of sentences along the "inclusion" $\mathcal{U} \to \mathcal{U}_A$ corresponds in fact simply to the addition of the quantifiers to the signature. For this reason, and in the spirit of the tradition, we regard any $\mathcal{U}$-equation as a $\mathcal{U}_A$-equation without any further new notations.

Corollary 5.51 Let $\mathcal{U} : \mathbb{A} \to \mathbb{X}$ be a category-based equational signature such that the category of models $\mathbb{A}$ has binary coproducts and let A be any model in $\mathbb{A}$. Then the translation of a $\mathcal{U}$-equation $(\forall B)\langle s, t\rangle$ along $\langle \mathbb{A}_A, 1_{\mathbb{X}}\rangle$ is the $\mathcal{U}_A$-equation $(\forall j_A)\langle s; j_B\mathcal{U}, t; j_B\mathcal{U}\rangle$, where j are the co-cone arrows of the coproduct of B and A.

$$\begin{tikzcd}
k \arrow[d, "t"'] & & k \arrow[d, "t"] \\
B\mathcal{U} & \mapsto & B\mathcal{U} \arrow[d, "j_B\mathcal{U}"] \\
 & & (B\coprod A)\mathcal{U} = (A \xrightarrow{j_A} B\coprod A)\mathcal{U}_A
\end{tikzcd}$$

$\square$

Corollary 5.52 <u>Theorem of Constants</u> Let Γ be a collection of conditional $\mathcal{U}$-equations. Then,

$$\Gamma \models_{\mathcal{U}} (\forall A)\langle s,t\rangle \text{ iff } \Gamma \models_{\mathcal{U}_A} (\forall 1_A)\langle s,t\rangle$$

Proof: We have to show only that the satisfaction relation between models and universal sentences defined internally (Definition 5.48 applied to the morphism of category-based equational signatures, $\langle \mathbb{A}_A, 1_{\mathbb{X}}\rangle$ is the same as the satisfaction relation between models and abstract equations from Definition 3.6, i.e., for any $k \xrightarrow{\langle s,t\rangle} A\mathcal{U}$ and any model $M \in |\mathbb{A}|$,

$$M \models_{\mathcal{U}} (\forall\langle \mathbb{A}_A, 1_{\mathbb{X}}\rangle)\rho \text{ iff } M \models_{\mathcal{U}} (\forall A)\langle s,t\rangle, \quad \text{where } \rho = (\forall 1_A)\langle s,t\rangle$$

This reduces to show that $h \models_{\mathcal{U}_A} (\forall 1_A)\langle s,t\rangle$ for all $A \xrightarrow{h} M \in |(A{\downarrow}\mathbb{A})|$ iff $M \models_{\mathcal{U}} (\forall A)\langle s,t\rangle$. This holds since for any $A \xrightarrow{h} M$, $h \models_{\mathcal{U}_A} (\forall 1_A)\langle s,t\rangle$ iff $s;h\mathcal{U} = t;h\mathcal{U}$. $\square$

Note that 1_A is the initial object of $(A{\downarrow}\mathbb{A})$. In the traditional MSA version of the Theorem of Constants, the interpretation of the variables as new temporary constants empties the quantifier. In a more model-theoretic setup this would correspond to a quantification by a model of ground terms, categorically characterised by their initiality property.

6 Extensible Constraint Logic Programming

Constraint programming has been recently emerging as a powerful programming paradigm and it has attracted much research interest over the past decade. Mathematical Programming, Symbolic Computation, Artificial Intelligence, Program Verification and Computational Geometry are examples of application areas for constraint programming. In general, constraint logic programming replaces unification with constraint solving over computational domains. Constraint solving techniques have been incorporated in many programming systems; CLP [58], PrologIII [15], and Mathematica are the best known examples. The computational domains include linear arithmetic, boolean algebra, lists, finite sets. Conventional logic programming (i.e., Prolog) can be regarded as constraint programming over term models (i.e., Herbrand universes). In general, the actual constraint programming systems allow constraint solving for a fixed collection of data types or computational domains.[72] As already mentioned in the Introduction, constraint programming allowing constraints over *any* data type will be called **extensible**.

In [58], Jaffar and Lassez propose a scheme for constraint logic programming based on embedding constraint systems into Horn clause logic by axiomatising computational domains by Horn clauses. In [87], Smolka propose a completely different framework for constraint logic programming by regarding programs as collections of definitions of new constraints extending the underlying constraint system.

This chapter deals only with the model theoretic semantics of constraint logic programming, we don't address any issues directly related to the computation level of constraint solving. Our approach to constraint programming departs fundamentally from the previous ones; our semantics for extensible constraint logic programming follows the principles underlying the model theory for constraint logic programming proposed by Goguen and Meseguer in the context of the language Eqlog [39] and is essentially based on a version of Herbrand's Theorem for *constraint logic*, i.e., the logic underlying extensible constraint logic programming. Similarly to the approach proposed by Jaffar and Lassez, both constraint relations and programs are [collections of] sentences within the same logical system (in the sense of institutions rather than of deduction systems). However, constraint logics are much more general than Horn clause logic. In fact, the computation domain is a primitive in our approach and plays a central rôle in the definition of constraint logic, rather than being axiomatised in an already defined logic (i.e., Horn clause logic) like in CLP. When regarded as a model in constraint logic, the computation domain appears as the *initial* model. This is mathematically linked to the semantics of OBJ-like module systems, the fundamental idea being to regard the models of extensible constraint logic programming as expansions of an appropriate *built-in model* A along a signature inclusion $\iota : \Sigma \hookrightarrow \Sigma'$, where Σ is the signature of built-in sorts, operations and relation symbols, and Σ' adds new "logical" symbols. In practice, the constraint relations (i.e., logical relations that one wishes to impose on a set of potential solutions) are limited to sets of atomic sentences involving both Σ-symbols and elements of the built-in model A. However, at the theory level there is no reason to restrict the shape of constraint relations only to atomic formulae. The models for constraint logic programming

[72] A computational domain can be regarded as a model (not necessarily the standard one) for a certain data type specification.

appear as expansions of the built-in model to the larger signature Σ' and any morphism of constraint models has to preserve the built-ins. Therefore, the constraint models form a category which can be formally defined as the comma category $(A{\downarrow}\textsc{Mod}(\iota))$.

Example 6.1 Consider the example of a specification of the Euclidean plane as a vector space over the real numbers.

```
obj R2 is
  pr FLOAT * (sort Float to Real) .
  sort Vect .

  op 0 : -> Vect .
  op <_,_> : Real Real -> Vect .
  op _+_ : Vect Vect -> Vect .
  op -_ : Vect -> Vect .
  op _*_ : Real Vect -> Vect .

  vars a b a´ b´ k : Real .
  eq 0 = < 0 , 0 > .
  eq < a , b > + < a´ , b´ > = < a + a´ , b + b´ > .
  eq k * < a , b > = < k * a , k * b > .
  eq - < a , b > = < - a , - b > .
endo
```

The signature Σ of built-in sorts, operation and relation symbols contains one sort `Real` for the real numbers together with the usual ring operation symbols and a relation symbol `<`. The built-in model is just the usual ring of real numbers (denoted as $\mathbb{R}$) with `<` interpreted as the usual 'strictly less than' predicate. The signature Σ' of the module `R2` introduces a new operation symbol `< , >` for representing the points of the Euclidean plane as tuples of real numbers, and overloads the ring operations by organising the Euclidean plane as a vector space over the real numbers. The axioms express the basic fact that the evaluation of the ring operations on vectors is done component-wise.

A standard model for this specification, denoted by $\mathbb{R}^2$ is given by the cartesian representation of the points of the Euclidean plane, i.e., any point is represented as the tuple of its coordinates. Another model for `R2` interprets the sort `Vect` as the set of real numbers, the ring operations on `Vect` as ordinary operations on numbers, and `< , >` as addition of numbers. Let's denote this model by $\mathbb{R}+$. $\square$

6.1 Generalised Polynomials and Constraint Satisfaction

It is important to have a formal definition for constraint sentences, constraint models, and a satisfaction relation between them. This would define a logic underlying constraint logic programming; we call this **constraint logic**. A fundamental principle in this logic is the preservation of the built-ins.

Consistently to our previous notations, let $A^{\$\$}$ denote the free expansion of the built-in model A along the inclusion [of the signature of the built-ins] $\iota\colon \Sigma \to \Sigma'$. Also, let $\mathcal{F}'$ be a left adjoint to the forgetful functor $\mathcal{U}'\colon \textsc{Mod}(\Sigma') \to \textsc{Dom}(\Sigma')$.[73] The rôle played

[73] From the category of the models of the signature Σ' to the category of the domains of Σ'.

by the terms in ordinary logic is played by *generalised polynomials*[74] in constraint logic. Generalised polynomials are term-like structures involving both operator symbols and elements of the built-in model. Generalised polynomials can be regarded as elements of models in the same way as ordinary terms are regarded as elements of [free] models as a basis for a semantical aproach to the concept of sentence and satisfaction in equational logic.

Given a domain x (i.e., a collection of variables in practice), the Σ'-model of the polynomials over x is usualy denoted as $A[x]$. This is in fact the coproduct $A^{\$\$} \coprod x\mathcal{F}'$ between $A^{\$\$}$ and the free Σ'-model $x\mathcal{F}'$. When $\Sigma = \Sigma'$ are unsorted algebraic signatures, this is a well known construction in universal algebra [48]. However, the best known example still comes from linear algebra:

Example 6.2 Let $\underline{X}$ be a set of variables. $\mathbb{R}[\underline{X}]$ is the ring of polynomials over $\underline{X}$ and with real numbers as coefficients. In this example, the signature Σ of built-ins is a ring signature, and Σ' doesn't add any new symbols, thus $\Sigma = \Sigma'$. The model of the built-ins is $\mathbb{R}$, the usual ring of real numbers. $\square$

The universal property of the models of generalised polynomials allow a more general definition that extends the concept of generalised polynomial to the semantic case when models play the rôle of the collections of variables and model morphisms paly the rôle of the valuation maps.

Definition 6.3 Let B be any Σ'-model. The **model of generalised polynomials over** B is the coproduct $A^{\$\$} \coprod B$, and it is denoted as $A[B]$. $\square$

6.1.1 Internal constraint logic

Constraint logic can be defined internally to category-based equational logic. This means that the signature of buit-ins Σ is abstracted to a category-based equational signature $\mathcal{U}$, Σ' to $\mathcal{U}'$, and the inclusion $\iota\colon \Sigma \to \Sigma'$ to a morphism of category-based equational signatures $\mathcal{U} \to \mathcal{U}'$. In this way, the extensible constraint programming paradigm is accomodated by any logical system that is a category-based equational logic.

Definition 6.4 Let $\langle \mathcal{M}, \mathcal{D} \rangle\colon (\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ be any liberal morphism of category-based equational signatures. Fix any model $A \in |\mathbb{A}|$ (playing the rôle of the model of the **built-ins**).

A **constraint model** is a model in $\mathbb{A}'$ whose reduct to the signature of built-ins contains an image of A, i.e., a map $c\colon A \to C\mathcal{M}$ with $C \in |\mathbb{A}'|$. A model morphism $h\colon c \to c'$ is a map $C \to C'$ in $\mathbb{A}'$ such that

$$\begin{tikzcd} A \arrow[r, "c"] \arrow[dr, "c'"'] & C\mathcal{M} \arrow[d, "h\mathcal{M}"] \\ & C'\mathcal{M} \end{tikzcd}$$

commutes.

A **constraint identity** in $B \in |\mathbb{A}'|$ is a binary relation $k \xrightarrow{\langle s,t \rangle} (A[B])\mathcal{U}'$. An identity $\langle s, t \rangle$ in B is satisfied in a model $A \xrightarrow{c} C\mathcal{M}$ with respect to a model morphism $f\colon B \to C$

[74] The ordinary polynomials from linear algebra are an instantiation of this notion. The word *generalised* plays here the same rôle as the word *general* plays in the so-called "general algebra."

iff $s;[f,c^\sharp]\mathcal{U}' = t;[f,c^\sharp]\mathcal{U}'$, where $c^\sharp$ is the unique 'extension' of c to a model morphism $A^{\$\$} \to C$.

$$\begin{tikzcd} A^{\$\$}\mathcal{M} \arrow[dr, "c^\sharp\mathcal{M}"] & \\ A \arrow[u] \arrow[r, "c"'] & C\mathcal{M} \end{tikzcd} \qquad \begin{tikzcd} A^{\$\$} \arrow[r, "j_{A^{\$\$}}"] \arrow[dr, "c^\sharp"'] & A^{\$\$} \coprod B \arrow[d, "{[f,c^\sharp]}"] & B \arrow[l, "j_B"'] \arrow[dl, "f"] \\ & C & \end{tikzcd}$$

This definition extends to **constraint equations, queries** and their **satisfaction** by constraint models in the same manner as Definition 3.6. □

Example 6.5 An example of a constraint equation in the context of Example 6.1 is

```
open .
vars X Y : Real .
eq < 3.14 * X , Y > + - < Y , 3.14 * X > = 0 .
close
```

Notice that although this equation is *not* satisfied by the standard model $\mathbb{R}^2$, the constraint model $\mathbb{R}+$ does satisfy it. □

Example 6.6 Another example of a constraint sentence in the same context is that of a query:

```
find X Y Z : Real such that
  3 * < X , Y > = < Y , Z > ;
  2.79 * X + Y < Z  = true .
```

Finding a solution to this query in the standard model $\mathbb{R}^2$ reduces by the application of a rewrite step followed by a simplification step to finding a solution for the system of linear inequalities:

```
find X Y Z : Real such that
  3 * X = Y ;
  3 * Y = Z ;
  2.79 * X + Y < Z = true .
```

□

The crucial technical idea of our approach on the semantics of constraint logic programming is to fit constraint logic into category-based equational logic. While this simply cannot be achieved within the usual concrete algebraic or model theoretic approaches (no notion of algebraic signature being abstract enough for this purpose), it works at our level of abstraction. We consider this as a good example of the benefits the use of abstract model theoretic methodology[75] could bring to Computing Science. This idea is resumed by the following slogan and formally formulated by the next definition:

> Constraint logic = equational logic in a special category-based equational signature.

[75] In the sense of category-based equational logic.

Definition 6.7 Let $\langle \mathcal{M}, \mathcal{D} \rangle : (\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ be a liberal morphism of category-based equational signatures. Then any model $A \in |\mathbb{A}|$ determines a forgetful functor $\mathcal{U}'_A : (A{\downarrow}\mathcal{M}) \to \mathbb{X}'$, such that $\mathcal{U}'_A = \mathcal{M}_A ; \mathcal{U}'$, where $\mathcal{M}_A$ is the forgetful functor $(A{\downarrow}\mathcal{M}) \to \mathbb{A}'$.

$$\begin{tikzcd}
\mathbb{A} \arrow[d, "\mathcal{U}"] & \mathbb{A}' \arrow[l, "\mathcal{M}"'] \arrow[d, "\mathcal{U}'"] & (A{\downarrow}\mathcal{M}) \arrow[l, "\mathcal{M}_A"'] \arrow[d, "\mathcal{U}'_A"] \\
\mathbb{X} & \mathbb{X}' \arrow[l, "\mathcal{D}"'] \arrow[r, equal] & \mathbb{X}'
\end{tikzcd}$$

□

In this way the constraint logic introduced by Definition 6.4 is the category-based equational logic determined by the forgetful functor $\mathcal{U}'_A$. The correctness of this definition relies on the following fact:

Fact 6.8 If $\mathcal{U}'$ is faithful and preserves pullbacks, then $\mathcal{U}'_A$ is faithful preserves pullbacks.

Proof: $\mathcal{M}$ preserves pullbacks as a right adjoint. By using this fact, it is straighforward to show that the forgetful functor $\mathcal{M}_A : (A{\downarrow}\mathcal{M}) \to \mathbb{A}'$ creates pullbacks, thus it preserves them too. $\mathcal{U}'_A$ preserves pullbacks as a composite of two pullback preserving functors.

$\mathcal{U}'_A$ is faithful as a composite of two faithful functors, since the forgetful functor $\mathcal{M}_A : (A{\downarrow}\mathcal{M}) \to \mathbb{A}'$ is faithful. □

Proposition 6.9 Let $\langle \mathcal{M}, \mathcal{D} \rangle : (\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ be a liberal morphism of category-based equational signatures. Then for any model $A \in |\mathbb{A}|$

1. there is an isomorphism of categories $(A{\downarrow}\mathcal{M}) \cong (A^{\$\$}{\downarrow}\mathbb{A}')$;
2. if $\mathbb{A}'$ has binary coproducts, then $\mathcal{M}_A$ has a left adjoint; and
3. the forgetful functor $\mathcal{M}_A$ creates filtered colimits.

Proof: 1. Because $\_^{\$\$}$ is a left adjoint to $\mathcal{M}$.

2. Because the forgetful functor $(C{\downarrow}\mathbb{A}') \to \mathbb{A}'$ has a left adjoint for any $C \in |\mathbb{A}'|$ (since $\mathbb{A}'$ has binary coproducts, see also the proof of Lemma 5.50) and by 1.

3. We first show that for any model $C \in |\mathbb{A}'|$, the forgetful functor $(C{\downarrow}\mathbb{A}') \to \mathbb{A}'$ creates filtered colimits. Then we consider $C = A^{\$\$}$ and apply 1.

Let $\{a_i\}_{i \in I}$ be a filtered diagram in $(C{\downarrow}\mathbb{A}')$. The forgetful functor $(C{\downarrow}\mathbb{A}') \to \mathbb{A}'$ maps this diagram into a filtered diagram $\{A_i\}_{i \in I}$ in $\mathbb{A}'$. Consider $\mu : A \to D$ a colimit of this diagram in $\mathbb{A}'$. We define $g : C \to D$ as $a_i ; \mu_i$ for $i \in |I|$; the correctness of this definition is ensured by the fact that $a_i ; \mu_i = a_j ; \mu_j$ for all $i, j \in |I|$ because of the filteredness of I.

$$\begin{tikzcd}
C \arrow[rr, "a_i"] \arrow[dr, "g"] \arrow[ddr, "k"'] & & A_i \arrow[dl, "\mu_i"'] \arrow[ddl, "\gamma_i"] \\
& D \arrow[d, "\theta"] & \\
& E &
\end{tikzcd}$$

Now, we show that μ is a colimiting co-cone $a \to g$ in $(C{\downarrow}\mathbb{A}')$. Consider another co-cone $\gamma : a \to k$ in $(C{\downarrow}\mathbb{A}')$, where $k : C \to E$. γ is also a co-cone $A \to E$ in $\mathbb{A}'$. By

the universal property of μ as a colimiting co-cone in $\mathbb{A}'$, there exists a unique arrow $\theta\colon D \to E$ such that $\mu;\theta = \gamma$ in $\mathbb{A}'$. All it remains to be shown is that θ is a map $g \to k$. But $g;\theta = a_i;\mu_i;\theta$ for some $i \in |I|$. Since $\mu_i;\theta = \gamma_i$ we deduce that $g;\theta = k$. $\square$

6.2 Herbrand's Theorem for Extensible Constraint Logic Programming

Herbrand's Theorem for constraint logic provides mathematical foundations for the concept of constraint solving. Our approach is to instantiate the category-based version of Herbrand's Theorem 3.21 to the particular case of constraint logic viewed as category-based equational logic determined by the forgetful functor $\mathcal{U}'_A$ of Definition 6.4.

Theorem 6.10 Let $\langle \mathcal{M}, \mathcal{D}\rangle\colon (\mathbb{A} \xrightarrow{\mathcal{U}} \mathbb{X}) \to (\mathbb{A}' \xrightarrow{\mathcal{U}'} \mathbb{X}')$ be a liberal morphism of category-based equational signatures. Fix any model $A \in |\mathbb{A}|$. Assume **DomainRegularity** and **DeductionFramework** for $\mathcal{U}'$, and that $\mathcal{U}'$ has a left-adjoint $\mathcal{F}'$ and preserves filtered colimits.

Consider a collection Γ of conditional constraint equations with finite hypotheses and with coequaliser projective quantifiers, and a $\mathcal{U}'$-constraint query $(\exists B)q$ with B is a coequaliser projective model. Then

1. there exists the initial Γ-constraint model 0_Γ;
2. $\Gamma \models (\exists B)q$ iff $0_\Gamma \models (\exists B)q$; and
3. if $\mathbb{A}'$ has non-empty sorts, then $\Gamma \models (\exists B)q$ iff $\Gamma \models (\forall y)q;[h, j_{A^{\$\$}}]$ for some domain $y \in |\mathbb{X}'|$ and some model morphism $h\colon B \to A[y]$.

Proof: The basis of this proof is to regard the constraint sentences (either equations or queries) as ordinary $\mathcal{U}'_A$-sentences (in the sense of Definition 3.6). Any quantifier B of a constraint sentence appears as $A\eta; j_{A^{\$\$}}\mathcal{M}$ in the rôle of the quantifier for the corresponding $\mathcal{U}'_A$-sentence. The category of constraint models is $(A{\downarrow}\mathcal{M})$ and the satisfaction relation between constraint models and constraint sentences reduces to category-based equational satisfaction.

$$A \xrightarrow{A\eta} A^{\$\$}\mathcal{M} \xrightarrow{j_{A^{\$\$}}} (B \coprod A^{\$\$})\mathcal{M}$$

Notice that

- $\mathcal{U}'_A$ has a left-adjoint which is the composite of two left adjoints $\mathbb{X}' \xrightarrow{\mathcal{F}'} \mathbb{A}' \longrightarrow (A{\downarrow}\mathcal{M})$ (see 2. of Proposition 6.9),
- $\mathcal{U}'_A$ preserves filtered colimits as a composite of two filtered preserving functors (see 3. of Proposition 6.9), and
- $(A{\downarrow}\mathcal{M})$ has initial models, i.e., $A \xrightarrow{A\eta} (A^{\$\$}{\downarrow}\mathbb{A}')$ (see 1. of Proposition 6.9) and since the forgetful functor $(A^{\$\$}{\downarrow}\mathbb{A}') \to \mathbb{A}'$ creates limits.

The last general remark is that if B is a coequaliser projective model in $\mathbb{A}'$, then $A\eta; j_{A^{\$\$}}\mathcal{M}$ is coequaliser projective in $(A{\downarrow}\mathcal{M})$. This holds because $A\eta; j_{A^{\$}}\mathcal{M}$ is free over B with respect to the forgetful functor $\mathcal{M}_A$ and because left adjoints to coequaliser preserving functors preserve the coequaliser projectivity (see Lemma 5.21).

1.-2. The congruence closures exist in $(A{\downarrow}\mathcal{M})$ by Proposition 3.28 because $\mathcal{U}'_A$ has a left-adjoint. $\mathcal{U}'_A$ is finitary by Proposition 3.31 and because it preserves filtered colimits. By applying Corollary 3.21 to Γ viewed as a collection of conditional $\mathcal{U}'_A$-equations, we obtain the existence of the initial Γ-model in the category of constraint models and that $\Gamma \models (\exists B)q$ iff $0_\Gamma \models (\exists B)q$.

3. Since $\mathbb{A}'$ has non-empty sorts, for any domain $y \in |\mathbb{X}'|$, there exists at least one arrow $y \to 0_{\mathbb{A}'}\mathcal{U}'$, where $0_{\mathbb{A}'}$ is the initial model in $\mathbb{A}'$. Therefore, there exists at least one arrow $y \to A^{\$\$}\mathcal{U}' = (A\eta)\mathcal{U}'_A$. This means that $(A{\downarrow}\mathcal{M})$ has non-empty sorts. Now, Herbrand's Theorem for non-empty sorts 3.34 applies for Γ viewed as a collection of conditional $\mathcal{U}'_A$-equations. $\square$

In practice, it rarely happens that the sentences in Γ involve the built-in model A. Usually, the sentences in Γ don't involve any elements of the built-in model (i.e., Γ contains only Σ'-sentences, if using the notations from the discussion opening this section) and only the queries appear as full constraint sentences involving elements from the built-in model. In this case, the initial constraint model 0_Γ has a simpler representation as a quotient of the free expansion of the built-in model.

In our category-based framework, the $\mathcal{U}'$-sentences play the rôle of the Σ'-sentences, and they can be canonically viewed as constraint sentences (i.e., $\mathcal{U}'_A$-sentences) via the translation along the morphism of category-based equational signatures $\langle \mathcal{M}_A, 1_{\mathbb{X}'}\rangle : \mathcal{U}' \to \mathcal{U}'_A$ (see Definition 5.20).

Proposition 6.11 Assuming the hypotheses of the previous theorem, suppose that Γ contains only $\mathcal{U}'$-equations. Then the initial constraint model 0_Γ is isomorphic to the canonical map $!_\Gamma = A \xrightarrow{A\eta} A^{\$\$}\mathcal{M} \xrightarrow{e\mathcal{M}} (A^{\$\$}/_{\equiv_\Gamma})\mathcal{M}$, where $\equiv_\Gamma$ is the least congruence on $A^{\$\$}$ closed under Γ-substitutivity.

Proof: We will show that $!_\Gamma$ satisfies the initiality property in the full subcategory of $(A{\downarrow}\mathcal{M})$ of all models satisfying $\hat{\Gamma}$, where $\hat{\Gamma}$ is the translation of Γ along $\langle \mathcal{M}_A, 1_{\mathbb{X}'}\rangle$.

Let $f : A \to C\mathcal{M}$ be any constraint model satisfying $\hat{\Gamma}$. By the Satisfaction Condition (Theorem 5.33), this is equivalent to $C \models \Gamma$. All we have to prove is that there exists a unique arrow $f^\sharp : A^{\$\$}/_{\equiv_\Gamma} \to C$ such that $!_\Gamma; f^\sharp\mathcal{M} = f$.

$$\begin{tikzcd}
A \arrow[r, "A\eta"] \arrow[dr, "f"'] & A^{\$\$}\mathcal{M} \arrow[r, "e\mathcal{M}"] \arrow[d, "f'\mathcal{M}"] & (A^{\$\$}/_{\equiv_\Gamma})\mathcal{M} \arrow[dl, "f^\sharp\mathcal{M}"] \\
& C\mathcal{M} &
\end{tikzcd}$$

By the universal property of the free extension $A\eta$ along $\mathcal{M}$, there exists a unique map $f' : A^{\$\$} \to C$ such that $A\eta; f'\mathcal{M} = f$. By the universal property of e (Theorem 3.17), there exists a unique map $f^\sharp : A^{\$\$}/_{\equiv_\Gamma} \to C$ such that $e; f^\sharp = f'$. $\square$

In the case of order sorted Horn clause logic with equality, Goguen and Meseguer have proved in [39] the existence of initial constraint models for the particular case of Γ containing only Σ'-sentences. This result crucial for the semantics of extensible contraint logic programming in Eqlog can obtained as an instantiation of our previous results.

As pointed out by Goguen and Meseguer in [39], the notion of *protecting expansion* gives the right semantic condition for built-ins. This means that 0_Γ must *protect* the built-in model A, i.e., that 0_Γ is an isomorphism $A \cong (A^{\$\$}/_{\equiv_\Gamma})\mathcal{M}$, where $\equiv_\Gamma$ is the

least congruence on $A^{\$\$}$ closed under Γ-substitutivity. In the context of order sorted Horn clause logic with equality, Goguen and Meseguer [39] give a set of conditions that guarantee protection but impose some restrictions on the sentences in Γ. However, these restrictions are almost always met in practice. We mention their result:

Proposition 6.12 Let $(S, \leq, \Sigma, \Pi) \hookrightarrow (S', \leq, \Sigma', \Pi')$ be an inclusion of order sorted first order signatures and Γ be an order sorted Horn clause logic with equality specification in $(S, \leq, \Sigma, \Pi)$ such that:

(1) $s \in S' - S$ for any operator symbol $\sigma \in \Sigma'_{w',s'} - \Sigma_{w',s'}$,

(2) if $s \in S$ and $s' \in S'$ and $s' \leq s$, then $s' \in S$ and $s' \leq s$ in S,

(3) for π predicate symbol, if $\pi \in \Pi_w$ and $\pi \in \Pi'_{w'}$, then $\pi \in \Pi_{w'}$, and

(4) Γ doesn't involve operation symbols from $\Sigma' - \Sigma$ and contains only clauses whose heads are all predicate symbols from $\Pi' - \Pi$.

Then for any $(S, \leq, \Sigma, \Pi)$-model A, its free extension to a Γ-model is an isomorphism. $\square$

7 Conclusions and Future Work

This thesis developed a category-based semantics for equational and constraint logic programming that is based on the concept of category-based equational logic. We showed how this general framework can be successfully applied to topics like equational proof theory, paramodulation-based operational semantics, modularisation in equational logic programming, and extensible constraint logic programming.

An abstract version of Herbrand's Theorem was derived as a consequence of the completeness result for the category-based equational deduction. This not only provides mathematical foundations for the equational logic programming paradigm, but also constitutes a basis for the full integration of constraint logic programming into this programming paradigm.

We developed a model theoretic approach to the completeness of the operational semantics of equational logic programming languages based on the analysis of the relationship between the congruence and the paramodulation relation induced by a program on a given model. We showed that this approach covers the case of computations modulo a set of axioms naturally, in the sense that no special treatment is necessary anymore for this case. However, the full implications of this approach to the case of computations modulo a set of axioms still remain to be discovered. One particular way would be to lift the treatment of narrowing and its refinement at the same level of abstraction to that of paramodulation.

The concept of category-based equational signature morphism has been successfully used for setting up the mathematical structures underlying the fundamental modularisation problems specific to equational logic programming. Also, category-based equational signature morphisms, proved to be central for the category-based semantics of extensible constraint logic programming. Based on this semantics, further work can be done to develop theories and technologies for *extensible modular constraint programming*. The principles underlying Eqlog module system should provide a good basis for developing a technology for combinig decision procedures. A concrete operational semantics will define a control strategy for combining various efficient decision procedures for specific problems, with narrowing and resolution as a general inference mechanism. This will involve backtracking, the introduction of symbolic variables (i.e., deferred constants), the computation of symbolic solutions, and symbolic simplification (e.g., see [68, 59] for the case of linear arithmetic constraints). A promising approach is given by the category-based approach to the paramodulation-based operational semantics for equational logic programming developed in Chapter 4. Since constraint logic can be regarded as a particular case of category-based equational logic, we expect to obtain some relevant results by applying that theory.

One of the most important further research directions is to apply the category-based results of this work for developing equational logic programming over non-conventional structures. This might provide the right framework for integrating equational and constraint logic programming with other programming paradigms, especially higher-order programming, object-orientation, or concurrency.

On the implementation side, much work has to be done for building an efficient Eqlog compiler that will support extensible modular constraint solving. The actual Eqlog prototype implementation is an extension of the OBJ3 system that implements leftmost innermost order sorted basic narrowing with constructor discipline, and it can be suc-

cessfully used for experimentations with the operational semantics.

Finally, we can conclude that the framework of category-based equational logic can be regarded as a mathematical structure that is fundamental to the equational logic programming paradigm. We have seen how a wide spectrum of problems in this area can be successfully solved within this framework, and I hope that the theory developed here can be used for solving many other problems raised by such a dynamic field as equational logic programming is today.

A Running Eqlog

This appendix gives a brief presentation of all necessary information for running the Oxford implementation of Eqlog. It is assumed the reader has some familiarity with the user manual for the OBJ3 system [46].

The way to input Eqlog files is similar to that of the OBJ3 system. The name of Eqlog files must end in `.eql`. OBJ files can be also loaded, but only by using their full name (i.e., including `.obj`). Each time an Eqlog module is loaded or selected, the system computes a couple of hash tables used by the order sorted unification function.[76]

Eqlog syntax (in BNF notation) for solving queries is as follows:

$\langle Solve \rangle$ ::= `find` $\langle LogicVarsDeclar \rangle$ `such that` $\langle queries \rangle$.

$\langle LogicVarsDeclar \rangle$ ::= $\langle VarId \rangle$... : $\langle Sort \rangle$ [, $\langle VarId \rangle$... : $\langle Sort \rangle$]...

$\langle queries \rangle$ ::= $\langle Term \rangle$ = $\langle Term \rangle$ [; $\langle Term \rangle$ = $\langle Term \rangle$]...

Eqlog modules are the same as the OBJ modules except that the shape of the hypotheses part of an clause is restricted to

$\langle Term \rangle$ == $\langle Term \rangle$ [and $\langle Term \rangle$ == $\langle Term \rangle$]...

$\langle VarId \rangle$ and $\langle Sort \rangle$ stand for the OBJ syntactical entities of **variable identifiers** and **sorts**, while $\langle Term \rangle$ stands for the OBJ terms. The BNF definition for all OBJ syntactical entities can be found in the OBJ manual [46].

Operator declarations in Eqlog admit `cons` as an attribute meaning that the corresponding operator is regarded as a constructor.

[76] In the case of big modules, the computation of these hash tables could be quite time consuming!

References

[1] Hajnal Andréka and István Németi. A general axiomatizability theorem formulated in terms of cone-injective subcategories. In B. Csakany, E. Fried, and E.T. Schmidt, editors, *Universal Algebra*, pages 13–35. North-Holland, 1981. Colloquia Mathematics Societas János Bolyai, 29.

[2] John Backus. Can programming be liberated from the von Neumann styleΓ *Communications of the Association for Computing Machinery*, 21(8):613–641, 1978.

[3] *et al.* Barbara Liskov. *CLU Reference Manual.* Springer-Verlag, 1981.

[4] Michael Barr and Charles Wells. *Toposes, Triples and Theories.* Springer, 1984. Grundlehren der mathematischen Wissenschafter, Volume 278.

[5] Jon Barwise. Axioms for abstract model theory. *Annals of Mathematical Logic*, 7:221–265, 1974.

[6] Jon Barwise and Solomon Feferman. *Model-Theoretic Logics.* Springer, 1985.

[7] M. Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Math. Systems Theory*, 20, 1987.

[8] Jean Benabou. Structures algébriques dans les catégories. *Cahiers de Topologie et Géometrie Différentiel*, 10:1–126, 1968.

[9] Jan Bergstra, Jan Heering, and Paul Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.

[10] Garrett Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

[11] Rod Burstall and Răzvan Diaconescu. Hiding and behaviour: an institutional approach. In A. William Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*, pages 75–92. Prentice-Hall, 1994. Also Technical Report ECS-LFCS-8892-253, Laboratory for Foundations of Computer Science, University of Edinburgh, 1992.

[12] Rod Burstall and Joseph Goguen. Putting theories together to make specifications. In Raj Reddy, editor, *Proceedings, Fifth International Joint Conference on Artificial Intelligence*, pages 1045–1058. Department of Computer Science, Carnegie-Mellon University, 1977.

[13] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *Proceedings, 1979 Copenhagen Winter School on Abstract Software Specification*, pages 292–332. Springer, 1980. Lecture Notes in Computer Science, Volume 86; based on unpublished notes handed out at the Symposium on Algebra and Applications, Stefan Banach Center, Warsaw, Poland, 1978.

[14] C.C.Chang and H.J.Keisler. *Model Theory.* North Holland, Amsterdam, 1973.

[15] Alain Colmerauer. An introduction to PrologIII. Technical report, Groupe Intelligence Artificielle, Faculte de Sciences de Luminy.

[16] Alain Colmerauer. Prolog II. Manuel de référence et modèle théorique. Technical report, GIA Luminy, Marseille, 1982.

[17] Hubert Comon. Equational formulas in order-sorted algebras. In *Proceedings, ICALP '90*, Warwick, 1990. Springer Verlag.

[18] Virgil Căzănescu. Local equational logic. In Zoltan Esik, editor, *Proceedings, 9th International Conference on Fundamentals of Computation Theory FCT'93*, pages 162–170. Springer-Verlag, 1993. Lecture Notes in Computer Science, Volume 710.

[19] Nachum Dershowitz. Computing with rewrite rules. Technical Report ATR-83(8478)-1, The Aerospace Corp., 1983.

[20] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewriting systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Methods and Semantics*, pages 243–320. North-Holland, 1990.

[21] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993. Proceedings of a Workshop held in Edinburgh, Scotland, May 1991.

[22] Laurent Fribourg. SLOG: A logic programming language interpreter based on clausal superposition and rewriting. In *Proceedings, SLP '85*, pages 172–185. 1985.

[23] Joseph Goguen. Categorical foundations for general systems theory. In F. Pichler and R. Trappl, editors, *Advances in Cybernetics and Systems Research*, pages 121–130. Transcripta Books, 1973.

[24] Joseph Goguen. Semantics of computation. In Ernest G. Manes, editor, *Proceedings, First International Symposium on Category Theory Applied to Computation and Control*, pages 234–249. University of Massachusetts at Amherst, 1974. Also in Lecture Notes in Computer Science, Volume 25, Springer, 1975, pages 151–163.

[25] Joseph Goguen. Order sorted algebra. Technical Report 14, UCLA Computer Science Department, 1978. Semantics and Theory of Computation Series.

[26] Joseph Goguen. Reusing and interconnecting software components. *Computer*, 19(2):16–28, February 1986. Reprinted in *Tutorial: Software Reusability*, Peter Freeman, editor, IEEE Computer Society, 1987, pages 251–263, and in *Domain Analysis and Software Systems Modelling*, Rubén Prieto-Díaz and Guillermo Arango, editors, IEEE Computer Society, 1991, pages 125–137.

[27] Joseph Goguen. What is unificationΓ A categorical view of substitution, equation and solution. In Maurice Nivat and Hassan Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques*, pages 217–261. Academic, 1989. Also Report SRI-CSL-88-2R2, SRI International, Computer Science Lab, August 1988.

[28] Joseph Goguen. Higher-order functions considered unnecessary for higher-order programming. In David Turner, editor, *Research Topics in Functional Programming*,

pages 309–352. Addison Wesley, 1990. University of Texas at Austin Year of Programming Series; preliminary version in SRI Technical Report SRI-CSL-88-1, January 1988.

[29] Joseph Goguen. Types as theories. In George Michael Reed, Andrew William Roscoe, and Ralph F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 357–390. Oxford, 1991. Proceedings of a Conference held at Oxford, June 1989.

[30] Joseph Goguen. *Theorem Proving and Algebra.* MIT, to appear.

[31] Joseph Goguen and Rod Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. Technical Report Report CSL-118, SRI Computer Science Lab, October 1980.

[32] Joseph Goguen and Rod Burstall. A study in the foundations of programming methodology: Specifications, institutions, charters and parchments. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Conference on Category Theory and Computer Programming*, pages 313–333. Springer, 1986. Lecture Notes in Computer Science, Volume 240; also, Report CSLI-86-54, Center for the Study of Language and Information, Stanford University, June 1986.

[33] 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.

[34] Joseph Goguen and Răzvan Diaconescu. An Oxford survey of order sorted algebra. *Mathematical Structures in Computer Science*, 4:363–392, 1994.

[35] Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Harmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification*, volume 785 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 1994.

[36] Joseph Goguen and Susanna Ginali. A categorical approach to general systems theory. In George Klir, editor, *Applied General Systems Research*, pages 257–270. Plenum, 1978.

[37] Joseph Goguen and José Meseguer. Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985. Preliminary versions have appeared in: *SIGPLAN Notices*, July 1981, Volume 16, Number 7, pages 24–37; SRI Computer Science Lab, Report CSL-135, May 1982; and Report CSLI-84-15, Center for the Study of Language and Information, Stanford University, September 1984.

[38] Joseph Goguen and José Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In Douglas DeGroot and Gary Lindstrom, editors, *Logic Programming: Functions, Relations and Equations*, pages 295–363. Prentice-Hall, 1986.

An earlier version appears in *Journal of Logic Programming*, Volume 1, Number 2, pages 179–210, September 1984.

[39] Joseph Goguen and José Meseguer. Models and equality for logical programming. In Hartmut Ehrig, Giorgio Levi, Robert Kowalski, and Ugo Montanari, editors, *Proceedings, 1987 TAPSOFT*, pages 1–22. Springer, 1987. Lecture Notes in Computer Science, Volume 250.

[40] Joseph Goguen and José Meseguer. Unifying functional, object-oriented and relational programming, with logical semantics. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT, 1987. Preliminary version in *SIGPLAN Notices*, Volume 21, Number 10, pages 153–162, October 1986.

[41] Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992. Also Programming Research Group Technical Monograph PRG–80, Oxford University, December 1989, and Technical Report SRI-CSL-89-10, SRI International, Computer Science Lab, July 1989; originally given as lecture at *Seminar on Types*, Carnegie-Mellon University, June 1983; many draft versions exist, from as early as 1985.

[42] Joseph Goguen, Andrew Stevens, Keith Hobley, and Hendrik Hilberdink. 2OBJ, a metalogical framework based on equational logic. *Philosophical Transactions of the Royal Society, Series A*, 339:69–86, 1992. Also in *Mechanized Reasoning and Hardware Design*, edited by C.A.R. Hoare and M.J.C. Gordon, Prentice-Hall, 1992, pages 69–86.

[43] Joseph Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In Raymond Yeh, editor, *Current Trends in Programming Methodology, IV*, pages 80–149. Prentice-Hall, 1978.

[44] Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. Abstract data types as initial algebras and the correctness of data representations. In Alan Klinger, editor, *Computer Graphics, Pattern Recognition and Data Structure*, pages 89–93. IEEE, 1975.

[45] Joseph Goguen, James Thatcher, Eric Wagner, and Jesse Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95, January 1977. An early version is "Initial Algebra Semantics", with James Thatcher, IBM T.J. Watson Research Center, Report RC 4865, May 1974.

[46] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen, editor, *Algebraic Specification with OBJ: An Introduction with Case Studies*. Cambridge, to appear 1995. Also to appear as Technical Report from SRI International.

[47] Joseph Goguen and David Wolfram. On types and FOOPS. In Robert Meersman, William Kent, and Samit Khosla, editors, *Object Oriented Databases: Analysis,*

Design and Construction, pages 1–22. North Holland, 1991. Proceedings, IFIP TC2 Conference, Windermere, UK, 2–6 July 1990.

[48] George Gratzer. *Universal Algebra*. Springer, 1979.

[49] Michael Hanus. Compiling logic programs with equality. In *Proc. Int. Workshop on Programming Language Implementation and Logic Programming*, pages 387–401. Springer LNCS 456, 1990.

[50] Robert Harper, David MacQueen, and Robin Milner. Standard ML. Technical Report ECS-LFCS-86-2, Department of Computer Science, University of Edinburgh, 1986.

[51] William S. Hatcher. Quasiprimitive categories. *Math. Ann.*, (190):93–96, 1970.

[52] Horst Herrlich and C.M.Ringel. Identities in categories. *Can. Math. Bull.*, (15):297–299, 1972.

[53] Phillip J. Higgins. Algebras with a scheme of operators. *Mathematische Nachrichten*, 27:115–132, 1963.

[54] Steffen Hölldobler. Foundations of equational logic programming. In *Lecture Notes in Artificial Intelligence*, number 353. Springer Verlag, 1988.

[55] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980. Preliminary version in *Proceedings*, 18th IEEE Symposium on Foundations of Computer Science, IEEE, 1977, pages 30–45.

[56] Gérard Huet and Derek Oppen. Equations and rewrite rules: A survey. In Ron Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic, 1980.

[57] Jean-Marie Hullot. Canonical forms and unification. In Wolfgang Bibel and Robert Kowalski, editors, *Proceedings, 5th Conference on Automated Deduction*, pages 318–334. Springer, 1980. Lecture Notes in Computer Science, Volume 87.

[58] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *14th ACM Symposium on the Principles of Programming languages*, pages 111–119. 1987.

[59] K. McAloon Jean-Louis Lassez and T. Huynh. Simplification and elimination of redundant linear arithmetic constraints. Technical report, IBM Thomas J. Watson Research Center.

[60] Horts Herrlich Jiri Adamek and George Strecker. *Abstract and Concrete Categories*. John Wiley, 1990.

[61] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *Proceedings, 11th Symposium on Principles of Programming Languages*, 1984. In *SIAM Journal of Computing*.

[62] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT-Press, 1991.

[63] Jan Willem Klop. Term rewriting systems: from Church-Rosser to Knuth-Bendix and beyond. In Samson Abramsky, Dov Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science*. Oxford, 1992.

[64] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

[65] Dallas Lankford and A.M. Ballantyne. Decision procedures for simple equational theories with permutative axioms: Complete sets of permutative reductions. Technical Report ATP-37, Dept. of Mathematics and Computer Science, Univ. of Texas, Austin, 1977.

[66] Jean-Louis Lassez. From LP to LP: Programming with Constraints. Technical report, IBM T.J. Watson Research Center.

[67] Jean-Louis Lassez, Michael Maher, and Kimbal Marriott. Unification revisited. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, 1988.

[68] Jean-Louis Lassez and Michael J. Maher. On Fourier's algorithm for linear arithmetic constraints. Technical report, IBM Thomas J. Watson Research Center.

[69] F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings, National Academy of Sciences, U.S.A.*, 50:869–872, 1963. Summary of Ph.D. Thesis, Columbia University.

[70] Ernest Manes. *Algebraic Theories*. Springer, 1976. Graduate Texts in Mathematics, Volume 26.

[71] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[72] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium, 1987*, pages 275–329. North-Holland, 1989.

[73] José Meseguer and Joseph Goguen. Order-sorted algebra solves the constructor selector, multiple representation and coercion problems. *Information and Computation*, 103(1):114–158, March 1993.

[74] José Meseguer, Joseph Goguen, and Gert Smolka. Order-sorted unification. *Journal of Symbolic Computation*, 8:383–413, 1989. Preliminary version appeared as Report CSLI-87-86, Center for the Study of Language and Information, Stanford University, March 1987.

[75] Peter Mosses. Unified algebras and institutions. In *Proceedings, Fourth Annual Conference on Logic in Computer Science*, pages 304–312. IEEE, 1989.

[76] Michael J. O'Donnell. Computing in systems described by equations. In *Lecture Notes in Computer Science*, volume 58. Springer, 1977.

[77] David Parnas. Information distribution aspects of design methodology. *Information Processing '72*, 71:339–344, 1972. Proceedings of 1972 IFIP Congress.

[78] David Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the Association for Computing Machinery*, 15:1053–1058, 1972.

[79] David Parnas. A technique for software module specification. *Communications of the Association for Computing Machinery*, 15:330–336, 1972.

[80] G.A. Robinson and L. Wos. Paramodulation and theorem proving in first order theories with equality. In *Machine Intelligence*, number 4. 1969.

[81] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[82] Pieter Hendrik Rodenburg. A simple algebraic proof of the equational interpolation theorem. *Algebra Universalis*, 28:48–51, 1991.

[83] Donald Sannella and Andrzej Tarlecki. Extended ML: an institution independent framework for formal program development. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Summer Workshop on Category Theory and Computer Programming*, pages 364–389. Springer, 1986. Lecture Notes in Computer Science, Volume 240.

[84] Donald Sannella and Andrzej Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Science*, 34:150–178, 1987. Earlier version in *Proceedings, Colloquium on Trees in Algebra and Programming*, Lecture Notes in Computer Science, Volume 185, Springer, 1985.

[85] Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Control*, 76:165–210, 1988. Earlier version in *Proceedings, International Symposium on the Semantics of Data Types*, Lecture Notes in Computer Science, Volume 173, Springer, 1985.

[86] J.R. Slagle. Automatic theorem proving in theories with simplifiers, commutativity and associativity. *Journal of ACM*, 21:622–642, 1974.

[87] Gert Smolka. *Logic Programming over Polymorphically Order-Sorted Types.* PhD thesis, University of Kaiserslautern, 1989. FB Informatik.

[88] Ross Street. The formal theory of monads. *Jour. of Pure and Applied Algebra*, (2):149–169, 1972.

[89] Andrzej Tarlecki. Free constructions in algebraic institutions. In M.P. Chytil and V. Koubek, editors, *Proceedings, International Symposium on Mathematical Foundations of Computer Science*, pages 526–534. Springer, 1984. Lecture Notes in Computer Science, Volume 176; extended version, University of Edinburgh, Computer Science Department, Report CSR-149-83, and revised version 'On the existence of Free Models in Abstract Algebraic Institutions', September 1984.

[90] Andrzej Tarlecki. Bits and pieces of the theory of institutions. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Summer Workshop on Category Theory and Computer Programming*, pages 334–360. Springer, 1986. Lecture Notes in Computer Science, Volume 240.

[91] Andrzej Tarlecki. On the existence of free models in abstract algebraic institutions. *Theoretical Computer Science*, 37:269–304, 1986. Preliminary version, University of Edinburgh, Computer Science Department, Report CSR-165-84, 1984.

[92] Andrzej Tarlecki. Quasi-varieties in abstract algebraic institutions. *Journal of Computer and System Sciences*, 33(3):333–360, 1986. Original version, University of Edinburgh, Report CSR-173-84.

[93] Alfred Tarski. The semantic conception of truth. *Philos. Phenomenological Research*, 4:13–47, 1944.

[94] Will Tracz. Parameterized programming in LILEANNA. In *Proceedings, Second International Workshop on Software Reuse*, March 1993. Lucca, Italy.

[95] 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.

[96] Christoph Walther. A mechanical solution of Schubert's steamroller problem by many-sorted resolution. *Artificial Intelligencer*, 26(2):217–214, 1985.

[97] Christoph Walther. Many-sorted inferences in automated theorem proving. In *Sorts and Types in Artificial Intelligence*, pages 18–48. Springer, 1990. Lecture Notes in Artificial Intelligence, Volume 418.

[98] Gio Wiederhold, Peter Wegner, and Stefano Ceri. Toward megaprogramming. *Communications of the ACM*, 35(11):89–99, 1992.

Contents

1 Introduction 2
- 1.1 The Equational Logic Programming Paradigm . . . 2
  - 1.1.1 A historical perspective . . . 2
  - 1.1.2 Equational logic programming . . . 3
- 1.2 Contributions of this Thesis . . . 4
  - 1.2.1 Beyond conventional "abstract model theory" . . . 5
  - 1.2.2 Category-based equational logic . . . 6
  - 1.2.3 Category-based operational semantics . . . 7
  - 1.2.4 Modularisation and extensible constraint logic programming . . . 8
- 1.3 The Structure of the Thesis . . . 9
  - 1.3.1 Preliminaries . . . 9
  - 1.3.2 Category-based Equational Deduction . . . 9
  - 1.3.3 Operational Semantics . . . 10
  - 1.3.4 Modularisation . . . 10
  - 1.3.5 Extensible Constraint Logic Programming . . . 11
- 1.4 The Programming Language Eqlog . . . 11
  - 1.4.1 Eqlog as a framework for decision procedures . . . 12

2 Preliminaries 13
  - 2.0.2 Comma categories . . . 13
  - 2.0.3 Limits and colimits . . . 13
  - 2.0.4 2-categories . . . 14
- 2.1 Categorical Relations . . . 15
  - 2.1.1 Representations of binary relations . . . 15
  - 2.1.2 Unions of relations . . . 16
  - 2.1.3 Equivalences . . . 17
- 2.2 Finiteness . . . 18
  - 2.2.1 Finite objects . . . 18
  - 2.2.2 Finiteness for binary relations . . . 19
  - 2.2.3 Reflexive-transitive closures . . . 20
  - 2.2.4 Confluent relations . . . 22
- 2.3 Models and Domains . . . 22
  - 2.3.1 Many sorted algebra . . . 23
  - 2.3.2 Order sorted algebra . . . 24
  - 2.3.3 Horn clause logics . . . 25
  - 2.3.4 Equational logic modulo axioms . . . 29
  - 2.3.5 Summary of Examples . . . 29

3 Category-based Equational Deduction 30
- 3.1 Congruences . . . 30
- 3.2 Equations, Queries and Satisfaction . . . 31
- 3.3 Completeness . . . 33
- 3.4 Herbrand's Theorem . . . 36
- 3.5 Consequences of Freeness . . . 37
  - 3.5.1 The existence of congruence closures . . . 37
  - 3.5.2 Finitary model operations . . . 39

3.5.3 The Axiom of Choice *versus* projectivity 40
3.5.4 Herbrand's Theorem revisited 41

4 Operational Semantics 43
4.0.5 Completeness of Paramodulation: its Architecture 43
4.1 Preliminaries 45
4.1.1 Rewriting contexts 45
4.2 Inference Rules 48
4.2.1 Resolution as a refinement of paramodulation 49
4.3 Model Theoretic Paramodulation 50
4.3.1 The paramodulation relation 50
4.3.2 Completeness of model theoretic paramodulation 53
4.4 Paramodulation modulo a Model Morphism 55
4.5 Confluence 57
4.5.1 Model theoretic rewriting 58
4.5.2 Transitivity *versus* confluence 59
4.5.3 Confluence modulo a Model Morphism 61
4.6 Narrowing in MSA 61
4.6.1 Canonical term rewriting systems 63
4.7 Computing in Eqlog 63
4.7.1 OS unification in Eqlog 63
4.7.2 Examples with narrowing 64
4.7.3 Constructor discipline 67

5 Modularisation 68
5.0.4 Some History 69
5.1 Institutions and Modularisation 70
5.1.1 Exactness 71
5.1.2 Parametric modules and views 73
5.2 Satisfaction Condition for Category-based Equational Logic 76
5.2.1 Many-sorted institutions 77
5.2.2 Sentence translations along morphisms of category-based equational signatures 78
5.2.3 The Satisfaction Condition 84
5.3 Queries and Solutions *versus* Modularisation 85
5.3.1 The institution of queries and substitutions 86
5.3.2 Soundness and completeness for module imports 87
5.4 Theorem of Constants 90
5.4.1 The level of institutions 90
5.4.2 The level of category-based equational logic 91

6 Extensible Constraint Logic Programming 93
6.1 Generalised Polynomials and Constraint Satisfaction 94
6.1.1 Internal constraint logic 95
6.2 Herbrand's Theorem for Extensible Constraint Logic Programming . . . 98

7 Conclusions and Future Work 101

A Running Eqlog **103**