

INSTITUTION-INDEPENDENT LOGIC PROGRAMMING

by

IONUȚ ȚUȚU

A thesis submitted for the degree of
Doctor of Philosophy



Department of Computer Science
Royal Holloway University of London

2015

DECLARATION

I hereby declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that it has not been submitted for any other degree or qualification at Royal Holloway University of London or any other institution.

A handwritten signature in black ink, appearing to read 'I. Țuțu', with a stylized flourish at the end.

Ionuț Țuțu
2015

ABSTRACT

The logic-programming paradigm is a prime example of how the integration of the declarative and the operational aspects of logical systems can be used to provide a unified framework for both specification and programming languages. In essence, programming in logic amounts to giving appropriate axiomatic formalizations of computable functions, which can then be executed by means of carefully designed goal-directed deduction rules.

In this thesis, we examine common features of various conventional logic-programming languages, ranging from the most traditional variant of the paradigm – defined over Horn-clause logic – to first-order and higher-order equational logic programming. Based on these, we propose an abstract model-theoretic framework that allows us to develop and conduct research into logic programming over an arbitrary logical system, without concrete models, sentences, satisfaction, or deduction, and thus to explore the logic-programming paradigm for other, less conventional formalisms, like the logic of orchestration schemes used in the context of service-oriented computing.

Our study is based on abstractions of notions such as logic program, clause, query, solution, and computed answer, which we develop over Goguen and Burstall's theory of institutions. These give rise to a series of concepts that formalize the interplay between the denotational and the operational semantics of logic programming. We investigate properties concerning the satisfaction of quantified sentences, discuss a variant of Herbrand's theorem that is not limited in scope to any logical formalism or construction of logic programs, and define a sound and conditionally complete procedure for computing solutions to queries. Within this setting, we further examine two of the most fundamental aspects of the modularization of logic programs – the preservation and the reflection of solutions along morphisms of programs – leading to results that can be applied not only to unstructured logic programs (plain sets of clauses), but also to elaborate module systems.

ACKNOWLEDGEMENTS

The material in this thesis is a natural extension of a series of developments in institutional abstract model theory, algebraic specification, and logic programming. I would like to thank the members of these communities, and some of them in particular below, for their contributions to logic and computer science, without which this research would not have been possible.

First of all, I would like to express my deepest gratitude to my supervisor, José Luiz Fiadeiro, for his guidance, dedication, and support during my PhD studies, and generally for giving me the opportunity to investigate the logic-programming paradigm from an institutional point of view. His research questions, ideas, and critical feedback have been invaluable for the completion of this project and for my development as a researcher.

Special thanks go to Răzvan Diaconescu for introducing me to the elegant theory of institutions and for his constant encouragement and friendship. I also owe a great deal to Virgil Căzănescu, Mihai Codescu, Daniel Găină, Antónia Lopes, Till Mossakowski, Fernando Orejas, and Uwe Wolter for many pleasant and inspiring discussions, all of which have contributed to the present form of this thesis. My research has benefited from the constructive advice and feedback of the members of the Department of Computer Science at the University of Bucharest, who kindly invited me to present different aspects of my work, at various stages of their development, whenever I visited my home country. In addition, I would like to acknowledge the members of Department of Computer Science at Royal Holloway University of London for creating a friendly and welcoming atmosphere, as well as a good research environment. I am very grateful for all of that.

Last but far from least, I would like to thank my parents, Oana, and Claudia for being supportive, patient, and understanding throughout these years.

This research has been partly funded by an EPSRC Doctoral Training Grant between 2011 and 2012, a Reid Scholarship awarded by Royal Holloway University of London from 2012 onward, and by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0439.

CONTENTS

1	INTRODUCTION	7
1.1	The logic-programming paradigm	7
1.2	Towards abstract logic programming	8
1.3	The structure of the thesis	12
1.4	Relevant publications	14
2	TECHNICAL PRELIMINARIES	15
2.1	Categories	15
2.2	Institutions	18
2.3	Institutions as functors	21
3	INTRODUCING LOGIC-INDEPENDENT LOGIC PROGRAMMING	23
3.1	The logical system of reference	23
3.1.1	Institutions of substitutions	25
3.2	Substitution systems	28
3.2.1	Quantified sentences	34
3.3	Abstract logic programming	36
3.3.1	Herbrand's theorem	40
3.3.2	Operational semantics	43
3.3.3	Completeness	47
4	CONNECTION WITH INSTITUTIONS	52
4.1	Equational logic programming	52
4.1.1	First-order equational logic	53
4.1.2	Higher-order logic with Henkin semantics	54
4.2	Institution-independent substitutions	55
4.3	Quantification spaces	57
4.3.1	Representable signature extensions	59
4.3.2	Representable substitutions	61
4.3.3	Deriving generalized substitution systems	65
4.4	Logic programming over an arbitrary institution	67
4.4.1	Paramodulation	67

4.4.2	Herbrand's theorem revisited	69
5	A LOGIC-PROGRAMMING SEMANTICS OF SERVICES	72
5.1	Orchestration schemes	72
5.1.1	Program expressions	73
5.1.2	Asynchronous relational networks	79
5.2	A logical view on service discovery and binding	95
5.2.1	A generalized substitution system of orchestration schemes	96
5.2.2	The clausal structure of services	103
5.2.3	Resolution as service discovery and binding	105
6	SOLVING QUERIES OVER MODULAR LOGIC PROGRAMS	112
6.1	Program modules	112
6.2	Foundations of modular logic programming	114
6.3	Computing answer substitutions	117
7	CONCLUSIONS AND FURTHER WORK	121
	BIBLIOGRAPHY	124
	INDEX OF NOTATIONS	136
	INDEX OF CONCEPTS	141

INTRODUCTION

This thesis concerns the model-theoretic foundations of logic programming from the vantage point of the theory of institutions put forward by Goguen and Burstall in the late 1970s in the context of algebraic specification. Its purpose is to provide a general mathematical framework for the study of logic programming so that, through abstraction, one could gain a more accurate and deeper understanding of the intrinsic nature of the paradigm. In this introduction, we discuss the motivation that underlies the present research and give an overview of the following chapters.

1.1 THE LOGIC-PROGRAMMING PARADIGM

The essential idea of exploring the computational aspects of logical inference as a foundation for the development of new programming paradigms was first investigated by Kowalski [KK71; Kow74] and Colmerauer [Col+73] based on the pioneering work of Herbrand in proof theory [see Her67] and on the introduction by Robinson [Rob65] of resolution as an inference rule well suited for automation. Logic programming thus originated from the observation that a considerable fragment of first-order logic has a natural computational interpretation that makes it adequate not only as a declarative language but also as a programming language. To be more precise, logic programming (a) defines computable functions by way of standard constructs from model theory and (b) executes these definitions as programs through goal-directed deductions that are performed according to a fixed strategy. This means that logic programming provides, at the same time, both a denotational and an operational perspective on programs.

Conventional logic programming is semantically based upon the Horn-clause fragment of (single-sorted) first-order logic without equality, and implements deduction as backwards reasoning defined in terms of resolution steps [see Llo87]. However, the essence of the paradigm is, to a great extent, independent of any logical system of choice. This is reflected by a multitude of variants that have been developed over time, such as order-sorted [GM86]

and category-based equational logic programming [Dia95], constraint logic programming [JL87] (see also [Dia00] for a presentation that is more closely related to the setting of this thesis), as well as higher-order [Mes89; Mes92] and behavioural (i.e. object-oriented) logic programming [GMK02]. Even more, recent developments in the operational semantics of systems in the context of service-oriented computing [see e.g. FLB11] have suggested a deep connection with logic programming. This connection was made explicit in [TF13] with the introduction of a service-oriented variant of the paradigm, which we will further pursue in Chapter 5 of this thesis.

1.2 TOWARDS ABSTRACT LOGIC PROGRAMMING

Through our inquiry, we aim to provide a single unifying framework for the model-theoretic foundations of logic programming, with a high level of abstraction, that incorporates ideas from concrete instances of the phenomena but without being committed to any particular formalism. To this end, our work has roots in the theory of institutions [GB92] – a major field of study in universal logic that originated in computer science – and draws inspiration from previous institution-theoretic developments related to logic programming such as [Tar86; Dia04] and also [Mes89]. We propose an axiomatic approach to logic programming, and intrinsically to logic programs, that is founded on a three-level hierarchy of concepts meant to capture:

- 1 the denotational semantics of logic programming, based on a notion of *generalized substitution system* that extends institutions with appropriate abstractions of variables, substitutions, local sentences, interpretations (of variables), and satisfaction, all parameterized by the signature used;
- 2 the operational semantics of logic programming, supported by a notion of *logic-programming framework* that describes clauses and queries – two of the most important syntactic structures used to define and execute logic programs – as well as goal-directed rules that allow the integration of resolution;
- 3 the various constructions of logic programs through a notion of *logic-programming language* that defines programs as abstract objects characterized by signatures, sets of axioms (clauses) and classes of models, thus allowing us to uniformly accommodate a great variety of module systems.

Under this conceptual structure, we study a logic-independent version of Herbrand’s theorem, describe a general resolution-based procedure for computing solutions to queries, introduce a new form of service-oriented logic programming, and develop foundations of modularization.

HERBRAND'S FUNDAMENTAL THEOREM

The Fundamental Theorem of Herbrand [Her67] is a central result in proof theory that deals with the reduction of provability in first-order logic to provability in propositional logic. Its importance in the context of automated theorem proving was realized in the early 1960s, when, together with the theory of Horn-clause logic, it played a key role in establishing the mathematical foundations of logic programming [see e.g. Llo87]. In the traditional setting of relational first-order logic, Herbrand's theorem states that, for a set Γ of Horn clauses (an unstructured logic program), the answers to an existential query can be found simply by examining a term model – the least Herbrand model – instead of all the models that satisfy Γ . Thus, the theorem links the denotational and the operational semantics of a logic-programming language by reducing the satisfiability of a query with respect to a given logic program to the search of a suitable correct-answer substitution.

Over the last three decades, the original result has been extended to a variety of logical systems [e.g. GM87; Dia00; GMK02], culminating in [Dia04] with an investigation of Herbrand's theorem in an arbitrary institution. Thanks to its generality, the institution-based approach to Herbrand's theorem enabled the development of logic programming over a wide range of logical systems [see e.g. Găiip; and also Dia08]. All the same, certain institution-based forms of logic programming do not fit into the framework proposed in [Dia04]. In particular, the logic-programming semantics of services [TF13] is grounded on a family of logical systems for which the concept of variable cannot be faithfully captured by means of representable extensions of signatures, thus failing to meet one of the most basic assumptions of the institution-independent variant of the theorem. This led us to study Herbrand's fundamental theorem to an even greater extent in Chapters 3 and 4, in the general context of abstract logic-programming languages.

SOUND AND COMPLETE OPERATIONAL SEMANTICS

The second main contribution of our work is the development of a sound procedure for searching for solutions to queries. As in conventional logic programming, this is achieved through the computation of a series of intermediate results (namely computed substitutions) by means of resolution steps. The challenge here lies primarily in determining an appropriate formalization of the goal-directed rules that underlie our general form of resolution so as to capture both the first-order resolution of relational logic programming [see Rob65] and the paramodulation of equational logic pro-

gramming [see [RW83](#)]. We study the proposed abstract notion of resolution in [Chapter 3](#), where, in addition to soundness, we also discuss a minimal set of assumptions under which it can be shown to be complete.

SERVICE-ORIENTED LOGIC PROGRAMMING

Service-oriented computing is a modern computational paradigm that deals with the execution of programs over distributed information-processing infrastructures in which software applications can discover and bind dynamically, at run time, to services offered by providers. Whereas the paradigm has been effectively in use for a more than a decade in the form of Web services [[Alo+04](#)] or Grid computing [[FK04](#)], research into its formal foundations has lagged somewhat behind, partly because of the lack of understanding of (or agreement on) what is really new about the paradigm, especially in relation to distributed computing in general [see e.g. [Vog03](#)].

It is fair to say that significant advances have been made towards formalizing new kinds of distributed computation that have arisen around the notion of service (see, for example, the choreography models discussed in [[Su+07](#)]), notably through several variants of the π -calculus. However, service-oriented computing raises more profound challenges at the level of the structure of systems due to their ability to discover and bind dynamically, in a non-programmed manner, to other systems. As a result, the structure of the systems that we are now creating in the virtual space of computational networks is intrinsically dynamic, a phenomenon hitherto unknown. Formalisms such as the π -calculus do not address these structural properties of systems. This prevents us from fully controlling and developing trust in the systems that are now operating in cyberspace, and also from exploiting the power of the paradigm beyond the way it is currently deployed.

Towards that end, a great deal of work has focused on the algebraic structures that account for modularity [e.g. [FLB07](#); [FS07](#)] – referring to the way services are orchestrated as composite structures of components and how binding is performed through interaction protocols – and on the mechanisms through which discovery can be formalized in terms of logical specifications of required/provided services and constraint optimisation for service-level agreements [e.g. [FLB11](#); [FL13a](#)]. In [Chapter 5](#), we take this research further to address the operational aspects behind dynamic reconfiguration, i.e. the mechanisms through which applications discover and bind, at run time, to services. Our aim is to develop an abstract, foundational setting – independent of the specific technologies that are currently deployed, such as

SOAP for message-exchange protocols and UDDI for description, discovery, and integration – that combines both the denotational and the operational semantics of services. The most difficult and interesting task is to define an integrated algebraic framework that accounts for (a) logical specifications of services, (b) the way models of those specifications capture orchestrations of components that may depend on externally provided services, and (c) how the discovery of services and the binding of their orchestrations to client applications can be expressed in logical/algebraic terms.

The approach that we plan to develop to meet this challenge builds on the relational variant of logic programming. In a nutshell, the analogy between service-oriented computing and conventional logic programming that we propose to systematically examine in this thesis unfolds as follows:

- The Herbrand universe consists of service orchestrations that have no dependencies on external services – what we refer to as ground orchestrations.
- Variables and terms correspond to dependencies on external services to be discovered and to the actual services made available by orchestrations.
- Service clauses express properties of services required or provided by orchestrations, thus capturing the notion of service module described in [FLB11]. Their declarative semantics is that, when bound to the orchestrations of other service clauses that ensure the required properties, they deliver, through their orchestration, services that satisfy the specified properties.
- Service queries express properties of services that an application requires in order to fulfil its goal – what are described in [FLB11] as activity modules.
- Logic programs define service repositories as collections of service modules.
- Resolution accounts for service discovery by matching required properties with provided ones and the binding of required with provided services.

MODULARIZATION

The literature on modularization in logic programming is vast, and it has evolved primarily along two somewhat divergent lines of research. Whereas a number of proposals have focused on augmenting logic-programming languages with module systems through dedicated constructs for building programs as hierarchical combinations of components, preserving in this way the denotational and the operational semantics of the underlying formalisms [see e.g. O’K85; GM86; SW92], others have explored the possibility of extending the base language with new logical connectives in order to capture the operators needed for building and composing program modules [see e.g. Mil89; GMR92; and also the comprehensive survey BLM94].

Through our work, we aim to bring the two aforementioned lines of research into harmony by studying modularization over the model-theoretic framework for logic programming advanced in [Chapter 3](#), which combines ideas specific to each of the directions. On the one hand, by extending the concept of institution with appropriate notions of variable, substitution, clause, query, and goal-directed rule, the theory we propose accommodates logical systems whose expressive power goes beyond that of the original formalism of Horn-clause logic. On the other hand, inspired by recent developments in the theory of structured specifications [see [Dia12a](#)], it defines logic programs in an axiomatic manner, capturing in this way both representations of programs as plain sets of (definite) clauses [see [Llo87](#)] and module systems like those described, for example, in [[ST88a](#); [Boro2](#); [DȚ11](#)].

In the final chapter of this thesis, we investigate elementary modularization properties concerning the preservation and the reflection of solutions along morphisms of programs. From a computational point of view, these properties have a significant impact on the efficiency of searching for solutions to queries. In particular, the preservation of solutions gives us the possibility to search for solutions to queries in restricted contexts that correspond to subprograms or imported modules, and then translate these solutions back to the original setting, while the reflection property guarantees that all solutions can be obtained in this manner.

1.3 THE STRUCTURE OF THE THESIS

CHAPTER 3: [Section 3.1](#) reviews the foundations of conventional logic programming, with explicit emphasis on the abstraction of first-order variables and substitutions, and on the mappings induced by signature morphisms. [Section 3.2](#) introduces the basic notion of generalized substitution system and details the conditions that guarantee the invariance of the satisfaction of universally and existentially quantified sentences (defined over a given generalized substitution system) under change of notation. [Section 3.3](#) is dedicated to the development of abstract logic programming, encompassing the notions of logic-programming framework and language, as well as the first main results of the thesis.

CHAPTER 4: [Section 4.1](#) is devoted to the logical foundations of two other notable variants of logic programming, (many-sorted) first-order and higher-order equational logic programming. In [Section 4.2](#), we examine a class of substitution systems whose variables are defined through extensions of signatures (of a given institution), and whose substitutions correspond to

the institution-independent notion of substitution; based on this formalization, in [Section 4.3](#) we further investigate the translation of variables along signature morphisms and identify a set of sufficient conditions under which an institution can give rise to a generalized substitution system. Lastly, in [Section 4.4](#) we present a different perspective on the institution-independent versions of Herbrand’s fundamental theorem.

CHAPTER 5: In [Section 5.1](#), we present a new categorical model of service orchestrations, called orchestration scheme, that enables us to treat orchestrations as fully abstract entities required to satisfy only a few elementary properties. This framework is flexible enough to accommodate, for example, orchestrations in the form of program expressions, as considered in [\[Fia12\]](#), or as asynchronous relational networks similar to those defined in [\[FL13b\]](#). In our study, such schemes play an essential role in managing the inherent complexity of orchestrations whilst making available, at the same time, the fundamental building blocks of service-oriented logic programming. In [Section 5.2](#), we define a logical system of orchestration schemes over which we can express properties that can be further used to guide the interconnection of orchestrations. We prove that the resulting logic forms a generalized substitution system, thus providing the declarative semantics of our approach to service-oriented computing, as well as a definite mathematical foundation to the analogy between service-oriented computing and conventional logic programming. Building on these results, we show how (definite) clauses, queries, unification and resolution can be defined over the generalized substitution system of orchestration schemes, obtaining in this way the corresponding operational semantics of service-oriented computing.

CHAPTER 6: In [Section 6.1](#), we recall some of the most important specification-building operators from the literature, and explain how they can be used to define modular logic programs. This provides us with a structured model-oriented alternative to the original view of logic programs as plain sets of clauses. [Section 6.2](#) is devoted to the translation of queries and to the preservation and the reflection of solutions along signature morphisms. In [Section 6.3](#), we extend these results to computed answers (i.e. to the more operational notion of solution to a query), and discuss the soundness and completeness of resolution with respect to modularization.

1.4 RELEVANT PUBLICATIONS

Parts of this thesis have been presented at academic conferences, workshops, or meetings, or have been advanced in research papers, as follows:

- **Chapter 3** is a modified version of the paper [T̄Fipa], and has been presented at the 22nd meeting of the IFIP-TC1 Working Group on Foundations of System Specification – IFIP WG1.3.
- **Chapter 4** is based on the results presented in the papers [T̄Tu13; T̄Fipb], of which [T̄Fipb] has been presented at the 6th Conference on Algebra and Coalgebra in Computer Science – CALCO 2015.
- **Chapter 5** is a modified version of the paper [T̄Fipc]; a preliminary version of this work has been presented at the 5th Conference on Algebra and Coalgebra in Computer Science – CALCO 2013 – [T̄F13].
- **Chapter 6** is to some extent based on the results included in the paper [T̄Fipa], which have been further developed and presented at the 22nd International Workshop on Algebraic Development Techniques – WADT 2014.

TECHNICAL PRELIMINARIES

In this chapter, we review some of the most important category- and institution-theoretic concepts, notations, and terminology that support the general approach to logic programming developed in the subsequent sections of the present thesis. We focus particularly on comma categories, indexed categories, institutions and comorphisms of institutions, as well as on the presentation of institutions as functors into the category of rooms, all of which will be used extensively in our work.

2.1 CATEGORIES

We assume the reader is familiar with basic notions of category theory such as category, functor, and natural transformation. With a few exceptions, we use the terminology and the notations from [Mac98]. In this sense, we denote by $|C|$ the collection of objects of a category C , by $C(A, B)$ the collection of arrows from A to B , by $f \circ g$ the composition of arrows f and g in diagrammatic order, and by 1_A the identity arrow of an object A ; furthermore, we denote by $\tau \circ \sigma$ the ‘vertical’ composition of natural transformations τ and σ , and by $\tau \cdot \sigma$ (or sometimes by simple juxtaposition) their ‘horizontal’ composition.

A NOTE ON FOUNDATIONS. Most of the constructions described in this thesis rely on large or very large collections of objects, and thus some care must be taken with respect to the set-theoretic foundations of category theory. One may consider for instance the hierarchy of set universes discussed in [Mac98], in which every universe is closed under the usual set-theoretic constructions, contains the elements of its elements, and belongs to the next universe in the hierarchy. In this way, we obtain concepts such as category and functor for every level of the hierarchy, which means that we cannot define, for example, the category of all sets, but rather a category of sets for every universe. In terms of notation, given an arbitrary but fixed level of the hierarchy of set universes, we denote by Set its corresponding category of sets and functions, and by Cat its category of categories and functors, which belongs, of course, to a higher set-theoretic universe.

The most important category-theoretic notions for our work are those of comma category, Grothendieck construction, and generalized subfunctor. We only recall here a number of definitions and elementary properties, mainly for fixing the terminology and the notation to be used throughout the thesis. The interested reader can find more detailed presentations in canonical texts such as [Mac98; AHS09], as well as in works like [GB84; TBG91] on the applications of category theory to algebraic specification.

DEFINITION 2.1.1 (Comma category). Consider two functors $F_1: \mathbf{C}_1 \rightarrow \mathbb{K}$ and $F_2: \mathbf{C}_2 \rightarrow \mathbb{K}$ with the same codomain category \mathbb{K} . The *comma category* F_1 / F_2 is defined as follows:

- the *objects* are triples $\langle A_1, f, A_2 \rangle$, where A_1 is an object of \mathbf{C}_1 , A_2 is an object of \mathbf{C}_2 , and $f: F_1(A_1) \rightarrow F_2(A_2)$ is an arrow in \mathbb{K} ;
- the *arrows* $\langle A_1, f, A_2 \rangle \rightarrow \langle A'_1, f', A'_2 \rangle$ are pairs $\langle g_1, g_2 \rangle$, where $g_1: A_1 \rightarrow A'_1$ and $g_2: A_2 \rightarrow A'_2$ are arrows in \mathbf{C}_1 and \mathbf{C}_2 , such that $f \circ F_2(g_2) = F_1(g_1) \circ f'$;
- the *composition* of arrows is defined componentwise: for every pair of composable arrows $\langle g_1, g_2 \rangle$ and $\langle g'_1, g'_2 \rangle$, $\langle g_1, g_2 \rangle \circ \langle g'_1, g'_2 \rangle = \langle g_1 \circ g'_1, g_2 \circ g'_2 \rangle$.

$$\begin{array}{ccccc} F_1(A_1) & \xrightarrow{F_1(g_1)} & F_1(A'_1) & \xrightarrow{F_1(g'_1)} & F_1(A''_1) \\ f \downarrow & & \downarrow f' & & \downarrow f'' \\ F_2(A_2) & \xrightarrow{F_2(g_2)} & F_2(A'_2) & \xrightarrow{F_2(g'_2)} & F_2(A''_2) \end{array}$$

The comma category F_1 / F_2 is often denoted by \mathbf{C}_1 / F_2 , F_1 / \mathbf{C}_2 , or simply by $\mathbf{C}_1 / \mathbf{C}_2$ if F_1, F_2 , or both of them, respectively, correspond to inclusions of categories. A special case arises when F_1 is the constant functor with value $A \in |\mathbb{K}|$ and F_2 is the identity of \mathbb{K} . Then we denote the comma category F_1 / F_2 by A / \mathbb{K} – the *category of \mathbb{K} -objects under A* – and the forgetful functor $A / \mathbb{K} \rightarrow \mathbb{K}$ that maps every object $\langle A, f, A' \rangle$ of A / \mathbb{K} to A' by $|-|_A$.

FACT 2.1.2. Every arrow $h: A \rightarrow A'$ in a given category \mathbb{K} induces a left-composition functor $h / \mathbb{K}: A' / \mathbb{K} \rightarrow A / \mathbb{K}$ that maps every object $\langle A', f, B \rangle$ of A' / \mathbb{K} to $\langle A, h \circ f, B \rangle$ and every arrow $g: \langle A', f_1, B_1 \rangle \rightarrow \langle A', f_2, B_2 \rangle$ in A' / \mathbb{K} to $g: \langle A, h \circ f_1, B_1 \rangle \rightarrow \langle A, h \circ f_2, B_2 \rangle$.

DEFINITION 2.1.3 (Arrow category). For any category \mathbb{K} , the *category \mathbb{K}^\rightarrow of \mathbb{K} -arrows* is given by the comma category $1_{\mathbb{K}} / 1_{\mathbb{K}}$.

This means that, intuitively, the objects of the arrow category \mathbb{K}^\rightarrow are arrows $f: A_1 \rightarrow A_2$ in \mathbb{K} , and that the morphisms in \mathbb{K}^\rightarrow from $f: A_1 \rightarrow A_2$ to

$f': A'_1 \rightarrow A'_2$ correspond to commutative squares as depicted below.

$$\begin{array}{ccc} A_1 & \xrightarrow{g_1} & A'_1 \\ f \downarrow & & \downarrow f' \\ A_2 & \xrightarrow{g_2} & A'_2 \end{array}$$

We denote the two canonical projection functors $\mathbb{K}^\rightarrow \rightarrow \mathbb{K}$ that map the arrows $f: A_1 \rightarrow A_2$ in \mathbb{K} (i.e. the objects of \mathbb{K}^\rightarrow) to their *domain* A_1 and *codomain* A_2 by $\text{dom}: \mathbb{K}^\rightarrow \rightarrow \mathbb{K}$ and $\text{cod}: \mathbb{K}^\rightarrow \rightarrow \mathbb{K}$, respectively.

DEFINITION 2.1.4 (Indexed category). For any category \mathbb{I} of *indices*, an \mathbb{I} -*indexed category* is a functor $\mathbb{C}: \mathbb{I}^{\text{op}} \rightarrow \text{Cat}$.

DEFINITION 2.1.5 (Grothendieck construction). Any \mathbb{I} -indexed category $\mathbb{C}: \mathbb{I}^{\text{op}} \rightarrow \text{Cat}$ can be ‘flattened’ to a *Grothendieck category* \mathbb{C}^\sharp in which

- the *objects* are pairs $\langle i, A \rangle$, where i is an object of \mathbb{I} and A is an object of $\mathbb{C}(i)$,
- the *arrows* $\langle i, A \rangle \rightarrow \langle i', A' \rangle$ are pairs $\langle u, f \rangle$ such that $u: i \rightarrow i'$ is an arrow in \mathbb{I} and $f: A \rightarrow \mathbb{C}(u)(A')$ is an arrow in $\mathbb{C}(i)$, and
- the *composition* of arrows $\langle u, f \rangle$ and $\langle u', f' \rangle$ is defined as $\langle u \circledast u', f \circledast \mathbb{C}(u)(f') \rangle$.

$$\begin{array}{ccccc} & \overset{u}{\curvearrowright} & & \overset{u'}{\curvearrowright} & \\ \langle i, A \rangle & & \langle i', A' \rangle & & \langle i'', A'' \rangle \\ & \downarrow f & & \downarrow f' & \\ & \mathbb{C}(u)(A') & \xrightarrow{\mathbb{C}(u)(f')} & \mathbb{C}(u \circledast u')(A'') & \end{array}$$

FACT 2.1.6. Every *indexed functor* F between \mathbb{I} -indexed categories \mathbb{C} and $\mathbb{D}: \mathbb{I}^{\text{op}} \rightarrow \text{Cat}$, that is every natural transformation $F: \mathbb{C} \Rightarrow \mathbb{D}$, determines a functor $F^\sharp: \mathbb{C}^\sharp \rightarrow \mathbb{D}^\sharp$ that maps every object $\langle i, A \rangle$ of the Grothendieck category \mathbb{C}^\sharp to $\langle i, F_i(A) \rangle$ and every arrow $\langle u, f \rangle: \langle i, A \rangle \rightarrow \langle i', A' \rangle$ in \mathbb{C}^\sharp to $\langle u, F_i(f) \rangle$. We obtain in this way a ‘flattening’ functor $(_)^\sharp$ from the category $[\mathbb{I}^{\text{op}} \rightarrow \text{Cat}]$ of \mathbb{I} -indexed categories to Cat .

DEFINITION 2.1.7 (Category of functors). For any category \mathbb{K} , the category $[_ \rightarrow \mathbb{K}]^\sharp$ of *functors into* \mathbb{K} is the Grothendieck category defined by the functor $[_ \rightarrow \mathbb{K}]: \text{Cat}^{\text{op}} \rightarrow \text{Cat}$ that maps every category \mathbb{C} to the functor category $[\mathbb{C} \rightarrow \mathbb{K}]$ (with natural transformations as arrows) and any functor $U: \mathbb{C} \rightarrow \mathbb{C}'$ to the left-composition functor $U_-: [\mathbb{C}' \rightarrow \mathbb{K}] \rightarrow [\mathbb{C} \rightarrow \mathbb{K}]$.

This means that the objects of $[_ \rightarrow \mathbb{K}]^\sharp$ are, in essence, functors $F: \mathbb{C} \rightarrow \mathbb{K}$ into the category \mathbb{K} , and that the morphisms between two such functors

$F: \mathbb{C} \rightarrow \mathbb{K}$ to $F': \mathbb{C}' \rightarrow \mathbb{K}$ are pairs $\langle U, \tau \rangle$, where U is a functor $\mathbb{C} \rightarrow \mathbb{C}'$ and τ is a natural transformation $F \Rightarrow U \circ F'$. Moreover, the composition of morphisms in $[_ \rightarrow \mathbb{K}]^\sharp$ is defined by $\langle U, \tau \rangle \circ \langle U', \tau' \rangle = \langle U \circ U', \tau \circ (U' \cdot \tau') \rangle$.

FACT 2.1.8. Functors $G: \mathbb{K} \rightarrow \mathbb{K}'$ determine appropriate natural transformations $[G]$ between $[_ \rightarrow \mathbb{K}]$ and $[_ \rightarrow \mathbb{K}']$; their components $[G]_{\mathbb{C}}$, for categories $\mathbb{C} \in |\text{Cat}^{\text{op}}|$, are the obvious right-composition functors from $[\mathbb{C} \rightarrow \mathbb{K}]$ to $[\mathbb{C} \rightarrow \mathbb{K}']$. Hence, by [Fact 2.1.6](#), every functor G between categories \mathbb{K} and \mathbb{K}' induces a functor $[G]^\sharp: [_ \rightarrow \mathbb{K}]^\sharp \rightarrow [_ \rightarrow \mathbb{K}']^\sharp$.

DEFINITION 2.1.9 (Subfunctor). For any two functors $F, G: \mathbb{C} \rightarrow \text{Set}$, F is said to be a *subfunctor* of G (denoted $F \subseteq G$) if there exists a natural transformation $F \Rightarrow G$ whose components are all set-theoretic inclusions.

Therefore, F is a subfunctor of G when for every object $A \in |\mathbb{C}|$, $F(A)$ is a subset of $G(A)$, and for every arrow $f: A \rightarrow A'$ in \mathbb{C} , $F(f)$ is the domain-codomain restriction of $G(f)$.

DEFINITION 2.1.10 (Generalized subfunctor). For any two functors $F, G: \mathbb{I} \rightarrow [_ \rightarrow \text{Set}]^\sharp$, F is said to be a *generalized subfunctor* of G (denoted $F \subseteq G$) when $F(i)$ is a natural subfunctor of $G(i)$ for every $i \in |\mathbb{I}|$.

This means that, for every object $i \in |\mathbb{I}|$, $F(i)$ is a subfunctor of $G(i)$ and, for every arrow $u: i \rightarrow i'$ in \mathbb{I} , the $[_ \rightarrow \text{Set}]^\sharp$ -morphisms $F(u)$ and $G(u)$ make the following square commute.

$$\begin{array}{ccc}
 i & F(i): \mathbb{C}_i \rightarrow \text{Set} & \xrightarrow{F(i) \subseteq G(i)} & G(i): \mathbb{C}_i \rightarrow \text{Set} \\
 u \downarrow & \downarrow F(u) & & \downarrow G(u) \\
 i' & F(i'): \mathbb{C}_{i'} \rightarrow \text{Set} & \xrightarrow{F(i') \subseteq G(i')} & G(i'): \mathbb{C}_{i'} \rightarrow \text{Set}
 \end{array}$$

2.2 INSTITUTIONS

The notion of institution emerged within the theory of algebraic specification from the general concept of language [see [BG79](#)]. It was introduced by Goguen and Burstall in [[GB92](#)] with the aim of formalizing the intuitive notion of logical system as a balanced interaction between its syntax and its semantics. The theory of institutions provides in this way a rigorous model-theoretic abstraction of logics, and thus it can be used to provide foundations for both specification and programming languages.

In what follows, we recall the basic notions and notations of institutions. Intuitively, an institution consists of a collection (category) of signatures, each of which determines (through dedicated functors) a set of sentences

and a collection of models, as well as a satisfaction relation between models and sentences that is assumed to be invariant under change of signature.

DEFINITION 2.2.1 (Institution). An *institution* \mathcal{J} consists of

- a category $\text{Sig}^{\mathcal{J}}$ of *signatures* and *signature morphisms*,
- a *sentence functor* $\text{Sen}^{\mathcal{J}}: \text{Sig}^{\mathcal{J}} \rightarrow \text{Set}$ defining, for every signature Σ , the set $\text{Sen}^{\mathcal{J}}(\Sigma)$ of Σ -sentences and, for every signature morphism $\varphi: \Sigma \rightarrow \Sigma'$, the *sentence-translation map* $\text{Sen}^{\mathcal{J}}(\varphi): \text{Sen}^{\mathcal{J}}(\Sigma) \rightarrow \text{Sen}^{\mathcal{J}}(\Sigma')$,
- a *model functor* $\text{Mod}^{\mathcal{J}}: (\text{Sig}^{\mathcal{J}})^{\text{op}} \rightarrow \text{Cat}$ defining, for every signature Σ , the category $\text{Mod}^{\mathcal{J}}(\Sigma)$ of Σ -models and Σ -(model) homomorphisms and, for every morphism $\varphi: \Sigma \rightarrow \Sigma'$, the *reduct functor* $\text{Mod}^{\mathcal{J}}(\varphi): \text{Mod}^{\mathcal{J}}(\Sigma') \rightarrow \text{Mod}^{\mathcal{J}}(\Sigma)$,
- a family of *satisfaction relations* $\vDash_{\Sigma}^{\mathcal{J}} \subseteq |\text{Mod}^{\mathcal{J}}(\Sigma)| \times \text{Sen}^{\mathcal{J}}(\Sigma)$ determining, for every signature Σ , the satisfaction of Σ -sentences by Σ -models,

such that, for any signature morphisms $\varphi: \Sigma \rightarrow \Sigma'$, the translation of sentences $\text{Sen}^{\mathcal{J}}(\varphi)$ and the reduction of models $\text{Mod}^{\mathcal{J}}(\varphi)$ preserve the satisfaction relation: for every Σ' -model M' and Σ -sentence ρ ,

$$M' \vDash_{\Sigma'}^{\mathcal{J}} \text{Sen}^{\mathcal{J}}(\varphi)(\rho) \quad \text{if and only if} \quad \text{Mod}^{\mathcal{J}}(\varphi)(M') \vDash_{\Sigma}^{\mathcal{J}} \rho.$$

This property is usually called the *satisfaction condition* for \mathcal{J} .

When there is no danger of confusion, we may omit the superscripts or subscripts from the notations introduced above for the components of institutions. For instance, when the institution \mathcal{J} and the signature Σ can be easily inferred, we will often denote the satisfaction relation $\vDash_{\Sigma}^{\mathcal{J}}$ simply by \vDash . In addition, we will frequently denote the sentence translation $\text{Sen}^{\mathcal{J}}(\varphi)$ by $\varphi(_)$ and the reduct functor $\text{Mod}^{\mathcal{J}}(\varphi)$ by $_ \upharpoonright_{\varphi}$; and we will say that M is the φ -reduct of M' and that M' is a φ -expansion of M whenever $M = M' \upharpoonright_{\varphi}$. Finally, we will use the fact that the satisfaction relation extends in a natural way from sentences to sets of sentences, and we will say that a Σ -sentence ρ is a *semantic consequence* of a set of Σ -sentences E , denoted $E \vDash_{\Sigma}^{\mathcal{J}} \rho$, when every Σ -model that satisfies every sentence in E satisfies ρ as well.

Note that, for any signature Σ of an institution \mathcal{J} , the satisfaction relation $\vDash_{\Sigma}^{\mathcal{J}}$ can be identified with a Boolean-valued function $\vDash_{\Sigma}^{\mathcal{J}}: \text{Sen}^{\mathcal{J}}(\Sigma) \rightarrow [|\text{Mod}^{\mathcal{J}}(\Sigma)| \rightarrow 2]$ that determines whether any given Σ -sentence is satisfied by any given Σ -model. In view of this observation, the satisfaction condition for \mathcal{J} can be captured categorically by defining the satisfaction relations $\vDash_{\Sigma}^{\mathcal{J}}$ as components of a natural transformation

$$\vDash^{\mathcal{J}}: \text{Sen}^{\mathcal{J}} \Rightarrow [|\text{Mod}^{\mathcal{J}}(_)| \rightarrow 2]$$

from $\text{Sen}^{\mathcal{J}}$ to the functor $[|\text{Mod}^{\mathcal{J}}(_)| \rightarrow 2]$ that maps every signature $\Sigma \in |\text{Sig}^{\mathcal{J}}|$ to the collection of all functions from $|\text{Mod}^{\mathcal{J}}(\Sigma)|$ to the two-element set 2.

$$\begin{array}{ccc}
 \Sigma & \text{Sen}^{\mathcal{J}}(\Sigma) & \xrightarrow{\varepsilon_{\Sigma}^{\mathcal{J}}} [|\text{Mod}^{\mathcal{J}}(\Sigma)| \rightarrow 2] \\
 \varphi \downarrow & \text{Sen}^{\mathcal{J}}(\varphi) \downarrow & \downarrow \text{Mod}^{\mathcal{J}}(\varphi)_{\#} \\
 \Sigma' & \text{Sen}^{\mathcal{J}}(\Sigma') & \xrightarrow{\varepsilon_{\Sigma'}^{\mathcal{J}}} [|\text{Mod}^{\mathcal{J}}(\Sigma')| \rightarrow 2]
 \end{array}$$

A large number of logical systems have been formalized as institutions. Some examples related to the study of logic programming include (many-sorted) first-order logic [see GB92], order-sorted Horn-clause logic with equality [see GM87], category-based equational logic [see Dia95], higher-order logic [see MTW87], hidden algebra [see GMK02], constraint logic [see Dia00], and the rather recent logic of asynchronous relational networks described in [TF13]. Many others are thoroughly discussed in monographs on institutions and algebraic specifications such as [Dia08; ST11].

The present thesis is mainly concerned with institutions of substitutions, that is with ‘local’ institutions that are usually defined over a given signature of a logical system such as first-order or higher-order logic, whose signatures are abstract representations of (sets of) variables, and whose signature morphisms correspond to substitutions. These structures provide convenient abstractions of the basic elements involved in defining the denotational and the operational semantics of logic programs. A detailed presentation of the institutions of first-order substitutions is given in Section 3.1.1.

Changing signatures (by means of signature morphisms) yields appropriate maps between their corresponding institutions of substitutions, formalized as *institution comorphisms* [GR02]. These capture the intuitive notion of embedding simpler logical systems into more complex ones, and are dual to the original concept of *institution morphism* [GB92], which corresponds to structure-forgetting relationships between logical systems. Institution comorphisms were originally discussed in [Mes89] under the name of *plain maps*, and in [Tar95] under the name of *institution representations*.

DEFINITION 2.2.2 (Comorphism of institutions). For any two institutions $\mathcal{J} = \langle \text{Sig}, \text{Sen}, \text{Mod}, \varepsilon \rangle$ and $\mathcal{J}' = \langle \text{Sig}', \text{Sen}', \text{Mod}', \varepsilon' \rangle$, a *comorphism* $\mathcal{J} \rightarrow \mathcal{J}'$ is a triple $\langle \Phi, \alpha, \beta \rangle$ for which

- Φ is a *signature functor* $\text{Sig} \rightarrow \text{Sig}'$,
- α is a natural transformation $\text{Sen} \Rightarrow \Phi \circ \text{Sen}'$, and
- β is a natural transformation $\Phi^{\text{op}} \circ \text{Mod}' \Rightarrow \text{Mod}$,

such that the following property – the *satisfaction condition* for $\langle \Phi, \alpha, \beta \rangle$ – holds for every \mathcal{J} -signature Σ , $\Phi(\Sigma)$ -model M' , and Σ -sentence ρ :

$$M' \vDash'_{\Phi(\Sigma)} \alpha_{\Sigma}(\rho) \quad \text{if and only if} \quad \beta_{\Sigma}(M') \vDash_{\Sigma} \rho.$$

Institution comorphisms compose in a natural, componentwise manner. Their composition is associative and has identities, thus providing a category Collns of institutions and institution comorphisms.

2.3 INSTITUTIONS AS FUNCTORS

Institutions can also be defined as functors into a category of local satisfaction systems called *rooms* or *twisted relations* [see GB92]. This alternative characterization has proved to be convenient for various studies on generalized institutions [see GB86], heterogeneous logical systems [see Mos02; Mos06], and logic translations [see MDT09].

DEFINITION 2.3.1 (Rooms and corridors). The category $\mathbb{R}\text{oom}$ of *rooms* and *corridors* is defined as follows:

The *objects* are *Boolean rooms*, that is triples $\langle S, \mathbb{M}, \vDash \rangle$ consisting of

- a set S of *sentences*,
- a category \mathbb{M} of *models*, and
- a *satisfaction relation* $\vDash \subseteq |\mathbb{M}| \times S$.

The *morphisms* – *corridors* – $\langle S, \mathbb{M}, \vDash \rangle \rightarrow \langle S', \mathbb{M}', \vDash' \rangle$ are pairs $\langle \alpha, \beta \rangle$, where

- α is a *sentence-translation function* $S \rightarrow S'$ and
- β is a *model-reduction functor* $\mathbb{M}' \rightarrow \mathbb{M}$,

such that the following condition holds for all $M' \in |\mathbb{M}'|$ and $\rho \in S$:

$$M' \vDash' \alpha(\rho) \quad \text{if and only if} \quad \beta(M') \vDash \rho.$$

The *composition* of morphisms is defined componentwise.

The ternary structure of rooms induces appropriate projections Sen and Mod^1 into the categories of sets and categories, respectively, as well as a natural transformation \vDash that captures the invariance of the satisfaction of sentences by models with respect to the change of rooms:

¹ For simplicity, we overload the use of Sen , Mod , and \vDash . To resolve the potential confusion between the projections of $\mathbb{R}\text{oom}$ and the sentence functor, model functor, and satisfaction relations of an institution \mathcal{J} , we usually denote the latter by $\text{Sen}^{\mathcal{J}}$, $\text{Mod}^{\mathcal{J}}$, and $\vDash^{\mathcal{J}}$, respectively.

- $\text{Sen}: \mathbb{R}\text{oom} \rightarrow \text{Set}$ and $\text{Mod}: \mathbb{R}\text{oom}^{\text{op}} \rightarrow \text{Cat}$ are the functors that map every room $\langle S, \mathbb{M}, \varepsilon \rangle$ to its underlying set of sentences S and category of models \mathbb{M} respectively, and
- ε is the natural transformation $\text{Sen} \Rightarrow [|\text{Mod}(_)| \rightarrow 2]$ whose components are given by the satisfaction relations of the considered rooms.

Hence, $\mathbb{R}\text{oom}$ can also be understood as the comma category $\text{Set} / [|_ | \rightarrow 2]$.

FACT 2.3.2. The categories $\mathbb{R}\text{oom}$ and $\text{Set} / [|_ | \rightarrow 2]$ are isomorphic.

The observation above allows us to identify every institution \mathcal{J} having Sig as the category of signatures with the functor $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$ that maps

- every signature Σ to $\langle \text{Sen}^{\mathcal{J}}(\Sigma), \text{Mod}^{\mathcal{J}}(\Sigma), \varepsilon_{\Sigma}^{\mathcal{J}} \rangle$ and
- every signature morphism φ to $\langle \text{Sen}^{\mathcal{J}}(\varphi), \text{Mod}^{\mathcal{J}}(\varphi) \rangle$.

Conversely, every functor $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$ describes an institution whose category of signatures, sentence functor, model functor, and family of satisfaction relations are given by $\text{Sig}, \mathcal{J} \circ \text{Sen}, \mathcal{J}^{\text{op}} \circ \text{Mod}$, and $\mathcal{J} \cdot \varepsilon$ respectively.

This one-to-one correspondence between institutions and functors into $\mathbb{R}\text{oom}$ can be easily extended to maps of institutions: every comorphism of institutions $\langle \Phi, \alpha, \beta \rangle: \mathcal{J} \rightarrow \mathcal{J}'$ can be regarded as an arrow $\langle \Phi, \tau \rangle$ between the functors $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$ and $\mathcal{J}': \text{Sig}' \rightarrow \mathbb{R}\text{oom}$, where, for every \mathcal{J} -signature Σ , τ_{Σ} is the corridor $\langle \alpha_{\Sigma}, \beta_{\Sigma} \rangle$; in the opposite direction, α and β can be derived from the ‘horizontal’ compositions $\tau \cdot \text{Sen}$ and $\tau^{\text{op}} \cdot \text{Mod}$ respectively.

FACT 2.3.3. The categories collns and $[_ \rightarrow \mathbb{R}\text{oom}]^{\#}$ are isomorphic.

INTRODUCING LOGIC-INDEPENDENT LOGIC PROGRAMMING

Following the work of Goguen and Burstall, in this chapter we propose a logic-independent approach to logic programming through which the paradigm as we know it for Horn-clause logic can be systematically examined for other formalisms. For this purpose, we introduce an abstract concept of generalized substitution system that extends institutions by allowing for direct representations of variables and substitutions, providing in this way support for the development of both the denotational and the operational semantics of logic programming. In this framework, we study a logic-independent variant of Herbrand's theorem and describe a general resolution-based procedure for computing solutions to queries. We prove that this procedure is sound and that, moreover, under additional hypotheses that reflect faithfully properties of actual logic-programming languages, it is also complete.

3.1 THE LOGICAL SYSTEM OF REFERENCE

Our first step towards a presentation of the foundations of logic programming that does not depend on any concrete logical system consists in determining an appropriate benchmark institution that will enable us to relate to concepts and results specific to conventional logic programming [see generally Llo87]. To this end, we recall that logic programming has been traditionally studied in the Horn substitution of $\underline{\text{FOL}}_{\neq}^1$ – the single-sorted variant of first-order logic without equality.

SIGNATURES. A $\underline{\text{FOL}}_{\neq}^1$ -signature is a pair $\langle F, P \rangle$ where F and P are ω -indexed families $(F_n)_{n \in \omega}$ and $(P_n)_{n \in \omega}$ of disjoint sets of operation and relation symbols respectively. The indices are usually called arities and qualify the operation and relation symbols; in this sense, we denote an operation or relation symbol ς of arity n by $\varsigma : n$. However, when there is no risk of confusion, we may drop the additional qualification and denote $\varsigma : n$ simply by ς .

The *signature morphisms* $\varphi: \langle F, P \rangle \rightarrow \langle F', P' \rangle$ reflect the structure of signatures and consist of families φ^{op} and φ^{rel} of functions $\varphi_n^{\text{op}}: F_n \rightarrow F'_n$ and $\varphi_n^{\text{rel}}: P_n \rightarrow P'_n$, for $n \in \omega$, between the corresponding sets of operation and relation symbols, respectively.

SENTENCES. The sentences are ordinary first-order sentences built over relational atoms: for any FOL_{\neq}^1 -signature $\langle F, P \rangle$, the set T_F of F -terms is the least set such that $\sigma(t_1, \dots, t_n) \in T_F$ for every $\sigma \in F_n$ and $t_1, \dots, t_n \in T_F$; the set of $\langle F, P \rangle$ -sentences can then be defined as the least set that contains the *relational atoms* $\pi(t_1, \dots, t_n)$, where $\pi \in P_n$ and $t_1, \dots, t_n \in T_F$, and is closed under Boolean connectives¹ and quantification over sets of first-order variables. It should be noted that $\langle F, P \rangle$ -variables are pairs (x, F_0) , often denoted simply by their name, x , and that the sets of $\langle F, P \rangle$ -variables are in fact collections of variables such that different variables have different names. Also, every set of $\langle F, P \rangle$ -variables X determines an extended signature $\langle F \cup X, P \rangle$, which can be obtained from $\langle F, P \rangle$ by adding the variables of X as new constants.

The *translation of sentences* along a signature morphism $\varphi: \langle F, P \rangle \rightarrow \langle F', P' \rangle$ is defined inductively on the structure of sentences by replacing the symbols of $\langle F, P \rangle$ according to φ . To be more precise, φ determines a function $\varphi^{\text{tm}}: T_F \rightarrow T_{F'}$ given by $\sigma(t_1, \dots, t_n) \mapsto \varphi_n^{\text{op}}(\sigma)(\varphi^{\text{tm}}(t_1), \dots, \varphi^{\text{tm}}(t_n))$, which allows us to define the translation of relational atoms $\pi(t_1, \dots, t_n)$ – corresponding to the signature $\langle F, P \rangle$ – as $\varphi_n^{\text{rel}}(\pi)(\varphi^{\text{tm}}(t_1), \dots, \varphi^{\text{tm}}(t_n))$. This definition can be straightforwardly extended to more complex sentences. For instance, the translation of universally quantified $\langle F, P \rangle$ -sentences is given by $\forall X \cdot \rho \mapsto \forall X^\varphi \cdot \varphi^X(\rho)$, where X^φ is the set of $\langle F', P' \rangle$ -variables $\{(x, F'_0) \mid (x, F_0) \in X\}$ and $\varphi^X: \langle F \cup X, P \rangle \rightarrow \langle F' \cup X^\varphi, P' \rangle$ is the canonical extension of φ such that $(\varphi^X)_0^{\text{op}}(x, F_0) = (x, F'_0)$ for every $(x, F_0) \in X$.

$$\begin{array}{ccc} \langle F, P \rangle & \xrightarrow{\varphi} & \langle F', P' \rangle \\ \subseteq \downarrow & & \downarrow \subseteq \\ \langle F \cup X, P \rangle & \xrightarrow{\varphi^X} & \langle F' \cup X^\varphi, P' \rangle \end{array}$$

MODELS. Given a FOL_{\neq}^1 -signature $\langle F, P \rangle$, an $\langle F, P \rangle$ -model M consists of

- a set $|M|$, called the *carrier set* of M , together with
- a function $M_\sigma: |M|^n \rightarrow |M|$ for each operation symbol $\sigma \in F_n$ and
- a subset $M_\pi \subseteq |M|^n$ for each relation symbol $\pi \in P_n$.

¹ For convenience, we assume that disjunctions, denoted $\vee E$, and conjunctions, denoted $\wedge E$, are defined over arbitrary finite sets of sentences E , and we abbreviate $\wedge\{\rho_1, \rho_2\}$ as $\rho_1 \wedge \rho_2$ and $\wedge \emptyset$ as *true*. When there is no danger of confusion, we may also abbreviate $\wedge\{\rho\}$ as ρ .

Homomorphisms $h: M_1 \rightarrow M_2$ are functions $h: |M_1| \rightarrow |M_2|$ between the carrier sets of M_1 and M_2 satisfying the following two properties:

- 1 Preservation of operations: $h(M_{1,\sigma}(m_1, \dots, m_n)) = M_{2,\sigma}(h(m_1), \dots, h(m_n))$ for all operation symbols $\sigma \in F_n$ and all arguments $m_1, \dots, m_n \in |M_1|$.
- 2 Preservation of relations: $h(M_{1,\pi}) \subseteq M_{2,\pi}$ for all relation symbols $\pi \in P_n$.

Concerning *model reducts*, for any signature morphism $\varphi: \langle F, P \rangle \rightarrow \langle F', P' \rangle$ and $\langle F', P' \rangle$ -model M' , the φ -reduct $M' \upharpoonright_\varphi$ is defined as the $\langle F, P \rangle$ -model M with the same carrier set as M' and with the interpretation of operation and relation symbols given by $M_\zeta = M'_{\varphi(\zeta)}$ for every symbol ζ in $\langle F, P \rangle$.

THE SATISFACTION RELATION. The *satisfaction* between models and sentences is the usual Tarskian satisfaction defined inductively on the structure of sentences and based on the evaluation of terms in models. For example, an $\langle F, P \rangle$ -model M satisfies a universally quantified $\langle F, P \rangle$ -sentence $\forall X \cdot \rho$ if all of its expansions along the signature inclusion $\langle F, P \rangle \subseteq \langle F \cup X, P \rangle$ satisfy ρ ; thus, $M \models_{\langle F, P \rangle} \forall X \cdot \rho$ if $N \models_{\langle F \cup X, P \rangle} \rho$ for all models N of $\langle F \cup X, P \rangle$ such that $N \upharpoonright_{\langle F, P \rangle} = M$, that is for all valuations in M of the variables in X .

The approach that we follow in the present inquiry on the foundations of logic programming relies on abstract concepts of universally and existentially quantified sentences, whose definitions are independent of the logical system of choice. For this reason, our description of conventional logic programming is not based on the institution $\underline{\text{FOL}}_\neq^1$ discussed above, but on its quantifier-free fragment (with sentences built from atoms by repeated applications of Boolean connectives), which we denote by $\text{QF-}\underline{\text{FOL}}_\neq^1$.

3.1.1 INSTITUTIONS OF SUBSTITUTIONS

THE CATEGORY OF SUBSTITUTIONS. Let us fix a $\text{QF-}\underline{\text{FOL}}_\neq^1$ -signature $\langle F, P \rangle$, for example the following signature of natural numbers with addition (specified as a ternary predicate): $F_{\text{NAT}} = \{\emptyset: 0, s_-: 1\}$, $P_{\text{NAT}} = \{\text{add}: 3\}$.

In this setting, variables (or, more precisely, sets of variables) can be seen as signatures of a specialized logical system. A *signature of variables* is just a set X of $\langle F, P \rangle$ -variables with distinct names, which, by definition, shares no elements with the set F_0 of constant-operation symbols. The syntactic entities over a given signature of variables X are defined as the corresponding $\text{QF-}\underline{\text{FOL}}_\neq^1$ -expressions that can be formed based on the operation and relation symbols of $\langle F, P \rangle$ and on the variables of X considered as new constants. For example, the terms over X , or with variables from X , can be informally defined in an inductive manner as follows:

- every variable of X is a term over X and
- for every arity n , if σ is an operation symbol in F_n and t_1, \dots, t_n are terms over X , then $\sigma(t_1, \dots, t_n)$ is a term over X as well.

Substitutions provide syntactic transformations on expressions that are defined over sets of variables: given two signatures of variables X and Y , a *substitution* $\psi: X \rightarrow Y$ is a map $\psi: X \rightarrow T_{F \cup Y}$ that associates a term over Y with every variable of X . Note that any substitution $\psi: X \rightarrow Y$ can be canonically extended to a function $\psi^{\text{tm}}: T_{F \cup X} \rightarrow T_{F \cup Y}$ defined by $x \mapsto \psi(x)$ for variables, and by $\sigma(t_1, \dots, t_n) \mapsto \sigma(\psi^{\text{tm}}(t_1), \dots, \psi^{\text{tm}}(t_n))$ for compound terms. This allows us to define the composition of substitutions $\psi: X \rightarrow Y$ and $\theta: Y \rightarrow Z$ as the map $\psi \circ \theta^{\text{tm}}: X \rightarrow T_{F \cup Z}$. It is easy to see that $\psi \circ \theta$ induces the term translation $\psi^{\text{tm}} \circ \theta^{\text{tm}}$, which entails that the composition of substitutions is associative. Moreover, it satisfies the identity laws by taking, for every signature of variables X , the identity 1_X as the function that encodes the variables of X as terms over X . Hence, the signatures of $\langle F, P \rangle$ -variables together with their corresponding substitutions form a category $\text{Subst}_{\langle F, P \rangle}$.

SENTENCES, MODELS, AND THE SATISFACTION RELATION. Signatures of variables do not inherit only the terms of the extended QF-FOL_{\neq}^1 -signatures, but also their sentences, models, and the satisfaction relation between them. For every signature of variables X we define

- the set of sentences $\text{Sen}_{\langle F, P \rangle}(X)$ as $\text{Sen}(F \cup X, P)$,
- the category of models $\text{Mod}_{\langle F, P \rangle}(X)$ as $\text{Mod}(F \cup X, P)$, and
- the satisfaction relation $\models_{\langle F, P \rangle, X}$ as $\models_{\langle F \cup X, P \rangle}$.

With respect to substitutions, notice that, for every two signatures of variables X and Y , every substitution $\psi: X \rightarrow Y$ determines

- a sentence-translation map $\text{Sen}_{\langle F, P \rangle}(\psi): \text{Sen}_{\langle F, P \rangle}(X) \rightarrow \text{Sen}_{\langle F, P \rangle}(Y)$ defined on atomic sentences by $\pi(t_1, \dots, t_n) \mapsto \pi(\psi^{\text{tm}}(t_1), \dots, \psi^{\text{tm}}(t_n))$, and
- a model-reduct functor $\text{Mod}_{\langle F, P \rangle}(\psi): \text{Mod}_{\langle F, P \rangle}(Y) \rightarrow \text{Mod}_{\langle F, P \rangle}(X)$ that maps every Y -model N to the X -model $N \upharpoonright_{\psi}$ given by $|N \upharpoonright_{\psi}| = |N|$, $(N \upharpoonright_{\psi})_{\zeta} = N_{\zeta}$ for each symbol ζ of $\langle F, P \rangle$, and $(N \upharpoonright_{\psi})_x = N_{\psi(x)}$ (the evaluation of the term $\psi(x)$ in N) for each variable x in X .

We have thus defined the four components of an institution of $\langle F, P \rangle$ -substitutions: the category $\text{Subst}_{\langle F, P \rangle}$ of signatures (of variables) and signature morphisms (substitutions), the sentence functor $\text{Sen}_{\langle F, P \rangle}: \text{Subst}_{\langle F, P \rangle} \rightarrow \text{Set}$, the model functor $\text{Mod}_{\langle F, P \rangle}: \text{Subst}_{\langle F, P \rangle}^{\text{op}} \rightarrow \text{Cat}$, and the family of satisfaction

relations $(\vDash_{\langle F,P \rangle, X})_{X \in |\text{Subst}_{\langle F,P \rangle}|}$. Our construction is completed by the following result, originally discussed in [Diao4], which states that satisfaction is invariant under substitution of variables.

PROPOSITION 3.1.1. *For every first-order substitution $\psi: X \rightarrow Y$, every model N of Y , and every sentence ρ with variables from X ,*

$$N \vDash_{\langle F,P \rangle, Y} \text{Sen}_{\langle F,P \rangle}(\psi)(\rho) \quad \text{if and only if} \quad \text{Mod}_{\langle F,P \rangle}(\psi)(N) \vDash_{\langle F,P \rangle, X} \rho. \quad \square$$

COROLLARY 3.1.2. *For every QF-FOL_{\neq}^1 -signature $\langle F, P \rangle$, the structure*

$$(\text{QF-FOL}_{\neq}^1)_{\langle F,P \rangle} = \langle \text{Subst}_{\langle F,P \rangle}, \text{Sen}_{\langle F,P \rangle}, \text{Mod}_{\langle F,P \rangle}, \vDash_{\langle F,P \rangle} \rangle$$

defines an institution – the institution of $\langle F, P \rangle$ -substitutions. \square

The institutions of substitutions are not considered in isolation; instead, they are linked by various institution comorphisms induced by morphisms between their underlying first-order signatures.

PROPOSITION 3.1.3. *Every QF-FOL_{\neq}^1 -signature-morphism $\varphi: \langle F, P \rangle \rightarrow \langle F', P' \rangle$ determines a comorphism $\langle \Psi_\varphi, \alpha_\varphi, \beta_\varphi \rangle$ between the institutions of $\langle F, P \rangle$ - and $\langle F', P' \rangle$ -substitutions, where*

- $\Psi_\varphi(X) = X^\varphi$ and $\Psi_\varphi(\psi)(x, F'_0) = (\varphi^Y)^{\text{tm}}(\psi(x, F_0))$,
- $\alpha_{\varphi, X} = \text{Sen}(\varphi^X)$, and
- $\beta_{\varphi, X} = \text{Mod}(\varphi^X)$,

for each signature of $\langle F, P \rangle$ -variables X , $\langle F, P \rangle$ -substitution $\psi: X \rightarrow Y$, and variable $(x, F'_0) \in X^\varphi$.

PROOF. We begin by proving that Ψ_φ is a functor $\text{Subst}_{\langle F,P \rangle} \rightarrow \text{Subst}_{\langle F',P' \rangle}$. Following an inductive argument on the structure of terms, we can easily infer that, for every $\langle F, P \rangle$ -substitution $\psi: X \rightarrow Y$, the translations $(\varphi^X)^{\text{tm}} \circ \Psi_\varphi(\psi)^{\text{tm}}$ and $\psi^{\text{tm}} \circ (\varphi^Y)^{\text{tm}}$ (of F -terms over X into F' -terms over Y^φ) are equal. Based on this property, for any pair of composable $\langle F, P \rangle$ -substitutions $\psi: X \rightarrow Y$ and $\theta: Y \rightarrow Z$, we obtain the following chain of equalities:

$$\begin{aligned} \Psi_\varphi(\psi \circ \theta)(x, F_0) &= (\varphi^Z)^{\text{tm}}((\psi \circ \theta)(x, F_0)) && \text{by the definition of } \Psi_\varphi(\psi \circ \theta) \\ &= (\varphi^Z)^{\text{tm}}(\theta^{\text{tm}}(\psi(x, F_0))) && \text{according to the definition of composition} \\ &&& \text{in } \text{Subst}_{\langle F,P \rangle} \end{aligned}$$

$$\begin{aligned}
&= \Psi_\varphi(\theta)^{\text{tm}}((\varphi^Y)^{\text{tm}}(\psi(x, F_0))) && \text{because } (\varphi^Y)^{\text{tm}} \circ \Psi_\varphi(\theta)^{\text{tm}} = \theta^{\text{tm}} \circ (\varphi^Z)^{\text{tm}} \\
&= \Psi_\varphi(\theta)^{\text{tm}}(\Psi_\varphi(\psi)(x, F_0)) && \text{by the definition of } \Psi_\varphi(\psi) \\
&= (\Psi_\varphi(\psi) \circ \Psi_\varphi(\theta))(x, F_0). && \text{according to the definition of composition} \\
& && \text{in } \text{Subst}_{\langle F', P' \rangle}
\end{aligned}$$

In a similar manner, it can be shown that Ψ_φ also preserves identities.

As regards the last two components of the comorphism, the naturality of α_φ and β_φ amounts to the fact that the following two squares commute for every $\langle F, P \rangle$ -substitution $\psi: X \rightarrow Y$.

$$\begin{array}{ccc}
\text{Sen}_{\langle F, P \rangle}(X) & \xrightarrow{\varphi^X(_)} \text{Sen}_{\langle F', P' \rangle}(\Psi_\varphi(X)) & \text{Mod}_{\langle F, P \rangle}(X) & \xleftarrow{-\uparrow_{\varphi^X}} \text{Mod}_{\langle F', P' \rangle}(\Psi_\varphi(X)) \\
\psi(_) \downarrow & & \downarrow \Psi_\varphi(\psi)(_) & & \uparrow -\uparrow_{\Psi_\varphi(\psi)} \\
\text{Sen}_{\langle F, P \rangle}(Y) & \xrightarrow{\varphi^Y(_)} \text{Sen}_{\langle F', P' \rangle}(\Psi_\varphi(Y)) & \text{Mod}_{\langle F, P \rangle}(Y) & \xleftarrow{-\uparrow_{\varphi^Y}} \text{Mod}_{\langle F', P' \rangle}(\Psi_\varphi(Y))
\end{array}$$

This can be established with ease through a series of straightforward calculations. For example, in the case of relational atoms $\pi(t_1, \dots, t_n)$ we obtain

$$\begin{aligned}
&\Psi_\varphi(\psi)(\varphi^X(\pi(t_1, \dots, t_n))) \\
&= \Psi_\varphi(\psi)(\varphi_n^{\text{rel}}(\pi)((\varphi^X)^{\text{tm}}(t_1), \dots)) && \text{by the definition of } \varphi^X(_) \\
&= \varphi_n^{\text{rel}}(\pi)(\Psi_\varphi(\psi)^{\text{tm}}((\varphi^X)^{\text{tm}}(t_1), \dots)) && \text{by the definition of } \Psi_\varphi(\psi)(_) \\
&= \varphi_n^{\text{rel}}(\pi)((\varphi^Y)^{\text{tm}}(\psi^{\text{tm}}(t_1), \dots)) && \text{because } (\varphi^X)^{\text{tm}} \circ \Psi_\varphi(\psi)^{\text{tm}} \\
& && = \psi^{\text{tm}} \circ (\varphi^Y)^{\text{tm}} \\
&= \varphi^Y(\pi(\psi^{\text{tm}}(t_1), \dots)) && \text{by the definition of } \varphi^Y(_) \\
&= \varphi^Y(\psi(\pi(t_1, \dots, t_n))). && \text{by the definition of } \psi(_) \quad \square
\end{aligned}$$

3.2 SUBSTITUTION SYSTEMS

The institutions of substitutions discussed in [Section 3.1](#) provide the most basic building blocks needed for writing logic programs: signatures of variables and sentences over those signatures. For instance, the following definition of the addition of natural numbers (which makes use of the dedicated clausal notation specific to logic programming)

$$\begin{aligned}
&\text{add}(\emptyset, M, M) \xleftarrow{M} \\
&\text{add}(s M, N, s P) \xleftarrow{M, N, P} \text{add}(M, N, P)
\end{aligned}$$

can be presented formally as the set given by the universal closures of the sentences $\text{true} \Rightarrow \text{add}(\emptyset, M, M)$ and $\text{add}(M, N, P) \Rightarrow \text{add}(s M, N, s P)$ defined

over the signatures of $\langle F_{\text{NAT}}, P_{\text{NAT}} \rangle$ -variables $\{M\}$ and $\{M, N, P\}$, respectively.

In order to describe the semantics of such definitions at the same level of abstraction, we need to make explicit the translations of sentences and the reductions of models that correspond to signature extensions such as $\langle F_{\text{NAT}}, P_{\text{NAT}} \rangle \subseteq \langle F_{\text{NAT}} \cup \{M\}, P_{\text{NAT}} \rangle$. A possible approach is to consider signatures of Σ -variables, for some signature Σ of a given logical system, as specific morphisms $X: \Sigma \rightarrow \Sigma(X)$. This treatment of (signatures of) variables was first outlined in [ST84] as part of an institutional approach to open formulae, and is related to many developments in institution theory such as institution-independent semantics for quantifiers [Tar86], ultraproducts [Diao8], general versions of Herbrand theorems [Diao04], Birkhoff completeness [CGo8], hybridization [Mar+11], structural induction [Dia11], constructor-based logics [GFO12], and forcing techniques [GP10; Găi14].

The framework that we propose for formalizing signatures of variables (and substitutions) generalizes the aforementioned approach by taking into account only their corresponding extensions of rooms. This choice is motivated by the difficulties that may arise in defining variables as signature extensions in other logical systems of interest for logic programming, such as the institution of asynchronous relational networks [see TF13]. In that case, signatures (and, consequently, signature morphisms) and variables are significantly different from a structural point of view: the former are logical systems satisfying given properties, while the latter are hypergraphs whose vertices and hyperedges are labelled with signatures and models of their underlying logical system; for this reason, variables cannot be faithfully represented through extensions of their base signatures.

DEFINITION 3.2.1 (Substitution system). A *substitution system* is a triple $\langle \text{Subst}, G, \mathcal{S} \rangle$, usually denoted simply by \mathcal{S} , consisting of

- a category Subst of *signatures of variables* and *substitutions*,
- a room G of *ground sentences* and *models*, and
- a functor $\mathcal{S}: \text{Subst} \rightarrow G / \mathbb{R}\text{Room}$ defining, for every signature of variables X , the corridor $\mathcal{S}(X): G \rightarrow G(X)$ to the room $G(X)$ of *X-sentences* and *X-models*.

Therefore, given a room $G = \langle \text{Sen}(G), \text{Mod}(G), \varepsilon_G \rangle$ of ground sentences and models, the signatures of variables are defined abstractly as objects X of a category Subst for which we consider (a) a set $\text{Sen}(G(X))$ of *X-sentences* together with a sentence-extension map $\alpha_X: \text{Sen}(G) \rightarrow \text{Sen}(G(X))$, (b) a category $\text{Mod}(G(X))$ of *X-models* together with a model-reduction functor $\beta_X: \text{Mod}(G(X)) \rightarrow \text{Mod}(G)$, and (c) a satisfaction relation $\varepsilon_{G(X)}$ between

X -models and X -sentences such that the following satisfaction condition holds for all X -models N and ground sentences ρ :

$$N \vDash_{G(X)} \alpha_X(\rho) \quad \text{if and only if} \quad \beta_X(N) \vDash_G \rho.$$

Similarly, substitutions $\psi: X \rightarrow Y$ are arrows in Subst for which we consider (a) a sentence-translation map $\text{Sen}(\psi): \text{Sen}(G(X)) \rightarrow \text{Sen}(G(Y))$ and (b) a model-reduction functor $\text{Mod}(\psi): \text{Mod}(G(Y)) \rightarrow \text{Mod}(G(X))$ such that the following diagrams commute

$$\begin{array}{ccc} & \text{Sen}(G) & \\ \alpha_X \swarrow & & \searrow \alpha_Y \\ \text{Sen}(G(X)) & \xrightarrow{\text{Sen}(\psi)} & \text{Sen}(G(Y)) \end{array} \quad \begin{array}{ccc} & \text{Mod}(G) & \\ \beta_X \swarrow & & \searrow \beta_Y \\ \text{Mod}(G(X)) & \xleftarrow{\text{Mod}(\psi)} & \text{Mod}(G(Y)) \end{array}$$

and, for every Y -model N and X -sentence ρ ,

$$N \vDash_{G(Y)} \text{Sen}(\psi)(\rho) \quad \text{if and only if} \quad \text{Mod}(\psi)(N) \vDash_{G(X)} \rho.$$

It should be noted that, whenever the signatures of variables are defined as extensions of their base signatures (in a given institution), the corridors $\langle \text{Sen}(\psi), \text{Mod}(\psi) \rangle$ determined by substitutions $\psi: X \rightarrow Y$ correspond to the institution-independent concept of substitution defined in [Diao04]. A detailed presentation of this approach is given in Section 4.2.

EXAMPLE 3.2.2. Every QF-FOL_{\neq}^1 -signature $\langle F, P \rangle$ gives rise to a substitution system $(\text{QF-FOL}_{\neq}^1)_{\langle F, P \rangle}: \text{Subst}_{\langle F, P \rangle} \rightarrow \text{QF-FOL}_{\neq}^1(F, P) / \mathbb{R}\text{oom}^2$ in which

- signatures of $\langle F, P \rangle$ -variables X determine corridors given by set-theoretic inclusions $\text{Sen}(F, P) \subseteq \text{Sen}_{\langle F, P \rangle}(X)$ and model-reduct functors $\text{Mod}_{\langle F, P \rangle}(X) \rightarrow \text{Mod}(F, P)$ that forget the interpretation of variables, and
- substitutions $\psi: X \rightarrow Y$ are mapped to $\langle \text{Sen}_{\langle F, P \rangle}(\psi), \text{Mod}_{\langle F, P \rangle}(\psi) \rangle$.

Note that we can easily recover the institution of $\langle F, P \rangle$ -substitutions described in Corollary 3.1.2 (regarded as a functor into $\mathbb{R}\text{oom}$) by means of a straightforward composition:

$$(\text{QF-FOL}_{\neq}^1)_{\langle F, P \rangle} = (\text{QF-FOL}_{\neq}^1)_{\langle F, P \rangle} \circ \text{Id}_{\text{QF-FOL}_{\neq}^1(F, P)}.$$

Working with substitution systems instead of the simpler institutions of

² In this case, the room $\text{QF-FOL}_{\neq}^1(F, P)$ is just the image of the first-order signature $\langle F, P \rangle$ under the functor representation of the institution QF-FOL_{\neq}^1 .

substitutions means that we also need to consider an adequate notion of map of substitution systems.

DEFINITION 3.2.3 (Morphism of substitution systems). A *morphism* between substitution systems $\mathcal{S}: \text{Subst} \rightarrow G / \mathbb{R}\text{oom}$ and $\mathcal{S}': \text{Subst}' \rightarrow G' / \mathbb{R}\text{oom}$ is a triple $\langle \Psi, \kappa, \tau \rangle$, where

$$\begin{array}{ccc} \text{Subst} & \xrightarrow{\mathcal{S}} & G / \mathbb{R}\text{oom} \\ \Psi \downarrow & \Downarrow \tau & \uparrow \kappa / \mathbb{R}\text{oom} \\ \text{Subst}' & \xrightarrow{\mathcal{S}'} & G' / \mathbb{R}\text{oom} \end{array}$$

- Ψ is a functor $\text{Subst} \rightarrow \text{Subst}'$,
- κ is a corridor $G \rightarrow G'$, and
- τ is a natural transformation $\mathcal{S} \Rightarrow \Psi \circ \mathcal{S}' \circ (\kappa / \mathbb{R}\text{oom})$.

PROPOSITION 3.2.4. For every morphism of QF-FOL_{\neq}^1 -signatures $\varphi: \langle F, P \rangle \rightarrow \langle F', P' \rangle$, the comorphism $\langle \Psi_{\varphi}, \alpha_{\varphi}, \beta_{\varphi} \rangle$ given in [Proposition 3.1.3](#) can be extended to a morphism of substitution systems

$$\langle \Psi_{\varphi}, \kappa_{\varphi}, \tau_{\varphi} \rangle: (\text{QF-FOL}_{\neq}^1)_{\langle F, P \rangle} \rightarrow (\text{QF-FOL}_{\neq}^1)_{\langle F', P' \rangle}$$

where κ_{φ} is the corridor $\text{QF-FOL}_{\neq}^1(\varphi) = \langle \text{Sen}(\varphi), \text{Mod}(\varphi) \rangle$ and, for every signature of $\langle F, P \rangle$ -variables X , $\tau_{\varphi, X} = \langle \alpha_{\varphi, X}, \beta_{\varphi, X} \rangle$.

PROOF. All we need to check is that, for every signature of $\langle F, P \rangle$ -variables X , $\tau_{\varphi, X}$ is indeed an arrow in the comma category $\text{QF-FOL}_{\neq}^1(F, P) / \mathbb{R}\text{oom}$. This follows directly from the fact that the equality

$$(\text{QF-FOL}_{\neq}^1)_{\langle F, P \rangle}(X) \circ \tau_{\varphi, X} = \kappa_{\varphi} \circ (\text{QF-FOL}_{\neq}^1)_{\langle F', P' \rangle}(\Psi_{\varphi}(X))$$

can be obtained by applying the functor $\text{QF-FOL}_{\neq}^1: \text{Sig}^{\text{QF-FOL}_{\neq}^1} \rightarrow \mathbb{R}\text{oom}$ to the commutative diagram below.

$$\begin{array}{ccc} \langle F, P \rangle & \xrightarrow{\varphi} & \langle F', P' \rangle \\ \wr \swarrow & & \searrow \wr \\ \langle F \cup X, P \rangle & \xrightarrow{\varphi^X} & \langle F' \cup X^{\varphi}, P' \rangle \end{array}$$

□

As expected, the composition of morphisms of substitution systems can be straightforwardly defined in terms of their components: for any two morphisms $\langle \Psi, \kappa, \tau \rangle: \mathcal{S} \rightarrow \mathcal{S}'$ and $\langle \Psi', \kappa', \tau' \rangle: \mathcal{S}' \rightarrow \mathcal{S}''$ between substitution

systems $\mathcal{S}: \text{Subst} \rightarrow G / \mathbb{R}\text{oom}$, $\mathcal{S}': \text{Subst}' \rightarrow G' / \mathbb{R}\text{oom}$, and $\mathcal{S}'': \text{Subst}'' \rightarrow G'' / \mathbb{R}\text{oom}$, $\langle \Psi, \kappa, \tau \rangle \mathbin{\S} \langle \Psi', \kappa', \tau' \rangle: \mathcal{S} \rightarrow \mathcal{S}''$ is given by

- the translation of signatures of variables $\Psi \mathbin{\S} \Psi': \text{Subst} \rightarrow \text{Subst}''$,
- the translation of ground sentences and models $\kappa \mathbin{\S} \kappa': G \rightarrow G''$, and
- the translation of sentences and models defined over signatures of variables

$$\tau \mathbin{\S} (\Psi \cdot \tau' \cdot (\kappa / \mathbb{R}\text{oom})): \mathcal{S} \Rightarrow (\Psi \mathbin{\S} \Psi') \mathbin{\S} \mathcal{S}'' \mathbin{\S} (\kappa \mathbin{\S} \kappa' / \mathbb{R}\text{oom}),$$

where, for every $X \in |\text{Subst}|$, $(\tau \mathbin{\S} (\Psi \cdot \tau' \cdot (\kappa / \mathbb{R}\text{oom})))_X = \tau_X \mathbin{\S} \tau'_{\Psi(X)}$.

Together with the obvious identities, the construction above yields a category SubstSys with substitution systems as objects and morphisms of substitution systems as arrows.

FACT 3.2.5. The category SubstSys arises from the Grothendieck construction for the functor $[_ \rightarrow _ / \mathbb{R}\text{oom}]: (\text{Cat} \times \mathbb{R}\text{oom})^{\text{op}} \rightarrow \text{Cat}$ that maps

- every category Subst and room G to $[\text{Subst} \rightarrow G / \mathbb{R}\text{oom}]$, the category of functors from Subst to $G / \mathbb{R}\text{oom}$, and
- every functor $\Psi: \text{Subst} \rightarrow \text{Subst}'$ and corridor $\kappa: G \rightarrow G'$ to the composition functor $\Psi _ (\kappa / \mathbb{R}\text{oom}): [\text{Subst}' \rightarrow G' / \mathbb{R}\text{oom}] \rightarrow [\text{Subst} \rightarrow G / \mathbb{R}\text{oom}]$.

We conclude the present section by noticing that the construction of first-order substitution systems is functorial, in the sense that it can be described as a functor into SubstSys (see [Example 3.2.2](#) and [Proposition 3.2.4](#)). We can thus define the following concept of generalized substitution system.

DEFINITION 3.2.6 (Generalized substitution system). *Generalized substitution systems* are objects of the category $[_ \rightarrow \text{SubstSys}]^{\#}$ of functors into SubstSys .

Therefore, generalized substitution systems are functors $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$ from a category Sig of *signatures* and *signature morphisms* to SubstSys . Because this definition is rather compact and lacks some of the transparency needed for dealing with logic-programming languages, let us first establish the notations and terminology that we will use throughout our work.

- Given a signature Σ of a generalized substitution system \mathcal{GS} , we will denote the (local) substitution system $\mathcal{GS}(\Sigma)$ by $\mathcal{GS}_{\Sigma}: \text{Subst}_{\Sigma} \rightarrow G_{\Sigma} / \mathbb{R}\text{oom}$, and we will refer to the objects and morphisms of the category Subst_{Σ} as *signatures of Σ -variables* and *Σ -substitutions*, respectively.³

³ In this regard, generalized substitution systems are similar to the context institutions of [Paw95], which enrich the structure of institutions with explicit notions of context (in place of signatures of variables) and substitution; the latter, however, are concrete, in the sense that the category of models of every signature is concrete over the category of indexed sets.

The room G_Σ consists of the set $\text{Sen}(\Sigma)$ of *ground Σ -sentences*, the category $\text{Mod}(\Sigma)$ of Σ -*models*, and the Σ -*satisfaction relation* $\vDash_\Sigma \subseteq |\text{Mod}(\Sigma)| \times \text{Sen}(\Sigma)$.

- On objects, the functor $\mathcal{G}\mathcal{S}_\Sigma$ maps every signature of Σ -variables X to the corridor $\mathcal{G}\mathcal{S}_\Sigma(X) = \langle \alpha_{\Sigma,X}, \beta_{\Sigma,X} \rangle$ from G_Σ to the room $G_\Sigma(X)$, whose set of X -*sentences*, category of X -*models*, and X -*satisfaction relation* are denoted by $\text{Sen}_\Sigma(X)$, $\text{Mod}_\Sigma(X)$, and $\vDash_{\Sigma,X}$, respectively.

$$\alpha_{\Sigma,X}: \text{Sen}(\Sigma) \rightarrow \text{Sen}_\Sigma(X) \quad \beta_{\Sigma,X}: \text{Mod}_\Sigma(X) \rightarrow \text{Mod}(\Sigma)$$

- On morphisms, $\mathcal{G}\mathcal{S}_\Sigma$ maps every Σ -substitution $\psi: X \rightarrow Y$ to the corridor $\mathcal{G}\mathcal{S}_\Sigma(\psi) = \langle \text{Sen}_\Sigma(\psi), \text{Mod}_\Sigma(\psi) \rangle$ from $G_\Sigma(X)$ to $G_\Sigma(Y)$, which satisfies, by definition, the equality $\mathcal{G}\mathcal{S}_\Sigma(X) \circ \mathcal{G}\mathcal{S}_\Sigma(\psi) = \mathcal{G}\mathcal{S}_\Sigma(Y)$.

$$\begin{array}{ccc} & \text{Sen}(\Sigma) & \\ \alpha_{\Sigma,X} \swarrow & & \searrow \alpha_{\Sigma,Y} \\ \text{Sen}_\Sigma(X) & \xrightarrow{\text{Sen}_\Sigma(\psi)} & \text{Sen}_\Sigma(Y) \end{array} \quad \begin{array}{ccc} & \text{Mod}(\Sigma) & \\ \beta_{\Sigma,X} \swarrow & & \searrow \beta_{\Sigma,Y} \\ \text{Mod}_\Sigma(X) & \xleftarrow{\text{Mod}_\Sigma(\psi)} & \text{Mod}_\Sigma(Y) \end{array}$$

- With respect to signature morphisms, every arrow $\varphi: \Sigma \rightarrow \Sigma'$ in Sig determines a morphism of substitution systems $\mathcal{G}\mathcal{S}_\varphi = \langle \Psi_\varphi, \kappa_\varphi, \tau_\varphi \rangle$ from $\mathcal{G}\mathcal{S}_\Sigma$ to $\mathcal{G}\mathcal{S}_{\Sigma'}$, where κ_φ is the corridor $\langle \text{Sen}(\varphi), \text{Mod}(\varphi) \rangle$ between G_Σ and $G_{\Sigma'}$ and, for every signature of Σ -variables X , $\tau_{\varphi,X}$ is the corridor $\langle \alpha_{\varphi,X}, \beta_{\varphi,X} \rangle$ between $G_\Sigma(X)$ and $G_{\Sigma'}(\Psi_\varphi(X))$.

$$\begin{array}{ccc} \text{Sen}(\Sigma) & \xrightarrow{\text{Sen}(\varphi)} & \text{Sen}(\Sigma') \\ \alpha_{\Sigma,X} \downarrow & & \downarrow \alpha_{\Sigma',\Psi_\varphi(X)} \\ \text{Sen}_\Sigma(X) & \xrightarrow{\alpha_{\varphi,X}} & \text{Sen}_{\Sigma'}(\Psi_\varphi(X)) \end{array} \quad \begin{array}{ccc} \text{Mod}(\Sigma) & \xleftarrow{\text{Mod}(\varphi)} & \text{Mod}(\Sigma') \\ \beta_{\Sigma,X} \uparrow & & \uparrow \beta_{\Sigma',\Psi_\varphi(X)} \\ \text{Mod}_\Sigma(X) & \xleftarrow{\beta_{\varphi,X}} & \text{Mod}_{\Sigma'}(\Psi_\varphi(X)) \end{array}$$

In addition, we adopt similar notational conventions as in the case of institutions. For example, we may use superscripts as in $\text{Subst}_\Sigma^{\text{GS}}$ in order to avoid potential ambiguities; or we may drop the subscripts of $\vDash_{\Sigma,X}$ when there is no danger of confusion. Likewise, we will often denote the functions $\text{Sen}(\varphi)$, $\alpha_{\Sigma,X}$, and $\text{Sen}_\Sigma(\psi)$ by $\varphi(_)$, $X(_)$, and $\psi(_)$, respectively, and the functors $\text{Mod}(\varphi)$, $\beta_{\Sigma,X}$, and $\text{Mod}_\Sigma(\psi)$ by $_ \uparrow \varphi$, $_ \uparrow \Sigma$, and $_ \uparrow \psi$.

EXAMPLE 3.2.7. The quantifier-free, single-sorted fragment of first-order logic without equality forms a generalized substitution system, which we denote by QF-FOL_\neq^1 .

$$\text{QF-FOL}_\neq^1: \text{Sig}^{\text{QF-FOL}_\neq^1} \rightarrow \text{SubstSys}$$

3.2.1 QUANTIFIED SENTENCES

The last ingredient needed to interpret logic programs is an appropriate concept of quantified sentence over a given generalized substitution system. To this purpose, we introduce universal and existential closures of sentences defined over signatures of variables by adapting the general institution-independent quantifiers of [Tar86] to the present framework of generalized substitution systems.

DEFINITION 3.2.8 (Quantified sentence). In any generalized substitution system $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$, a *universally quantified Σ -sentence* is a structure $\forall X \cdot \rho$, where X is a signature of Σ -variables and ρ is an X -sentence. The denotation of $\forall X \cdot \rho$ is given by the class of Σ -models whose X -expansions satisfy ρ . To be more precise, a Σ -model M is a model of $\forall X \cdot \rho$, denoted $M \models_{\Sigma}^{\text{qs}} \forall X \cdot \rho$, if, for every X -model N such that $N \upharpoonright_{\Sigma} = M$, $N \models_{\Sigma, X} \rho$.

Existentially quantified Σ -sentences $\exists X \cdot \rho$ are introduced in a similar manner. Their semantics is, as expected, existential rather than universal: the denotation of a sentence $\exists X \cdot \rho$ is given by the class of Σ -models that admit X -expansions satisfying ρ .

Let $\text{QSen}(\Sigma)$, or $\text{QSen}^{\mathcal{GS}}(\Sigma)$ when we want to make explicit the underlying generalized substitution system, be the set of quantified sentences (universal or existential) over a signature Σ of a generalized substitution system $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$. It is straightforward to see that the map QSen can be extended to a functor $\text{Sig} \rightarrow \text{Set}$ by defining $\text{QSen}(\varphi)(QX \cdot \rho)$ as $Q\Psi_{\varphi}(X) \cdot \alpha_{\varphi, X}(\rho)$ for every morphism of signatures $\varphi: \Sigma \rightarrow \Sigma'$ and every quantified Σ -sentence $QX \cdot \rho$, where $Q \in \{\forall, \exists\}$.

FACT 3.2.9. For any generalized substitution system $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$, QSen is a *quantified-sentence functor* $\text{Sig} \rightarrow \text{Set}$.

In order to reason about programs defined over different signatures, it is necessary to ensure that the translation of quantified sentences is consistent with the reduction of models – in the sense that the satisfaction relation is preserved. As in many institutions that capture logical systems with quantifiers, this condition relies on the essential property of model amalgamation, that is on the possibility of combining models of different but related signatures, provided that they have a common reduct to a given shared signature. The form of model amalgamation that we use here was first introduced in [BPP85]; other early papers include [ST88a; DGS93]. Its importance was also emphasized in works on heterogeneous specifications such as [Taroo; Boro2], which are closely related to our setting.

DEFINITION 3.2.10 (Model amalgamation). A generalized substitution system $\mathfrak{GS}: \text{Sig} \rightarrow \text{SubstSys}$ has (*weak*) *model amalgamation* when, for every signature morphism $\varphi: \Sigma \rightarrow \Sigma'$ and every signature of Σ -variables X , the diagram depicted below is a (weak) pullback.

$$\begin{array}{ccc} |\text{Mod}(\Sigma)| & \xleftarrow{-\uparrow_{\varphi}} & |\text{Mod}(\Sigma')| \\ \uparrow -\uparrow_{\Sigma} & & \uparrow -\uparrow_{\Sigma'} \\ |\text{Mod}_{\Sigma}(X)| & \xleftarrow{\beta_{\varphi,X}} & |\text{Mod}_{\Sigma'}(\Psi_{\varphi}(X))| \end{array}$$

This means that for every Σ' -model M' and every X -model N such that $M' \uparrow_{\varphi} = N \uparrow_{\Sigma}$ there exists a $\Psi_{\varphi}(X)$ -model N' , called the *amalgamation* of M' and N , that satisfies $N' \uparrow_{\Sigma'} = M'$ and $\beta_{\varphi,X}(N') = N$. Whenever this holds, the following commutative square, which subsumes the diagram above, is said to be a (*weak*) *model-amalgamation square*:

$$\begin{array}{ccc} G_{\Sigma} & \xrightarrow{\kappa_{\varphi}} & G_{\Sigma'} \\ \mathfrak{GS}_{\Sigma}(X) \downarrow & & \downarrow \mathfrak{GS}_{\Sigma'}(\Psi_{\varphi}(X)) \\ G_{\Sigma}(X) & \xrightarrow{\tau_{\varphi,X}} & G_{\Sigma'}(\Psi_{\varphi}(X)) \end{array}$$

In generalized substitution systems such as QF-FOL_{\neq}^1 – as well as, for example, $\text{QF-FOL}_{=}$, discussed in [Chapter 4](#) – for every signature morphism $\varphi: \langle F, P \rangle \rightarrow \langle F', P' \rangle$ and every signature of $\langle F, P \rangle$ -variables X , the commutative square of interest for model amalgamation can be obtained by taking the image through QF-FOL_{\neq}^1 of the following diagram of signature morphisms:

$$\begin{array}{ccc} \langle F, P \rangle & \xrightarrow{\varphi} & \langle F', P' \rangle \\ \subseteq \downarrow & & \downarrow \subseteq \\ \langle F \cup X, P \rangle & \xrightarrow{\varphi^X} & \langle F' \cup X^{\varphi}, P' \rangle \end{array}$$

It is well known that such diagrams are pushouts and, moreover, that every pushout of first-order signature morphisms gives rise to a model-amalgamation square (details can be found, for example, in [[Dia08](#); [ST11](#)]). These considerations lead us to the following result.

PROPOSITION 3.2.11. *The generalized substitution system QF-FOL_{\neq}^1 has model amalgamation.* \square

Model amalgamation allows us to prove that the translations of quantified sentences and the reductions of models preserve the satisfaction relation.

PROPOSITION 3.2.12. *Let $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$ be a generalized substitution system that has weak model amalgamation. Then for every signature morphism $\varphi: \Sigma \rightarrow \Sigma'$, every quantified Σ -sentence $QX \cdot \rho$, and every Σ' -model M' ,*

$$M' \vDash_{\Sigma'}^{\text{qs}} \text{QSen}(\varphi)(QX \cdot \rho) \quad \text{if and only if} \quad \text{Mod}(\varphi)(M') \vDash_{\Sigma}^{\text{qs}} QX \cdot \rho.$$

PROOF. Since the two kinds of quantified sentences can be treated similarly, we focus here only on the case of universal sentences. Let us thus consider a signature morphism $\varphi: \Sigma \rightarrow \Sigma'$, a Σ -sentence $\forall X \cdot \rho$, and a Σ' -model M' .

The proof of the ‘if’ part is simpler and it does not require the model-amalgamation property. Assume that $\text{Mod}(\varphi)(M') \vDash_{\Sigma}^{\text{qs}} \forall X \cdot \rho$, and let N' be an arbitrary $\Psi_{\varphi}(X)$ -expansion of M' . Since $\kappa_{\varphi} \circ \tau_{\varphi, X} \circ \mathcal{GS}_{\Sigma'}(\Psi_{\varphi}(X)) = \mathcal{GS}_{\Sigma}(X) \circ \tau_{\varphi, X}$ (because $\langle \Psi_{\varphi}, \kappa_{\varphi}, \tau_{\varphi} \rangle$ is a morphism of substitution systems), it follows that $\beta_{\varphi, X}(N')$ is an X -expansion of $\text{Mod}(\varphi)(M')$. As a result, $\beta_{\varphi, X}(N') \vDash_{\Sigma, X} \rho$, which implies, by the satisfaction condition for $\tau_{\varphi, X}$, that $N' \vDash_{\Sigma', \Psi_{\varphi}(X)} \alpha_{\varphi, X}(\rho)$. Hence, given that, by definition, $\text{QSen}(\varphi)(\forall X \cdot \rho)$ is the universally quantified sentence $\forall \Psi_{\varphi}(X) \cdot \alpha_{\varphi, X}(\rho)$, we conclude that $M' \vDash_{\Sigma'}^{\text{qs}} \text{QSen}(\varphi)(\forall X \cdot \rho)$.

For the ‘only if’ part, assume that M' is a model of $\text{QSen}(\varphi)(\forall X \cdot \rho)$, and let N be an X -expansion of $\text{Mod}(\varphi)(M')$, which means that $\text{Mod}(\varphi)(M') = \beta_{\Sigma, X}(N)$. We need to show that N satisfies ρ . Since \mathcal{GS} has weak model amalgamation, we deduce that there exists a $\Psi_{\varphi}(X)$ -model N' such that $\beta_{\Sigma', \Psi_{\varphi}(X)}(N') = M'$ and $\beta_{\varphi, X}(N') = N$. This means that N' is a $\Psi_{\varphi}(X)$ -expansion of M' , which further implies that $N' \vDash_{\Sigma', \Psi_{\varphi}(X)} \alpha_{\varphi, X}(\rho)$ because, by hypothesis, $M' \vDash_{\Sigma'}^{\text{qs}} \forall \Psi_{\varphi}(X) \cdot \alpha_{\varphi, X}(\rho)$. Therefore, by the satisfaction condition for $\tau_{\varphi, X}$, we have $\beta_{\varphi, X}(N') \vDash_{\Sigma, X} \rho$. \square

COROLLARY 3.2.13. *For any generalized substitution system $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$ that has weak model amalgamation, the structure $\mathcal{GS}^{\text{qs}} = \langle \text{Sig}, \text{QSen}, \text{Mod}, \vDash^{\text{qs}} \rangle$ describes an institution – the institution of quantified sentences over \mathcal{GS} . \square*

3.3 ABSTRACT LOGIC PROGRAMMING

As we have seen, generalized substitution systems provide us with a framework that is rich and flexible enough for capturing some of the most essential aspects of logic programming. However, since they make no assumptions on the structure of sentences defined over signatures of variables, we cannot distinguish between clauses and queries, nor can we describe unifiable sentences, which would further enable the search for solutions to queries by means of a suitable concept of resolution. To overcome these limitations, we

extend the notion of generalized substitution system with appropriate functors that define local clauses and queries, and with binary inference rules that allow the integration of the declarative and the operational semantics of logic programming. The following property gives a concise presentation of the local sentences defined by a generalized substitution system.

FACT 3.3.1. Any generalized substitution system $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$ determines a *local-sentence functor* $\text{LSen}: \text{Sig} \rightarrow [_ \rightarrow \text{Set}]^\sharp$, denoted $\text{LSen}^{\mathcal{GS}}$ when we want to make explicit the generalized substitution system, that maps

- every signature Σ to the pair $\langle \text{Subst}_\Sigma, \mathcal{GS}_\Sigma \circ \llbracket _ \rrbracket_{G_\Sigma} \circ \text{Sen} \rangle$, usually identified with the second component, the functor $\mathcal{GS}_\Sigma \circ \llbracket _ \rrbracket_{G_\Sigma} \circ \text{Sen}: \text{Subst}_\Sigma \rightarrow \text{Set}$, and
- every signature morphism $\varphi: \Sigma \rightarrow \Sigma'$ to $\langle \Psi_\varphi, \tau_\varphi \cdot (\llbracket _ \rrbracket_{G_\Sigma} \circ \text{Sen}) \rangle$.

$$\begin{array}{ccccc}
 \text{Subst}_\Sigma & \xrightarrow{\mathcal{GS}_\Sigma} & G_\Sigma / \mathbb{R}\text{oom} & \xrightarrow{\llbracket _ \rrbracket_{G_\Sigma}} & \mathbb{R}\text{oom} & \xrightarrow{\text{Sen}} & \text{Set} \\
 \Psi_\varphi \downarrow & & \Downarrow \tau_\varphi & \nearrow \kappa_\varphi / \mathbb{R}\text{oom} & & & \\
 \text{Subst}_{\Sigma'} & \xrightarrow{\mathcal{GS}_{\Sigma'}} & G_{\Sigma'} / \mathbb{R}\text{oom} & \xrightarrow{\llbracket _ \rrbracket_{G_{\Sigma'}}} & \mathbb{R}\text{oom} & &
 \end{array}$$

This means that, for every signature Σ , the functor $\text{LSen}(\Sigma)$ describes the sets $\text{LSen}(\Sigma)(X) = \text{Sen}_\Sigma(X)$ of *local Σ -sentences*, where X is a signature of Σ -variables, together with *translations of local sentences* $\text{LSen}(\Sigma)(\psi) = \text{Sen}_\Sigma(\psi)$, where ψ is a Σ -substitution. In addition, for every signature morphism $\varphi: \Sigma \rightarrow \Sigma'$, $\text{LSen}(\varphi)$ describes the translation of signatures of Σ -variables $\Psi_\varphi: \text{Subst}_\Sigma \rightarrow \text{Subst}_{\Sigma'}$ and the family of natural maps $\alpha_{\varphi, X}: \text{Sen}_\Sigma(X) \rightarrow \text{Sen}_{\Sigma'}(\Psi_\varphi(X))$, indexed by signatures of Σ -variables X .

We can now define the main algebraic concept that underlies our approach to logic-independent logic programming.

DEFINITION 3.3.2 (Logic-programming framework). A *logic-programming framework* is a tuple $\mathcal{F} = \langle \mathcal{GS}, C, Q, \Vdash \rangle$, where

- \mathcal{GS} is a generalized substitution system that has weak model amalgamation,
- C and Q are generalized subfunctors of LSen , defining the subsets $C_\Sigma(X) \subseteq \text{Sen}_\Sigma(X)$ and $Q_\Sigma(X) \subseteq \text{Sen}_\Sigma(X)$ of *X -clauses* and *X -queries*, respectively, for every signature Σ and every signature of Σ -variables X , and
- \Vdash is a generalized subfunctor of $(Q \times C) \times Q$ defining, for every signature Σ and signature of Σ -variables X , the subset $\Vdash_{\Sigma, X} \subseteq (Q_\Sigma(X) \times C_\Sigma(X)) \times Q_\Sigma(X)$ of *X -goal-directed rules* (ρ_1, γ, ρ_2) , usually denoted by $\rho_1, \gamma \Vdash_{\Sigma, X} \rho_2$,

such that the following *soundness property* holds for every signature Σ , signature of Σ -variables X , X -queries ρ_1 and ρ_2 , and every X -clause γ :

$$\rho_1, \gamma \Vdash_{\Sigma, X} \rho_2 \quad \text{implies} \quad \{\rho_2, \gamma\} \vDash_{\Sigma, X} \rho_1.$$

EXAMPLE 3.3.3. Conventional first-order logic programming is defined over the generalized substitution system QF-FOL_{\neq}^1 as follows: for every signature $\langle F, P \rangle$ and every signature of $\langle F, P \rangle$ -variables X ,

- the set $C_{\langle F, P \rangle}(X)$ of X -clauses consists of all implications of the form $\bigwedge H \Rightarrow C$, where H is a finite set of relational atoms and C is an atom,
- the set $Q_{\langle F, P \rangle}(X)$ of X -queries consists of all finite conjunctions of atoms $\bigwedge Q$,
- the rules are given by $\bigwedge(\{C\} \cup Q), (\bigwedge H \Rightarrow C) \Vdash_{\langle F, P \rangle, X} \bigwedge(Q \cup H)$.

One can easily see that the (local) first-order clauses, queries, and goal-directed rules are preserved along both signature morphisms and substitutions; for example, the translation of a clause along a signature morphism, when regarded as a sentence over the domain signature of that morphism, is a clause defined over the codomain signature of the morphism. Moreover, the goal-directed rules correspond, in essence, to applications of *modus ponens*, and thus they are sound. As a result, the constructions above define a logic-programming framework, which we denote by FOL_{\neq}^1 .

The definitions of clauses and queries are rather straightforward.

DEFINITION 3.3.4 (Clause and query). Let Σ be a signature of the underlying generalized substitution system \mathcal{GS} of a logic-programming framework $\langle \mathcal{GS}, C, Q, \Vdash \rangle$. A Σ -*clause* is a universally quantified sentence $\forall X \cdot \gamma$ over Σ such that $\gamma \in C_{\Sigma}(X)$. Similarly, a Σ -*query* is an existentially quantified sentence $\exists X \cdot \rho$ over Σ such that $\rho \in Q_{\Sigma}(X)$.

We obtain in this way two subfunctors $\forall C$ (defining the clauses) and $\exists Q$ (defining the queries) of the sentence functor QSen of the institution of quantified sentences over \mathcal{GS} (see [Corollary 3.2.13](#)).

Following recent developments in structuring specifications [see [Dia12a](#); [Tut13](#)], we propose an axiomatic approach to logic-programming languages in which programs are treated as abstract entities, each of them defining a signature (in a given logic-programming framework), a class of models – that provides the denotational semantics of the program – and an appropriate set of clauses – that supports the operational semantics of the program. In addition, programs are related by morphisms that induce appropriate reductions of models and translations of clauses, defined in terms of their projections to the underlying logic-programming framework.

DEFINITION 3.3.5 (Logic-programming language). A *logic-programming language* $\mathcal{L} = \langle \text{LP}, \text{Sign}, \text{PMod}, \text{Ax} \rangle$ over a framework $\langle \mathcal{GS}, C, Q, \Vdash \rangle$ consists of

- a category \mathbb{LP} of *logic programs* and *morphisms of logic programs*,
- a *signature functor* $\text{Sign}: \mathbb{LP} \rightarrow \text{Sig}$, and
- subfunctors $\text{PMod} \subseteq \text{Sign}^{\text{op}} \circ \text{Mod}$ and $\text{Ax} \subseteq \text{Sign} \circ \forall\text{C}$ defining, for every program P , the category $\text{PMod}(P)$ of P -*models* and the set $\text{Ax}(P)$ of P -*clauses*,

$$\begin{array}{ccccc}
 & & \mathbb{LP} & & \\
 & \swarrow & \downarrow & \searrow & \\
 & \text{PMod}^{\text{op}} & \text{Sign} & \text{Ax} & \\
 \text{Cat}^{\text{op}} & \xleftarrow{\text{Mod}^{\text{op}}} & \text{Sig} & \xrightarrow{\forall\text{C}} & \text{Set}
 \end{array}$$

such that, for every logic program P and model M of P , $M \vDash_{\text{Sign}(P)}^{\text{qs}} \text{Ax}(P)$.

By way of notation, when there is no risk of confusion, we will often denote logic programs P by $\langle \Sigma, \Gamma \rangle$, where $\Sigma = \text{Sign}(P)$ and $\Gamma = \text{Ax}(P)$, emphasizing in this way their corresponding signatures and sets of clauses; moreover, we may denote simply by ν the signature morphism $\text{Sign}(\nu)$ determined by a morphism of programs $\nu: P \rightarrow P'$. We may also indicate that a Σ -model M is a model of P by writing $M \vDash_{\Sigma}^{\text{lp}} P$ in place of $M \in |\text{PMod}(P)|$, and that a quantified Σ -sentence $QX \cdot \rho$ is semantically entailed by P , meaning that $M \vDash_{\Sigma}^{\text{qs}} QX \cdot \rho$ for all P -models M , by writing $P \vDash_{\Sigma}^{\text{lp}} QX \cdot \rho$.

EXAMPLE 3.3.6. Most of the mechanisms used in defining (structured) specifications can also be used to define logic programs. For instance, one of the simplest and most common descriptions of logic programs is as *theory presentations* over a logic-programming framework $\mathcal{F} = \langle \mathcal{SS}, \mathcal{C}, \mathcal{Q}, \mathcal{I} \rangle$ such as \mathcal{FOL}_{\neq}^1 [see e.g. GB92; DGS93]. In this case, the programs defined by the resulting logic-programming language – denoted $\mathcal{F}^{\text{pres}}$ – are pairs $\langle \Sigma, \Gamma \rangle$ of signatures Σ of \mathcal{SS} and sets of Σ -clauses Γ , while the morphisms $\varphi: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$ are merely signature morphisms $\varphi: \Sigma \rightarrow \Sigma'$ such that $\Gamma' \vDash_{\Sigma'}^{\text{qs}} \varphi(\Gamma)$. The functors Sign , Ax , and PMod have their usual interpretations: for every presentation $\langle \Sigma, \Gamma \rangle$, (a) $\text{Sign}(\Sigma, \Gamma)$ is the signature Σ , (b) $\text{Ax}(\Sigma, \Gamma)$ is the set of Σ -clauses Γ , and (c) $\text{PMod}(\Sigma, \Gamma)$ is the full subcategory of $\text{Mod}(\Sigma)$ given by the models of Σ that satisfy Γ .

Under this formalism, based on the first-order signature $\langle F_{\text{NAT}}, P_{\text{NAT}} \rangle$ considered in Section 3.1 and on the clauses presented in Section 3.2, the logic program that captures the addition of natural numbers can be defined over the logic-programming language $(\mathcal{FOL}_{\neq}^1)^{\text{pres}}$ as follows:

signature

ops 0: 0, s_: 1

pred add: 3

axioms

$$\begin{aligned} \text{add}(\emptyset, M, M) &\longleftarrow_M \\ \text{add}(sM, N, sP) &\longleftarrow_{M,N,P} \text{add}(M, N, P) \end{aligned}$$

For the remaining part of this section, we will assume that \mathcal{L} is an arbitrary but fixed logic-programming language $\langle \text{LP}, \text{Sign}, \text{PMod}, \text{Ax} \rangle$ over a logic-programming framework $\mathcal{F} = \langle \mathcal{SS}, \text{C}, \text{Q}, \Vdash \rangle$.

3.3.1 HERBRAND'S THEOREM

From a model-theoretic perspective, a logic program $\langle \Sigma, \Gamma \rangle$ gives a positive answer to a Σ -query $\exists X \cdot \rho$ when it semantically entails the query, which means, of course, that every model of $\langle \Sigma, \Gamma \rangle$ admits an X -expansion that satisfies ρ . In practice, this kind of answer is not always adequate for the particular task at hand, as we may be interested in finding the actual ‘values’ for X – independent of any choice of model of $\langle \Sigma, \Gamma \rangle$ – that meet all the requirements described by ρ . The notion of solution presented below relies technically on the concept of conservative signature of variables.

DEFINITION 3.3.7 (Conservative signature of variables). For any signature Σ , a signature of Σ -variables Y is said to be *conservative* whenever the model-reduct functor $\beta_{\Sigma,Y}: \text{Mod}_{\Sigma}(Y) \rightarrow \text{Mod}(\Sigma)$ is surjective on objects.

Note that, in the case of the single-sorted fragment of relational first-order logic, a (non-empty) signature of $\langle F, P \rangle$ -variables is conservative if and only if the set F_0 of constant-operation symbols is not empty.

DEFINITION 3.3.8 (Solution). For any logic program $\langle \Sigma, \Gamma \rangle$, a Σ -substitution $\psi: X \rightarrow Y$ is a $\langle \Sigma, \Gamma \rangle$ -*solution*, or *correct $\langle \Sigma, \Gamma \rangle$ -answer*, to a Σ -query $\exists X \cdot \rho$ if the signature of variables Y is conservative and $\langle \Sigma, \Gamma \rangle \vDash_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho)$.

Therefore, we have to consider two notions of answer to a query, both of which are relative to a given logic program: the first one corresponds to the denotational semantics of the logic-programming language, while the second provides the necessary foundations for its operational semantics.

Herbrand’s theorem ensures that the two notions are equivalent. First, it reduces the semantic entailment of the considered query to satisfaction in the initial model of the logic program; then, it shows that any expansion of the initial model that satisfies the underlying local sentence of the query gives rise to a solution, and vice versa. These properties are well known in the literature for various logical systems [see e.g. [Llo87](#); [GM87](#); [GMK02](#)], and

they were first investigated in an institution-independent setting in [Dia04]. The result we develop here upgrades the ones from [Dia04] by considering a different set of hypotheses, based on the existence of certain reachable models instead of representable signature morphisms and substitutions; moreover, it can be utilized even when the signatures of variables cannot be described as extensions of their base signatures. These new assumptions are significantly more permissive as they allow us to apply the theory advanced herein to logical systems that, similarly to those discussed in [TF13] (and further developed in Section 5.2.1), do not fit into the framework proposed in [Dia04] – which, in fact, can be shown to be a concrete realization of the conceptual structure that we put forward here (see Theorem 4.3.17).

The following concept of reachable model extends (non-trivially) the homonymous one from [GP10] to generalized substitution systems by eliminating the need for an initial signature of variables with the same sentences, models, and satisfaction relation as its corresponding base signature.

DEFINITION 3.3.9 (Reachable model). Let Σ be a signature and X a signature of Σ -variables in a generalized substitution system. A Σ -model M is said to be *X-reachable* when for every X -expansion N of M there exists a Σ -substitution $\psi: X \rightarrow Y$ such that

- Y is conservative and
- the canonical map $-\upharpoonright_{\Sigma}: N / \text{Mod}_{\Sigma}(\psi) \rightarrow M / \beta_{\Sigma, Y}$ determined by the reduct functor $\beta_{\Sigma, X}$ is surjective on objects.

Hence, given a model N and a substitution ψ as above, the Σ -model M is *X-reachable* if every homomorphism $h: M \rightarrow N_1 \upharpoonright_{\Sigma}$ from M to the Σ -reduct of an Y -model N_1 admits an X -expansion of the form $f: N \rightarrow N_1 \upharpoonright_{\psi}$.

In many concrete examples of institutions [see e.g. GP10], a model M is reachable (with respect to some signature of variables) if and only if all its elements are interpretations of terms, that is if the unique homomorphism $0_{\Sigma} \rightarrow M$ from the initial model of the signature of M to M is epi. In particular, for first-order logic, the initial model $0_{\langle F, P \rangle, \Gamma}$ of a set Γ of $\langle F, P \rangle$ -clauses is *X-reachable* for every signature of $\langle F, P \rangle$ -variables X : every expansion $N_{\langle F, P \rangle, \Gamma}$ of $0_{\langle F, P \rangle, \Gamma}$ yields a substitution $\psi: X \rightarrow \emptyset$ to the empty signature of variables (which is conservative, and has the same models as $\langle F, P \rangle$) such that $0_{\langle F, P \rangle, \Gamma} \upharpoonright_{\psi} = N_{\langle F, P \rangle, \Gamma}$; hence, every homomorphism $h: 0_{\langle F, P \rangle, \Gamma} \rightarrow N_1 \upharpoonright_{\langle F, P \rangle}$ in $\text{Mod}(F, P)$ admits an X -expansion $N_{\langle F, P \rangle, \Gamma} \rightarrow N_1 \upharpoonright_{\psi}$ given by $h \upharpoonright_{\psi}$.

PROPOSITION 3.3.10. In $(\mathcal{FOL}_{\neq}^1)^{\text{pres}}$, every program $\langle \langle F, P \rangle, \Gamma \rangle$ admits an initial model $0_{\langle F, P \rangle, \Gamma}$ that is reachable with respect to all signatures of $\langle F, P \rangle$ -variables. \square

In addition to the existence of particular reachable models, we also require (as in [Diao4]) that model homomorphisms preserve the satisfaction of the local sentence upon which the query under consideration is based.

DEFINITION 3.3.11 (Preservation of satisfaction). Given a signature Σ and a signature of Σ -variables X in a generalized substitution system $\mathcal{GS}: \text{Sig} \rightarrow \text{SubstSys}$, an X -homomorphism $h: N_1 \rightarrow N_2$ is said to *preserve the satisfaction* of an X -sentence ρ if $N_1 \models_{\Sigma, X} \rho$ implies $N_2 \models_{\Sigma, X} \rho$.

Note that, in the case of QF-FOL_{\neq}^1 , all homomorphisms preserve, by definition, the satisfaction of all relational atoms. This property can be easily extended to arbitrary conjunctions of atoms, and thus to local queries of FOL_{\neq}^1 .

FACT 3.3.12. In the logic-programming framework FOL_{\neq}^1 , all homomorphisms preserve the satisfaction of all local queries.

THEOREM 3.3.13 (Herbrand's theorem). *For every logic program $\langle\langle \Sigma, \Gamma \rangle\rangle$ and every Σ -query $\exists X \cdot \rho$ such that (a) $\langle\langle \Sigma, \Gamma \rangle\rangle$ has an X -reachable initial model $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$, and (b) the satisfaction of ρ is preserved by X -homomorphisms, the following statements are equivalent:*

- 1 $\langle\langle \Sigma, \Gamma \rangle\rangle \models_{\Sigma}^{\text{lp}} \exists X \cdot \rho$.
- 2 $0_{\langle\langle \Sigma, \Gamma \rangle\rangle} \models_{\Sigma}^{\text{qs}} \exists X \cdot \rho$.
- 3 $\exists X \cdot \rho$ admits a $\langle\langle \Sigma, \Gamma \rangle\rangle$ -solution.

PROOF.

1 \Rightarrow 2. Obvious, since $0_{\langle\langle \Sigma, \Gamma \rangle\rangle} \models_{\Sigma}^{\text{lp}} \langle\langle \Sigma, \Gamma \rangle\rangle$.

2 \Rightarrow 3. By hypothesis, there exists an X -expansion $N_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ of $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ such that $N_{\langle\langle \Sigma, \Gamma \rangle\rangle} \models_{\Sigma, X} \rho$. Since $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ is assumed to be X -reachable, we know that there exists a substitution $\psi: X \rightarrow Y$ (corresponding to the model $N_{\langle\langle \Sigma, \Gamma \rangle\rangle}$) satisfying the two properties listed in [Definition 3.3.9](#). Therefore, we only need to show that $\langle\langle \Sigma, \Gamma \rangle\rangle \models_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho)$.

Let us thus consider a model M of $\langle\langle \Sigma, \Gamma \rangle\rangle$ and a Y -expansion N of M . Based on the initiality property of $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$, we obtain a (unique) Σ -homomorphism $h: 0_{\langle\langle \Sigma, \Gamma \rangle\rangle} \rightarrow M = N \upharpoonright_{\Sigma}$, which can be lifted, by taking into account the surjectivity of the map $_{\Sigma}: N_{\langle\langle \Sigma, \Gamma \rangle\rangle} / \text{Mod}_{\Sigma}(\psi) \rightarrow 0_{\langle\langle \Sigma, \Gamma \rangle\rangle} / \beta_{\Sigma, Y}$, to an X -homomorphism $f: N_{\langle\langle \Sigma, \Gamma \rangle\rangle} \rightarrow N \upharpoonright_{\psi}$. Since f preserves the satisfaction of ρ (by hypothesis) and $N_{\langle\langle \Sigma, \Gamma \rangle\rangle} \models_{\Sigma, X} \rho$, it follows that $N \upharpoonright_{\psi} \models_{\Sigma, X} \rho$, which implies, by the satisfaction condition for ψ , that $N \models_{\Sigma, Y} \psi(\rho)$. Hence, $M \models_{\Sigma}^{\text{qs}} \forall Y \cdot \psi(\rho)$.

3 \Rightarrow 1. Assume that $\psi: X \rightarrow Y$ is a $\langle\langle \Sigma, \Gamma \rangle\rangle$ -solution to $\exists X \cdot \rho$, and that M is a model of $\langle\langle \Sigma, \Gamma \rangle\rangle$. This means that $\langle\langle \Sigma, \Gamma \rangle\rangle \models_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho)$, and thus

$M \vDash_{\Sigma}^{\text{qs}} \forall Y \cdot \psi(\rho)$. In addition, Y is conservative, from which we deduce that there exists a Y -expansion N of M such that $N \vDash_{\Sigma, Y} \psi(\rho)$. It follows by the satisfaction condition for ψ that $N \upharpoonright_{\psi}$ is an X -expansion of M such that $N \upharpoonright_{\psi} \vDash_{\Sigma, X} \rho$. Consequently, $M \vDash_{\Sigma}^{\text{qs}} \exists X \cdot \rho$. \square

REMARK 3.3.14. The additional hypotheses referring to the existence of a reachable initial model of $\langle\langle \Sigma, \Gamma \rangle\rangle$ and to the preservation of the satisfaction of ρ by homomorphisms are used only in the proof of the ‘completeness’ part of [Theorem 3.3.13](#), that is for the implication $1 \Rightarrow 3$;⁴ the ‘soundness’ part, corresponding to the implication $3 \Rightarrow 1$, holds for every logic program $\langle\langle \Sigma, \Gamma \rangle\rangle$ (defined over an arbitrary language) and every Σ -query $\exists X \cdot \rho$.

3.3.2 OPERATIONAL SEMANTICS

One of the most important (and distinctive) features of the concept of logic-programming language is that it allows us to make effective use of the resolution inference rule and, in a very natural way, to give an operational semantics to logic programs. Consider, for instance, the logic program $\langle\langle F_{\text{NAT}}, P_{\text{NAT}} \rangle, \Gamma \rangle$ described in [Example 3.3.6](#) and the query $\exists \{X_1\} \cdot \text{add}(s \theta, s \theta, X_1)$, sometimes written as

$$\frac{}{X_1} \text{add}(s \theta, s \theta, X_1)$$

using a notation similar to that of clauses. We can compute a solution to the query $\exists \{X_1\} \cdot \text{add}(s \theta, s \theta, X_1)$ using the clauses of Γ and the goal-directed rules of $\mathcal{FO}\mathcal{L}_{\neq}^1$ as follows: we first derive the query $\exists \{X_2\} \cdot \text{add}(\theta, s \theta, X_2)$, based on the second clause of Γ , the substitutions $\theta_1: \{X_1\} \rightarrow \{X_2\}$ and $\psi_1: \{M, N, P\} \rightarrow \{X_2\}$ given by $X_1 \mapsto s X_2$, $M \mapsto \theta$, $N \mapsto s \theta$ and $P \mapsto X_2$, and the following goal-directed rule over $\{X_2\}$;

$$\frac{\frac{\text{add}(s \theta, s \theta, s X_2), \text{add}(\theta, s \theta, X_2) \Rightarrow \text{add}(s \theta, s \theta, s X_2)}{\theta_1(\text{add}(s \theta, s \theta, X_1))} \quad \frac{}{\psi_1(\text{add}(M, N, P) \Rightarrow \text{add}(s M, N, s P))}}{\vDash_{\langle F_{\text{NAT}}, P_{\text{NAT}} \rangle, X_2} \text{add}(\theta, s \theta, X_2)}$$

by iterating these constructions, we can further derive the trivial query $\exists \emptyset \cdot \text{true}$ using the first clause of Γ , the substitutions $\theta_2: \{X_2\} \rightarrow \emptyset$ and $\psi_2: \{M\} \rightarrow \emptyset$ given by $X_2 \mapsto s \theta$ and $M \mapsto s \theta$, and the goal-directed rule over \emptyset detailed below;

$$\frac{\frac{\text{add}(\theta, s \theta, s \theta), \text{true} \Rightarrow \text{add}(\theta, s \theta, s \theta)}{\theta_2(\text{add}(\theta, s \theta, X_2))} \quad \frac{}{\psi_2(\text{true} \Rightarrow \text{add}(\theta, M, M))}}{\vDash_{\langle F_{\text{NAT}}, P_{\text{NAT}} \rangle, \emptyset} \text{true}}$$

⁴ Moreover, it would suffice to assume that the model $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ is weakly initial.

finally, we compose the substitutions θ_1 and θ_2 (which are usually computed through term unification) to obtain a solution to $\exists\{X_1\} \cdot \text{add}(s \theta, s \theta, X_1)$.

The procedure outlined above does not depend on any particular details of $(\mathcal{FOL}_{\neq}^1)^{\text{pres}}$; in fact, it admits a straightforward formalization within the abstract framework of logic-programming languages.

DEFINITION 3.3.15 (Resolution). Let $\exists X_1 \cdot \rho_1$ be a query and $\forall Y_1 \cdot \gamma_1$ a clause over a signature Σ . A Σ -query $\exists X_2 \cdot \rho_2$ is said to be *derived by resolution* from $\exists X_1 \cdot \rho_1$ and $\forall Y_1 \cdot \gamma_1$ using the *computed substitution* $\theta_1: X_1 \rightarrow X_2$ if there exists a substitution $\psi_1: Y_1 \rightarrow X_2$ such that $\theta_1(\rho_1), \psi_1(\gamma_1) \Vdash_{\Sigma, X_2} \rho_2$.

$$\frac{\exists X_1 \cdot \rho_1 \quad \forall Y_1 \cdot \gamma_1}{\exists X_2 \cdot \rho_2} \theta_1$$

UNIFICATION. Note that resolution describes not only the derivation of new and (presumably) simpler queries from appropriate pairs of queries and clauses, but also how partial answers to the original queries can be computed by means of *sentence unification*. In this sense, for any signature Σ and signatures of Σ -variables X_1 and Y_1 , an ordered pair $\langle \rho_1, \gamma_1 \rangle$ of sentences $\rho_1 \in \mathcal{Q}_{\Sigma}(X_1)$ and $\gamma_1 \in \mathcal{C}_{\Sigma}(Y_1)$ is *unifiable* if there exists a pair $\langle \theta_1, \psi_1 \rangle$ of substitutions $\theta_1: X_1 \rightarrow X_2$ and $\psi_1: Y_1 \rightarrow X_2$, called the *unifier* of ρ_1 and γ_1 , such that $\theta_1(\rho_1), \psi_1(\gamma_1) \Vdash_{\Sigma, X_2} \rho_2$ for some X_2 -sentence $\rho_2 \in \mathcal{Q}_{\Sigma}(X_2)$.

It is also possible to distinguish between the various levels of generality of the unifiers. Given two unifiers $\langle \theta_1, \psi_1 \rangle$ and $\langle \theta'_1, \psi'_1 \rangle$ of ρ_1 and γ_1 as depicted below, we say that $\langle \theta'_1, \psi'_1 \rangle$ is an *instance* of $\langle \theta_1, \psi_1 \rangle$, or that $\langle \theta_1, \psi_1 \rangle$ is *more general* than $\langle \theta'_1, \psi'_1 \rangle$, if there exists a substitution θ such that $\theta_1 \circ \theta = \theta'_1$ and $\psi_1 \circ \theta = \psi'_1$. Given this, the unifiers of ρ_1 and γ_1 can be defined as objects of a subcategory of the category of cospans of Σ -substitutions.

$$\begin{array}{ccccc} & & X_2 & & \\ & \theta_1 \rightarrow & & \leftarrow \psi_1 & \\ X_1 & & \downarrow \theta & & Y_1 \\ & \theta'_1 \rightarrow & X'_2 & \leftarrow \psi'_1 & \end{array}$$

It should be noted however that, under the present formalization, the most general unifiers, defined as initial objects in their corresponding category (along the lines of [Gog89]), cannot be guaranteed to exist. Even in the case of the logic-programming framework \mathcal{FOL}_{\neq}^1 , one cannot find, for example, a most general unifier of the sentences $\text{add}(s \theta, s \theta, X_1) \wedge \text{add}(s s \theta, \theta, X_1)$ and $\text{add}(M, N, P) \Rightarrow \text{add}(s M, N, s P)$ – although one exists for $\text{add}(s \theta, s \theta, X_1)$

and $\text{add}(M, N, P) \Rightarrow \text{add}(s M, N, s P)$, as well as for $\text{add}(s s \emptyset, \emptyset, X_1)$ and $\text{add}(M, N, P) \Rightarrow \text{add}(s M, N, s P)$. This does not restrict the applicability of the theory proposed here, because our abstract notion of resolution actually corresponds to an extended variant of first-order resolution, in which any unifier may give rise to a derivation, not just the most general ones.

The scope of [Definition 3.3.15](#) can be easily broadened to accommodate sets of clauses: given a set Γ of clauses over a signature Σ , a Σ -query $\exists X_2 \cdot \rho_2$ is said to be *derived by resolution* from $\exists X_1 \cdot \rho_1$ and Γ using the *computed substitution* $\theta_1: X_1 \rightarrow X_2$, written

$$\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta_1} \exists X_2 \cdot \rho_2,$$

if $\exists X_2 \cdot \rho_2$ can be derived by resolution from $\exists X_1 \cdot \rho_1$ and $\forall Y_1 \cdot \gamma_1$ using the computed substitution θ_1 , for some Σ -clause $\forall Y_1 \cdot \gamma_1 \in \Gamma$. This gives us a family $(\longrightarrow_{\Gamma, \theta})_{\theta \in \text{Subst}_{\Sigma}}$ of *one-step derivation relations generated by Γ* , whose union is denoted by \longrightarrow_{Γ} .

The rest of this section is devoted to the composition of one-step derivations, which will be shown to provide a general procedure for computing answers to queries. To this purpose, let us first investigate the soundness of the one-step derivation relations with respect to the concept of solution.

PROPOSITION 3.3.16. *Let $\langle\langle \Sigma, \Gamma \rangle\rangle$ be a logic program, and $\exists X_1 \cdot \rho_1$ and $\exists X_2 \cdot \rho_2$ two Σ -queries. For every inference step $\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta_1} \exists X_2 \cdot \rho_2$ and every solution $\psi: X_2 \rightarrow Y$ to $\exists X_2 \cdot \rho_2$, the substitution $\theta_1 \circ \psi$ is a solution to $\exists X_1 \cdot \rho_1$.*

PROOF. Assume that $\exists X_2 \cdot \rho_2$ is derived by resolution from $\exists X_1 \cdot \rho_1$ and Γ using the substitution $\theta_1: X_1 \rightarrow X_2$, and that $\psi: X_2 \rightarrow Y$ is a $\langle\langle \Sigma, \Gamma \rangle\rangle$ -solution to $\exists X_2 \cdot \rho_2$. The latter implies that the codomain Y of $\theta_1 \circ \psi$ is conservative. Therefore, we only need to prove that $\langle\langle \Sigma, \Gamma \rangle\rangle \models_{\Sigma}^{\text{lp}} \forall Y \cdot (\theta_1 \circ \psi)(\rho_1)$.

Let M be a model of $\langle\langle \Sigma, \Gamma \rangle\rangle$ and N a Y -expansion of M . By the definition of the one-step derivation relation $\longrightarrow_{\Gamma, \theta_1}$, there exists a clause $\forall Y_1 \cdot \gamma_1 \in \Gamma$ and a substitution $\psi_1: Y_1 \rightarrow X_2$ such that $\theta_1(\rho_1), \psi_1(\gamma_1) \Vdash_{\Sigma, X_2} \rho_2$. This allows us to deduce, based on the soundness of the goal-directed rules, that $\{\rho_2, \psi_1(\gamma_1)\} \models_{\Sigma, X_2} \theta_1(\rho_1)$. Furthermore, since the semantic consequence is preserved by translation along signature morphisms (which, in turn, is an immediate consequence of the invariance of truth under change of notation), we obtain $\{\psi(\rho_2), (\psi_1 \circ \psi)(\gamma_1)\} \models_{\Sigma, Y} (\theta_1 \circ \psi)(\rho_1)$. This means that, to conclude our proof, it suffices to show that N satisfies both $\psi(\rho_2)$ and $(\psi_1 \circ \psi)(\gamma_1)$.

In the case of the first relation, since, by hypothesis, the substitution ψ is a $\langle\langle \Sigma, \Gamma \rangle\rangle$ -solution to the query $\exists X_2 \cdot \rho_2$, we have $\langle\langle \Sigma, \Gamma \rangle\rangle \models_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho_2)$. It

follows that $M \vDash_{\Sigma}^{\text{qs}} \forall Y \cdot \psi(\rho_2)$, which further implies $N \vDash_{\Sigma, Y} \psi(\rho_2)$.

In the case of the second relation, by the general properties of substitution systems, we know that $N \uparrow_{\psi_1 \wp \psi}$ is a Y_1 -expansion of M . As a result, $N \uparrow_{\psi_1 \wp \psi} \vDash_{\Sigma, Y_1} \gamma_1$, because $M \vDash_{\Sigma}^{\text{qs}} \forall Y_1 \cdot \gamma_1$, and thus, by the satisfaction condition for $\psi_1 \wp \psi$, $N \vDash_{\Sigma, X_2} (\psi_1 \wp \psi)(\gamma_1)$. \square

The search for (computed) solutions to a given query proceeds by means of a sequence of one-step derivations, each of which contributes towards the final answer through its corresponding computed substitution.

DEFINITION 3.3.17 (Derivation). For any set Γ of clauses over Σ , and any two Σ -queries $\exists X_1 \cdot \rho_1$ and $\exists X_n \cdot \rho_n$, a Γ -*derivation* of $\exists X_n \cdot \rho_n$ from $\exists X_1 \cdot \rho_1$ is a chain of one-step derivations

$$\begin{aligned} \exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta_1} \exists X_2 \cdot \rho_2 \longrightarrow_{\Gamma, \theta_2} \exists X_3 \cdot \rho_3 \cdots \\ \exists X_{n-1} \cdot \rho_{n-1} \longrightarrow_{\Gamma, \theta_{n-1}} \exists X_n \cdot \rho_n. \end{aligned}$$

Whenever such a chain exists, the query $\exists X_n \cdot \rho_n$ is said to be *derived* from $\exists X_1 \cdot \rho_1$ and Γ using the *computed substitution* $\theta = \theta_1 \wp \theta_2 \wp \cdots \wp \theta_{n-1}$; in this case, we write $\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta}^* \exists X_n \cdot \rho_n$. By definition, if the chain depicted above is empty, $\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, 1_{X_1}}^* \exists X_1 \cdot \rho_1$. We obtain in this way a family $(\longrightarrow_{\Gamma, \theta}^*)_{\theta \in \text{Subst}_{\Sigma}}$ of *derivation relations generated by Γ* .

REMARK 3.3.18. The union of $(\longrightarrow_{\Gamma, \theta}^*)_{\theta \in \text{Subst}_{\Sigma}}$ is the reflexive and transitive closure of the one-step derivation relation \longrightarrow_{Γ} .

Proposition 3.3.16 can be generalized without difficulty from one-step to arbitrary derivations by induction on the length of the derivation.

COROLLARY 3.3.19. Let $\langle\langle \Sigma, \Gamma \rangle\rangle$ be a logic program, and $\exists X_1 \cdot \rho_1$ and $\exists X_n \cdot \rho_n$ two Σ -queries. For every derivation $\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta}^* \exists X_n \cdot \rho_n$ and every solution $\psi: X_n \rightarrow Y$ to $\exists X_n \cdot \rho_n$, the substitution $\theta \wp \psi$ is a solution to $\exists X_1 \cdot \rho_1$. \square

All we require now in order to define computed answers is a concept of trivial query, which is meant to characterize the successful termination of the search procedure.

DEFINITION 3.3.20 (Trivial query). Given a signature Σ , a Σ -query $\exists Y \cdot \top$ is said to be *trivial* if Y is conservative and every Y -model satisfies \top .

FACT 3.3.21. In the logic-programming framework \mathcal{FOL}_{\neq}^1 , a query is trivial if and only if it is of the form $\exists Y \cdot \text{true}$.⁵

⁵ Note that, to ensure the conservativity of non-empty signatures of variables, the considered first-order signature has to define at least one constant.

An immediate consequence of the definition above is that every trivial query corresponds to a local tautology (and, even more, the query itself is a tautology). As a result, for every logic program $\langle\langle\Sigma, \Gamma\rangle\rangle$ and every trivial Σ -query $\exists Y \cdot \top$, we have $\langle\langle\Sigma, \Gamma\rangle\rangle \vDash_{\Sigma}^{\text{lp}} \forall Y \cdot \top$. This amounts to describing the identity 1_Y as a $\langle\langle\Sigma, \Gamma\rangle\rangle$ -solution to $\exists Y \cdot \top$.

FACT 3.3.22. Every trivial query $\exists Y \cdot \top$ over the signature Σ of a logic program $\langle\langle\Sigma, \Gamma\rangle\rangle$ admits a $\langle\langle\Sigma, \Gamma\rangle\rangle$ -solution: the identity substitution 1_Y .

DEFINITION 3.3.23 (Computed answer). Given a program $\langle\langle\Sigma, \Gamma\rangle\rangle$, a Σ -substitution $\theta: X \rightarrow Y$ is a *computed $\langle\langle\Sigma, \Gamma\rangle\rangle$ -answer* to a Σ -query $\exists X \cdot \rho$ if there exists a trivial query $\exists Y \cdot \top$ such that $\exists X \cdot \rho \rightarrow_{\Gamma, \theta}^* \exists Y \cdot \top$.

Corollary 3.3.19 and **Fact 3.3.22** lead to our first main result related to the operational semantics of abstract logic-programming languages.

THEOREM 3.3.24 (Soundness of resolution). *Consider a logic program $\langle\langle\Sigma, \Gamma\rangle\rangle$ and a Σ -query $\exists X \cdot \rho$. Then every computed $\langle\langle\Sigma, \Gamma\rangle\rangle$ -answer to $\exists X \cdot \rho$ is also a solution to $\exists X \cdot \rho$.* \square

3.3.3 COMPLETENESS

As expected, completeness is more difficult to obtain, and requires additional hypotheses. To simplify the proof, we consider two lemmas, each of which introduces a new property to be satisfied by the query at hand or by the considered logic program. The first lemma allows us to derive by resolution any translation (along a substitution) of a given query, which is reminiscent of the well-known lifting lemma of conventional logic programming [see e.g. Av82; Llo87]; the second lemma reduces the search for solutions to a sequence of elementary inferences in a local institution of substitutions – which, in the case of first-order logic-programming, involves no quantifiers.

DEFINITION 3.3.25 (Identity clause). Let $\exists X \cdot \rho$ be a query over a signature Σ . A Σ -clause $\forall X \cdot \gamma$ is an *identity* of $\exists X \cdot \rho$ if γ is a tautology and $\rho, \gamma \Vdash_{\Sigma, X} \rho$.

FACT 3.3.26. In \mathcal{FOL}_{\neq}^1 , every non-trivial query $\exists X \cdot \bigwedge Q$ admits an identity $\forall X \cdot \pi(t_1, \dots, t_n) \Rightarrow \pi(t_1, \dots, t_n)$, where $\pi(t_1, \dots, t_n)$ is an atom in Q .

LEMMA 3.3.27. *Consider a signature Σ , a Σ -query $\exists X \cdot \rho$, and an identity $\forall X \cdot \gamma$ of $\exists X \cdot \rho$. For every Σ -substitution $\psi: X \rightarrow Y$ there exists a one-step derivation of $\exists Y \cdot \psi(\rho)$ from $\exists X \cdot \rho$ and $\forall X \cdot \gamma$ having ψ as the computed substitution.*

$$\exists X \cdot \rho \rightarrow_{\{\forall X \cdot \gamma\}, \psi} \exists Y \cdot \psi(\rho)$$

$\longrightarrow_{\Gamma}^*$ and \Vdash^* ; because of this, in the following we will focus entirely on the more interesting case that corresponds to the induction step.

For the ‘if’ part, assume there exists an X_n -query ρ_2 and an X_n -clause $\psi_1(\gamma_1) \in X_n(\Gamma)$, further indicating the existence of a clause $\forall Y_1 \cdot \gamma_1$ in Γ and of a substitution $\psi_1: Y_1 \rightarrow X_n$ such that $\theta(\rho_1), \psi_1(\gamma_1) \Vdash_{\Sigma, X_n} \rho_2$ and $\rho_2, X_n(\Gamma) \Vdash_{\Sigma, X_n}^* \rho_n$. Based on the definition of resolution and on the induction hypothesis, we deduce that $\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta} \exists X_n \cdot \rho_2$ and $\exists X_n \cdot \rho_2 \longrightarrow_{\Gamma, 1_{X_n}}^* \exists X_n \cdot \rho_n$; therefore, by the composition of these derivations, we obtain $\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta}^* \exists X_n \cdot \rho_n$.

For the ‘only if’ part, let $\theta_1 \circlearrowleft \theta_2$ be a factorization of the substitution θ such that $\exists X_1 \cdot \rho_1 \longrightarrow_{\Gamma, \theta_1} \exists X_2 \cdot \rho_2$ and $\exists X_2 \cdot \rho_2 \longrightarrow_{\Gamma, \theta_2}^* \exists X_n \cdot \rho_n$. Then, by the induction hypothesis, $\theta_2(\rho_2), X_n(\Gamma) \Vdash_{\Sigma, X_n}^* \rho_n$. We also know, by the definition of resolution, that there exists a clause $\forall Y_1 \cdot \gamma_1 \in \Gamma$ and a substitution $\psi_1: Y_1 \rightarrow X_2$ such that $\theta_1(\rho_1), \psi_1(\gamma_1) \Vdash_{\Sigma, X_2} \rho_2$. Based on the functoriality of \Vdash^* , this implies that $(\theta_1 \circlearrowleft \theta_2)(\rho_1), (\psi_1 \circlearrowleft \theta_2)(\gamma_1) \Vdash_{\Sigma, X_n} \theta_2(\rho_2)$, and thus, since $(\psi_1 \circlearrowleft \theta_2)(\gamma_1)$ belongs to $X_n(\Gamma)$ and $\theta_2(\rho_2), X_n(\Gamma) \Vdash_{\Sigma, X_n}^* \rho_n$, we conclude that $\theta(\rho_1), X_n(\Gamma) \Vdash_{\Sigma, X_n}^* \rho_n$. \square

Let us recall that, in general, the derivation of queries proceeds by selecting, at each step, a new rule – over a new signature of variables – to be applied to the current goal. Provided that we know the result of the derivation, [Proposition 3.3.30](#) allows us to reduce the derivation of queries to applications of goal-directed rules that are defined over the same signature of variables. In view of this characterization, the soundness of resolution can be interpreted locally as described in the following corollary.

COROLLARY 3.3.31. *Let $\langle\langle \Sigma, \Gamma \rangle\rangle$ be a logic program, X a conservative signature of Σ -variables, and ρ an X -query. Then for every trivial X -query \top ,*

$$\rho, X(\Gamma) \Vdash_{\Sigma, X}^* \top \quad \text{implies} \quad \langle\langle \Sigma, \Gamma \rangle\rangle \vDash_{\Sigma}^{\text{lp}} \forall X \cdot \rho.$$

PROOF. Assume \top to be a trivial query over X such that $\rho, X(\Gamma) \Vdash_{\Sigma, X}^* \top$. By [Proposition 3.3.30](#), it follows that $\exists X \cdot \rho \longrightarrow_{\Gamma, 1_X}^* \exists X \cdot \top$. This means that the identity 1_X is a computed $\langle\langle \Sigma, \Gamma \rangle\rangle$ -answer to $\exists X \cdot \rho$ (because X is conservative), and thus, by [Theorem 3.3.24](#), 1_X is also a solution to $\exists X \cdot \rho$. As a result, $\langle\langle \Sigma, \Gamma \rangle\rangle \vDash_{\Sigma}^{\text{lp}} \forall X \cdot \rho$. \square

With respect to the completeness of resolution, we are interested in the converse of the implication discussed in the corollary above. This may hold for certain programs in logic-programming languages of interest, but it cannot be guaranteed in general.

DEFINITION 3.3.32 (Query-completeness). A logic program $\langle\langle\Sigma, \Gamma\rangle\rangle$ is said to be *query-complete* if for every conservative signature of Σ -variables X and every X -query ρ ,

$$\langle\langle\Sigma, \Gamma\rangle\rangle \vDash_{\Sigma}^{\text{lp}} \forall X \cdot \rho \quad \text{implies} \quad \rho, X(\Gamma) \Vdash_{\Sigma, X}^* \top$$

for some trivial X -query \top . In addition, the logic-programming language \mathcal{L} is *query-complete* when all logic programs of \mathcal{L} have this property.

The following result is well known in the literature; it can be found, for instance, in a slightly different form, in [Llo87]. The result is based on the observation that for every signature of variables X over a first-order signature $\langle F, P \rangle$, and for every set Γ of clauses over $\langle F, P \rangle$, the X -expansion N of the free $\langle\langle F, P \rangle, \Gamma\rangle$ -model over X given by $N_x = x$ for every $x \in X$ satisfies a relational atom ρ if and only if there exist a clause $\forall Y \cdot \bigwedge H \Rightarrow C$ in Γ and a substitution $\psi: Y \rightarrow X$ such that $\psi(C) = \rho$ and $N \vDash_{\langle F, P \rangle, X} \bigwedge \psi(H)$.

Therefore, if $\langle\langle F, P \rangle, \Gamma\rangle \vDash_{\langle F, P \rangle}^{\text{lp}} \forall X \cdot \rho$, then $N \vDash_{\langle F, P \rangle, X} \rho$, and thus there exists a clause $\psi(\gamma) \in X(\Gamma)$ such that $\rho, \psi(\gamma) \Vdash_{\Sigma, X} \rho_1$, where $N \vDash_{\langle F, P \rangle, X} \rho_1$. By iterating this construction, we obtain an alternating sequence of X -queries and clauses as in Definition 3.3.29; furthermore, since $N \upharpoonright_{\langle F, P \rangle}$ is the free $\langle\langle F, P \rangle, \Gamma\rangle$ -model over X , it is always possible to assemble such a sequence that terminates in a trivial query – which, in the case of \mathcal{FOL}_{\neq}^1 , is simply the sentence *true* (see Fact 3.3.21). Consequently, $\rho, X(\Gamma) \Vdash_{\Sigma, X}^* \text{true}$. A more detailed presentation of this result can be found, for example, in [Llo87].

PROPOSITION 3.3.33. *The logic-programming language $(\mathcal{FOL}_{\neq}^1)^{\text{pres}}$ of theory presentations over \mathcal{FOL}_{\neq}^1 is query-complete.* \square

Our second lemma is now a direct consequence of Proposition 3.3.30.

LEMMA 3.3.34. *Suppose that $\langle\langle\Sigma, \Gamma\rangle\rangle$ is a query-complete logic program. Then for every Σ -query $\exists X \cdot \rho$ such that X is conservative and $\langle\langle\Sigma, \Gamma\rangle\rangle \vDash_{\Sigma}^{\text{lp}} \forall X \cdot \rho$, there exists a trivial query $\exists X \cdot \top$ that can be derived from $\exists X \cdot \rho$ and Γ , with the identity substitution 1_X as the computed answer.*

$$\exists X \cdot \rho \longrightarrow_{\Gamma, 1_X}^* \exists X \cdot \top$$

PROOF. Let $\langle\langle\Sigma, \Gamma\rangle\rangle$ be a logic program as above and $\exists X \cdot \rho$ a Σ -query such that $\langle\langle\Sigma, \Gamma\rangle\rangle \vDash_{\Sigma}^{\text{lp}} \forall X \cdot \rho$. By query-completeness, there exists a trivial X -query \top such that $\rho, X(\Gamma) \Vdash_{\Sigma, X}^* \top$. Hence, by Proposition 3.3.30, we can derive the trivial query $\exists X \cdot \top$ from $\exists X \cdot \rho$ and Γ using the substitution 1_X . \square

THEOREM 3.3.35 (Completeness of resolution). *Let $\langle\langle\Sigma, \Gamma\rangle\rangle$ be a query-complete logic program and $\exists X \cdot \rho$ a Σ -query that admits an identity $\forall X \cdot \gamma \in \Gamma$. Then every $\langle\langle\Sigma, \Gamma\rangle\rangle$ -solution to $\exists X \cdot \rho$ is also a computed $\langle\langle\Sigma, \Gamma\rangle\rangle$ -answer to $\exists X \cdot \rho$.*

PROOF. Let $\psi: X \rightarrow Y$ be a $\langle\langle\Sigma, \Gamma\rangle\rangle$ -solution to $\exists X \cdot \rho$. By [Lemma 3.3.27](#), we know that $\exists Y \cdot \psi(\rho)$ can be derived from $\exists X \cdot \rho$ and its identity, $\forall X \cdot \gamma$, using the substitution ψ . Therefore, since $\forall X \cdot \gamma \in \Gamma$, the query $\exists Y \cdot \psi(\rho)$ can also be derived from $\exists X \cdot \rho$ and Γ .

$$\exists X \cdot \rho \longrightarrow_{\Gamma, \psi} \exists Y \cdot \psi(\rho)$$

In addition, by [Definition 3.3.8](#), the signature of variables Y is conservative, and $\langle\langle\Sigma, \Gamma\rangle\rangle \vDash_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho)$. Hence, by [Lemma 3.3.34](#), we can derive a trivial query $\exists Y \cdot \top$ from $\exists Y \cdot \psi(\rho)$ and Γ using the computed substitution 1_Y .

$$\exists Y \cdot \psi(\rho) \longrightarrow_{\Gamma, 1_Y}^* \exists Y \cdot \top$$

Composing the two derivation chains outlined above yields a derivation of $\exists Y \cdot \top$ from $\exists X \cdot \rho$ using the substitution ψ . Consequently, ψ is a computed answer to $\exists X \cdot \rho$. \square

4

CONNECTION WITH INSTITUTIONS

This chapter is devoted to the formalization of the connection between the institution- and the substitution-system-based approach to logic programming. For this purpose, we investigate a number of features of institutions, like the existence of a quantification space or of representable substitutions, under which they give rise to suitable generalized substitution systems. Building on these results, we further show how the original institution-independent versions of Herbrand's theorem can be obtained as concrete instances of the general result developed in [Section 3.3.1](#).

Before we embark on the study of institution-based abstract logic-programming languages, let us briefly recall the logical systems that underlie two of the most prominent examples of (concrete) logic-programming languages examined in the context of institutions: first-order and higher-order equational logic programming [see e.g. [GM86](#); [Mes92](#); and also [OPE97](#)]. These will form the main reference points that we will use to illustrate the concepts and properties discussed in the subsequent sections of this chapter.

4.1 EQUATIONAL LOGIC PROGRAMMING

Equational logic programming integrates the machinery of its relational counterpart within algebraic specification in order to solve equations over abstract data types that are provided by given specifications. This is accomplished by replacing (a) the single-sorted relational variant of first-order logic (without equality) with the many-sorted equational variant of first-order logic (with equality) or with higher-order logic with Henkin semantics, (b) resolution with paramodulation, and (c) presentations (in the definition of logic programs) with program modules that are adequate for defining abstract data types. In this setting, the computation of the sum of $s\ 0$ and $s\ 0$ considered in [Section 3.3](#) can be triggered by a query of the form

$$\frac{}{\vdash_{X_1: \text{Nat}} s\ 0 + s\ 0 = X_1}$$

meant to be solved over a logic program that consists of two modules: NAT, which defines natural numbers as an abstract data type, and ADD, which specifies the addition of natural numbers in the usual inductive manner.

```

module NAT = free
  sort Nat
  op 0: → Nat
  op s_: Nat → Nat

module ADD = NAT then
  op _ + _: Nat Nat → Nat
  clause 0 + M = M ←M: Nat
  clause (s M) + N = s (M + N) ←M,N: Nat

```

4.1.1 FIRST-ORDER EQUATIONAL LOGIC

First-order equational logic programming is defined over the institution QF-FOL_\equiv of the quantifier-free fragment of many-sorted first-order equational logic. Since most of the definitions and properties to check are straightforward adaptations of the definitions and properties discussed in Section 3.1 for QF-FOL_\neq^1 to the many-sorted equational setting of QF-FOL_\equiv , we only shortly review some of the most important concepts that we need. A more in-depth discussion of the components of QF-FOL_\equiv can be found, for example, in [GM87],¹ or in the recent monographs [Diao8; ST11].

SIGNATURES. The *signatures* of QF-FOL_\equiv are pairs $\langle S, F \rangle$, where S is a (finite) set of *sorts* and F is a family $(F_{w \rightarrow s})_{w \in S^*, s \in S}$ of (finite) sets of *operation symbols* indexed by *arities* and *sorts*. *Signature morphisms* $\varphi: \langle S, F \rangle \rightarrow \langle S', F' \rangle$ are defined by functions $\varphi^{\text{st}}: S \rightarrow S'$ between the sets of sorts and by families of functions $\varphi_{w \rightarrow s}^{\text{op}}: F_{w \rightarrow s} \rightarrow F'_{\varphi^{\text{st}}(w) \rightarrow \varphi^{\text{st}}(s)}$, for $w \in S^*$ and $s \in S$, between the sets of operation symbols.

SENTENCES, MODELS, AND THE SATISFACTION RELATION. For every many-sorted signature $\langle S, F \rangle$ and every sort $s \in S$, the set $T_{F,s}$ of *F-terms* of sort s is the least set such that $\sigma(t_1, \dots, t_n): s \in T_{F,s}$ for all $\sigma \in F_{s_1 \dots s_n \rightarrow s}$ and $t_i \in T_{F,s_i}$. Similarly to the relational setting, the *sentences* over a QF-FOL_\equiv -signature $\langle S, F \rangle$ are built from *equational atoms* $l = r$, where l and r are *F-terms* having the same sort, by iteration of the usual Boolean connectives.

¹ It should be noted that in [GM87] the authors consider a more general setting of order-sorted equational logic, with subsorts and overloading of operation symbols.

The *models*, or *algebras*, M of $\langle S, F \rangle$ interpret each sort $s \in S$ as a set M_s , called the *carrier set* of s in M , and each operation symbol $\sigma \in F_{s_1 \dots s_n \rightarrow s}$ as a function $M_\sigma: M_{s_1} \times \dots \times M_{s_n} \rightarrow M_s$. *Homomorphisms* $h: M_1 \rightarrow M_2$ are families of functions $(h_s: M_{1,s} \rightarrow M_{2,s})_{s \in S}$ such that $h_s(M_{1,\sigma}(m_1, \dots, m_n)) = M_{2,\sigma}(h_{s_1}(m_1), \dots, h_{s_n}(m_n))$ for all $\sigma \in F_{s_1 \dots s_n \rightarrow s}$ and $m_i \in M_{s_i}$.

$$\begin{array}{ccc} M_{1,s_1} \times \dots \times M_{1,s_n} & \xrightarrow{M_{1,\sigma}} & M_{1,s} \\ h_{s_1} \times \dots \times h_{s_n} \downarrow & & \downarrow h_s \\ M_{2,s_1} \times \dots \times M_{2,s_n} & \xrightarrow{M_{2,\sigma}} & M_{2,s} \end{array}$$

Finally, the *satisfaction relation* is defined by induction on the structure of sentences, based on the evaluation of terms in models. For instance, an $\langle S, F \rangle$ -model M satisfies an equational atom $l = r$ if and only if $M_l = M_r$, that is if the terms l and r yield the same value in M .

4.1.2 HIGHER-ORDER LOGIC WITH HENKIN SEMANTICS

Building on the work of Russell on mathematical logic and the theory of types [see Ruso8], higher-order logic with Henkin semantics has been developed in [Chu40; Hen50] and later integrated into the framework of algebraic specifications in [MTW87]. As in [MTW87] and in more recent institution-theoretic works such as [ST11; Dia12b; Tu13] we use here a simplified version of higher-order logic that only takes into account λ -free terms. This does not limit its expressive power since for any term $\lambda(x: s). t$ we can define a new constant σ and a universal sentence of the form $\forall \{x: s\} \cdot \sigma x = t$ – a detailed presentation of the encoding of higher-order logic with λ -abstraction into its λ -free substitution can be found in [Gai14].

Analogously to first-order equational logic programming, for the results presented in the following sections it suffices to consider the quantifier-free fragment of higher-order logic, whose institution we denote by QF-HNK .²

SIGNATURES. A *higher-order signature* $\langle S, F \rangle$ consists of a set S of *basic types*, or *sorts*, and a family $(F_s)_{s \in \vec{S}}$ of sets of *constant-operation symbols* indexed by S -types, where \vec{S} is the least set for which $S \subseteq \vec{S}$ and $s_1 \rightarrow s_2 \in \vec{S}$ whenever $s_1, s_2 \in \vec{S}$. *Signature morphisms* $\varphi: \langle S, F \rangle \rightarrow \langle S', F' \rangle$ comprise functions $\varphi^{\text{st}}: S \rightarrow \vec{S}'$ and $\varphi_s^{\text{op}}: F_s \rightarrow F'_{\varphi^{\text{type}}(s)}$, for $s \in \vec{S}$, where $\varphi^{\text{type}}: \vec{S} \rightarrow \vec{S}'$ is the canonical extension of φ^{st} given by $\varphi^{\text{type}}(s_1 \rightarrow s_2) = \varphi^{\text{type}}(s_1) \rightarrow \varphi^{\text{type}}(s_2)$.

² Note that the universal sentences needed for encoding λ -terms can still be defined as Horn clauses of the logic programs under consideration.

SENTENCES, MODELS, AND THE SATISFACTION RELATION. Given a signature $\langle S, F \rangle$, the family $(T_{F,s})_{s \in \vec{S}}$ of F -terms is the least family of sets such that $\sigma: s \in T_{F,s}$ for all types $s \in \vec{S}$ and constants $\sigma \in F_s$, and $(t \ t_1) \in T_{F,s_2}$ for all terms $t \in T_{F,s_1 \rightarrow s_2}$ and $t_1 \in T_{F,s_1}$. As in first-order logic, the *sentences* over $\langle S, F \rangle$ are built from *equational atoms* $l = r$, where l and r are terms in $T_{F,s}$ for some type $s \in \vec{S}$, by repeated applications of the Boolean connectives.

The *models* M of a higher-order signature $\langle S, F \rangle$ interpret the types $s \in \vec{S}$ as sets M_s , the constant symbols $\sigma \in F_s$ as elements $M_\sigma \in M_s$, and define injective maps $\llbracket - \rrbracket_{s_1 \rightarrow s_2}^M: M_{s_1 \rightarrow s_2} \rightarrow [M_{s_1} \rightarrow M_{s_2}]$, where $[M_{s_1} \rightarrow M_{s_2}]$ denotes the set of functions from M_{s_1} to M_{s_2} , for any two types $s_1, s_2 \in \vec{S}$. *Model homomorphisms* $h: M_1 \rightarrow M_2$ are families of functions $(h_s: M_{1,s} \rightarrow M_{2,s})_{s \in \vec{S}}$ such that $h_s(M_{1,\sigma}) = M_{2,\sigma}$ for every type $s \in \vec{S}$ and operation symbol $\sigma \in F_s$, and the following diagram commutes for all $s_1, s_2 \in \vec{S}$ and $f \in M_{1,s_1 \rightarrow s_2}$.

$$\begin{array}{ccc} M_{1,s_1} & \xrightarrow{\llbracket f \rrbracket_{s_1 \rightarrow s_2}^{M_1}} & M_{1,s_2} \\ h_{s_1} \downarrow & & \downarrow h_{s_2} \\ M_{2,s_1} & \xrightarrow{\llbracket h_{s_1 \rightarrow s_2}(f) \rrbracket_{s_1 \rightarrow s_2}^{M_2}} & M_{2,s_2} \end{array}$$

The (inductive) definition of the *satisfaction relation* relies once again on the interpretation of terms in models, which extends the interpretation of constant-operation symbols as follows: for every $\langle S, F \rangle$ -model M and every pair of terms $t \in T_{F,s_1 \rightarrow s_2}$ and $t_1 \in T_{F,s_1}$, $M(t \ t_1) = \llbracket M_t \rrbracket_{s_1 \rightarrow s_2}^M(M_{t_1})$.

4.2 INSTITUTION-INDEPENDENT SUBSTITUTIONS

The institution-independent concept of substitution [see [Diao4](#); and also [Diao8](#)] generalizes first-order substitutions (as well as second-order and higher-order substitutions, among others) to arbitrary institutions by taking notice only of their syntactic and semantic effects: the translations of sentences and the reductions of models that they generate. A key step in arriving at this notion is the presentation of the extensions of signatures by sets of variables as particular cases of signature morphisms (along the lines of [ST84](#)). Thus, given two signature morphisms $\chi_1: \Sigma \rightarrow \Sigma_1$ and $\chi_2: \Sigma \rightarrow \Sigma_2$ (two extensions of a signature Σ) in an institution $\mathcal{J} = \langle \text{Sig}, \text{Sen}, \text{Mod}, \vDash \rangle$, a *substitution* $\psi: \chi_1 \rightarrow \chi_2$ is a pair $\langle \text{Sen}_\Sigma(\psi), \text{Mod}_\Sigma(\psi) \rangle$ consisting of

- a sentence-translation function $\text{Sen}_\Sigma(\psi): \text{Sen}(\Sigma_1) \rightarrow \text{Sen}(\Sigma_2)$ and
- a model-reduction functor $\text{Mod}_\Sigma(\psi): \text{Mod}(\Sigma_2) \rightarrow \text{Mod}(\Sigma_1)$

that preserve Σ , in the sense that $\text{Sen}(\chi_1) \S \text{Sen}_\Sigma(\psi) = \text{Sen}(\chi_2)$ and $\text{Mod}_\Sigma(\psi) \S \text{Mod}(\chi_1) = \text{Mod}(\chi_2)$, and satisfy the following condition:

$$M_2 \models_{\Sigma_2} \text{Sen}_\Sigma(\psi)(\rho_1) \quad \text{if and only if} \quad \text{Mod}_\Sigma(\psi)(M_2) \models_{\Sigma_1} \rho_1$$

for every Σ_2 -model M_2 and every Σ_1 -sentence ρ_1 .

In the present work, we take into consideration an equivalent formulation of the original definition that makes use of the category $\mathbb{R}\text{oom}$ of rooms and corridors. In addition, we extend the fact that substitutions inherit the composition of their components – thus giving rise to a category – to derive a general substitution system of Σ -substitutions for each signature Σ .

PROPOSITION 4.2.1. *Let $\mathcal{Q} \subseteq \text{Sig}$ be a class of signature morphisms of an institution $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$. For every \mathcal{J} -signature Σ we obtain a substitution system $\mathcal{S}\mathcal{J}_\Sigma^{\mathcal{Q}}: \text{Subst}_\Sigma^{\mathcal{Q}} \rightarrow \mathcal{J}(\Sigma) / \mathbb{R}\text{oom}$ defined as follows:*

- The objects of the category $\text{Subst}_\Sigma^{\mathcal{Q}}$ – i.e. the signatures of Σ -variables – are signature morphisms $\chi: \Sigma \rightarrow \Sigma(\chi)$ ³ belonging to the class \mathcal{Q} . Their corresponding corridors via the functor $\mathcal{S}\mathcal{J}_\Sigma^{\mathcal{Q}}$ are given simply by $\mathcal{J}(\chi): \mathcal{J}(\Sigma) \rightarrow \mathcal{J}(\Sigma(\chi))$.
- For every two signatures of Σ -variables $\chi_1: \Sigma \rightarrow \Sigma(\chi_1)$ and $\chi_2: \Sigma \rightarrow \Sigma(\chi_2)$, a Σ -substitution $\psi: \chi_1 \rightarrow \chi_2$ consists of a corridor $\langle \text{Sen}_\Sigma(\psi), \text{Mod}_\Sigma(\psi) \rangle$ between $\mathcal{J}(\Sigma(\chi_1))$ and $\mathcal{J}(\Sigma(\chi_2))$ such that $\mathcal{J}(\chi_1) \S \langle \text{Sen}_\Sigma(\psi), \text{Mod}_\Sigma(\psi) \rangle = \mathcal{J}(\chi_2)$.

$$\begin{array}{ccc}
 & \mathcal{J}(\Sigma) & \\
 \mathcal{J}(\chi_1) \swarrow & & \searrow \mathcal{J}(\chi_2) \\
 \mathcal{J}(\Sigma(\chi_1)) & & \mathcal{J}(\Sigma(\chi_2)) \\
 & \underbrace{\hspace{10em}}_{\langle \text{Sen}_\Sigma(\psi), \text{Mod}_\Sigma(\psi) \rangle} &
 \end{array}$$

As such, Σ -substitutions are merely arrows in the comma category $\mathcal{J}(\Sigma) / \mathbb{R}\text{oom}$, meaning that they are identified with their images under the functor $\mathcal{S}\mathcal{J}_\Sigma^{\mathcal{Q}}$. The composition of substitutions is defined accordingly. \square

EXAMPLE 4.2.2. In $\text{QF-FOL}_=$, a first-order variable over a signature $\langle S, F \rangle$ is a triple $(x, s, F_{\varepsilon \rightarrow s})$,⁴ often denoted simply by $x: s$, where x is the name of the variable and s is its sort. Thus, $\text{QF-FOL}_=$ -signatures of $\langle S, F \rangle$ -variables X are S -indexed families of sets X_s of variables of sort s such that different variables have different names. First-order substitutions $\psi: X \rightarrow Y$ can be

³ For convenience, we denote the codomain of the signature morphism χ by $\Sigma(\chi)$; this reflects the intuition that $\Sigma(\chi)$ is an extension of the signature Σ with variables defined by χ .

⁴ We denote the empty arity by ε ; hence, $F_{\varepsilon \rightarrow s}$ is the set of F -constants of sort s .

further defined as S -indexed families of maps $\psi_s : X_s \rightarrow T_{F \cup Y, s}$ that assign a term over the extended signature $\langle S, F \cup Y \rangle$ to every variable of X .

One can easily check that every first-order substitution $\psi : X \rightarrow Y$ gives rise to a general substitution between the inclusions $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ and $\langle S, F \rangle \subseteq \langle S, F \cup Y \rangle$, much the same as in the case of qF-FOL_{\neq}^1 [see e.g. Dia08]. For instance, the reduct $\text{Mod}_{\langle S, F \rangle}(\psi)(N)$ of an $\langle S, F \cup Y \rangle$ -model N is the $\langle S, F \cup X \rangle$ -expansion of $N \upharpoonright_{\langle S, F \rangle}$ given by $\text{Mod}_{\langle S, F \rangle}(\psi)(N)_{x:s} = N_{\psi(x)}$ for every variable $x : s$ of X . Note, however, that not every general substitution between $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ and $\langle S, F \rangle \subseteq \langle S, F \cup Y \rangle$ corresponds to a first-order substitution; we will discuss this aspect to a greater extent in Section 4.3.2.

Higher-order signatures of variables and substitutions can be defined likewise, by recalling that a *higher-order variable* over a signature $\langle S, F \rangle$ is a triple of the form (x, s, F_s) , where x and s correspond to the name and the type of the variable [see e.g. Tut13; and also Cod07]. As expected, this means that we allow higher-order variables to range over arbitrary functions.

4.3 QUANTIFICATION SPACES

Due to its mild assumptions, the construction outlined in Proposition 4.2.1 cannot be easily generalized to accommodate signature morphisms. More precisely, one cannot guarantee that signature morphisms $\varphi : \Sigma \rightarrow \Sigma'$ determine adequate morphisms between the substitution systems associated with Σ and Σ' : it would suffice, for example, to consider a class \mathcal{Q} of signature morphisms that consists only of extensions of Σ , thus preventing the translation of the signatures of Σ -variables along φ . To overcome this limitation, we take into account only those extensions of signatures that belong to a quantification space – a notion introduced in [Dia10] in the context of quasi-Boolean encodings⁵ and utilized in a series of papers on hybridization and many-valued institutions [see e.g. Mar+11; Dia13b; Dia13a].

Quantification spaces provide a way of translating signatures of variables along signature morphism by means of dedicated pushout constructions. For the purpose of our work, it will be convenient to consider a more categorical formulation of the original definition, based on Fact 4.3.1 below.

FACT 4.3.1. Consider a category \mathbb{K} and a subcategory \mathbb{Q} of the category \mathbb{K}^{\rightarrow} of \mathbb{K} -arrows. The domain functor $\text{dom} : \mathbb{Q} \rightarrow \mathbb{K}$ gives rise to a natural transformation $\iota_{\mathbb{Q}} : (_ / \mathbb{Q}) \Rightarrow \text{dom}^{\text{op}} \circ (_ / \mathbb{K})$ where, for every triple $\langle A_1, f, A_2 \rangle$ in

⁵ It should be noted, however, that the ideas that underlie quantification spaces can be traced back to [Tar86; ST88a] – two of the earliest works in which open formulae are treated in arbitrary institutions.

$|\mathbb{Q}|$ (i.e. arrow $f: A_1 \rightarrow A_2$ in \mathbb{K}), $\iota_{\mathbb{Q},f}: f/\mathbb{Q} \rightarrow A_1/\mathbb{K}$ is the functor that maps the morphisms $\langle g_1, g_2 \rangle: \langle A_1, f, A_2 \rangle \rightarrow \langle A'_1, f', A'_2 \rangle$ in \mathbb{Q} (corresponding to commutative squares in \mathbb{K}) to $g_1: A_1 \rightarrow A'_1$.

$$\begin{array}{ccc} A_1 & \xrightarrow{g_1} & A'_1 \\ f \downarrow & & \downarrow f' \\ A_2 & \xrightarrow{g_2} & A'_2 \end{array} \mapsto A_1 \xrightarrow{g_1} A'_1$$

DEFINITION 4.3.2 (Quantification space). A *quantification space* for an institution $\mathbb{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$ consists of a subcategory \mathbb{Q} of Sig^{\rightarrow} such that

- 1 every arrow in \mathbb{Q} corresponds to a pushout in Sig , and
- 2 the transformation $\iota_{\mathbb{Q}}: (_ / \mathbb{Q}) \Rightarrow \text{dom}^{\text{op}}; (_ / \text{Sig})$ is a natural isomorphism.

This means that for every extension of signatures $\chi: \Sigma \rightarrow \Sigma(\chi)$ in $|\mathbb{Q}|$ and every signature morphism $\varphi: \Sigma \rightarrow \Sigma'$ there exist a unique extension $\chi': \Sigma' \rightarrow \Sigma'(\chi')$ in $|\mathbb{Q}|$ and a unique signature morphism $\phi: \Sigma(\chi) \rightarrow \Sigma'(\chi')$ such that the pair $\langle \varphi, \phi \rangle$ defines a morphism in \mathbb{Q} between the arrows χ and χ' .⁶ We will henceforth denote the signature extension χ' and the signature morphism ϕ by $\chi^\varphi: \Sigma' \rightarrow \Sigma'(\chi^\varphi)$ and $\varphi^\chi: \Sigma(\chi) \rightarrow \Sigma'(\chi^\varphi)$, respectively.

$$\begin{array}{ccc} \Sigma & \xrightarrow{\varphi} & \Sigma' \\ \chi \downarrow & & \downarrow \chi^\varphi \\ \Sigma(\chi) & \xrightarrow{\varphi^\chi} & \Sigma'(\chi^\varphi) \end{array}$$

EXAMPLE 4.3.3. For both QF-FOL_- and QF-HINK , the extensions of signatures $\chi: \langle S, F \rangle \rightarrow \langle S, F \cup X \rangle$ defined by (families of) finite sets of first-order/higher-order $\langle S, F \rangle$ -variables X form a quantification space. More precisely, for every signature morphism $\varphi: \langle S, F \rangle \rightarrow \langle S', F' \rangle$,

- $\chi^\varphi: \langle S', F' \rangle \rightarrow \langle S', F' \cup X^\varphi \rangle$ is the extension of $\langle S', F' \rangle$ given by $X_s^\varphi = \{x: s' \mid x: s \in X_s \text{ for some sort } s \in S \text{ (or type } s \in \vec{S}) \text{ such that } \varphi(s) = s'\}$, and
- $\varphi^\chi: \langle S, F \cup X \rangle \rightarrow \langle S', F' \cup X^\varphi \rangle$ is the canonical extension of φ that maps each $\langle S, F \rangle$ -variable $x: s$ in X to the $\langle S', F' \rangle$ -variable $x: \varphi(s)$ in X^φ .

DEFINITION 4.3.4 (Adequacy). For any institution, a quantification space \mathbb{Q} is said to be *adequate* if every arrow $\langle \varphi, \varphi^\chi \rangle: \chi \rightarrow \chi^\varphi$ in \mathbb{Q} corresponds to a model-amalgamation square: for every Σ' -model M' and $\Sigma(\chi)$ -model

⁶ Moreover, the signature morphisms χ' and ϕ correspond to a pushout of φ and χ .

N such that $M' \upharpoonright_{\varphi} = N \upharpoonright_{\chi}$ there exists a unique model N' of $\Sigma'(\chi^{\varphi})$ – the *amalgamation* of M' and N – such that $N' \upharpoonright_{\chi^{\varphi}} = M'$ and $N' \upharpoonright_{\varphi\chi} = N$.

In semi-exact institutions⁷ – like $\text{QF-FOL}_{\underline{\quad}}$ [see [Mes89](#); [DGS93](#)] – all quantification spaces are adequate. This is not the case of QF-HNK , for which it is known that, due to the presence of higher-order types, not every pushout square of signature morphisms is a model-amalgamation square [see e.g. [Codo7](#)]. Nonetheless, the quantification space for QF-HNK outlined in [Example 4.3.3](#) is adequate: the amalgamation N' of any two given models M' of $\langle S', F' \rangle$ and N of $\langle S, F \cup X \rangle$ is the unique χ^{φ} -expansion of M' that satisfies $N'_{x: \varphi(s)} = N_{x: s}$ for each variable $x: s$ in X .

REMARK 4.3.5. Since the morphisms of any quantification space \mathbb{Q} are required to form a category (by definition, \mathbb{Q} is a subcategory of Sig^{\rightarrow}), for every signature extension $\chi: \Sigma \rightarrow \Sigma(\chi)$ in $|\mathbb{Q}|$ and every pair of composable signature morphisms $\varphi: \Sigma \rightarrow \Sigma'$ and $\varphi': \Sigma' \rightarrow \Sigma''$, we have $(\chi^{\varphi})^{\varphi'} = \chi^{\varphi \circ \varphi'}$ and $\varphi^{\chi} \circ (\varphi')^{\chi^{\varphi}} = (\varphi \circ \varphi')^{\chi}$. Moreover, $\chi^{1_{\Sigma}} = \chi$ and $1_{\Sigma(\chi)}^{\chi} = 1_{\Sigma(\chi)}$.

$$\begin{array}{ccccc}
 \Sigma & \xrightarrow{\varphi} & \Sigma' & \xrightarrow{\varphi'} & \Sigma'' \\
 \chi \downarrow & & \downarrow \chi^{\varphi} & & \downarrow (\chi^{\varphi})^{\varphi'} = \chi^{\varphi \circ \varphi'} \\
 \Sigma(\chi) & \xrightarrow{\varphi^{\chi}} & \Sigma'(\chi^{\varphi}) & \xrightarrow{(\varphi')^{\chi^{\varphi}}} & \Sigma''((\chi^{\varphi})^{\varphi'}) \\
 & \underbrace{\hspace{10em}}_{(\varphi \circ \varphi')^{\chi}} & & &
 \end{array}$$

Quantification spaces thus provide adequate support for translating abstract signature extensions along morphisms of signatures in a functorial manner.

4.3.1 REPRESENTABLE SIGNATURE EXTENSIONS

Since the institution-independent substitutions of [Proposition 4.2.1](#) correspond to a semantic concept, we cannot expect to translate them along signature morphisms in the same manner as the extensions of signatures provided by a given quantification space. The solution that we propose herein relies on an important characterization of the first-order signature extensions with variables (regarded as new constant-operation symbols): for every $\text{QF-FOL}_{\underline{\quad}}$ -signature extension $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ there is a one-to-one correspondence between the models of $\langle S, F \cup X \rangle$ and the model homomorphisms defined on the free $\langle S, F \rangle$ -algebra $T_F(X)$ over the set of variables X ; in particular, every $\langle S, F \cup X \rangle$ -model N determines the homomorphism

⁷ We recall that an institution is semi-exact – in the sense of [[Mes89](#); [DGS93](#)] – if its model functor preserves pullbacks.

$h: T_F(X) \rightarrow N \upharpoonright_{\langle S, F \rangle}$ given by $h(x) = N_x$ for every variable x in X . In this context, $T_F(X)$ is said to be a *representation* of the inclusion of signatures $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$. The following definition originates from [Dia03].

DEFINITION 4.3.6 (Representable signature morphism). In any institution, a signature morphism $\chi: \Sigma \rightarrow \Sigma(\chi)$ is *representable* if there exist a Σ -model M_χ , called the *representation* of χ , and an isomorphism of categories i_χ between $\text{Mod}(\Sigma(\chi))$ and $M_\chi / \text{Mod}(\Sigma)$ such that the following diagram commutes.

$$\begin{array}{ccc} \text{Mod}(\Sigma) & & \\ \uparrow \text{\scriptsize } _ \upharpoonright_\chi & \swarrow \text{\scriptsize } _ \perp_{M_\chi} & \\ \text{Mod}(\Sigma(\chi)) & \xrightarrow{\text{\scriptsize } i_\chi} & M_\chi / \text{Mod}(\Sigma) \end{array}$$

Representable first-order signature morphisms were studied in depth in [Sero4], from where we recall [Proposition 4.3.7](#) below [see also [Dia08](#)].

PROPOSITION 4.3.7. *A first-order signature morphism is representable if and only if it is bijective on all symbols, except constant-operation symbols.* \square

Consequently, for every QF-FOL_- -signature $\langle S, F \rangle$, all signature extensions $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ given by finite sets X of $\langle S, F \rangle$ -variables are representable.

A similar result can be obtained for QF-HNK . In that case, however, the signature extensions with constants $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ can only be guaranteed to be quasi-representable, in the sense that, for every $\langle S, F \cup X \rangle$ -model N , the canonical functor $N / \text{Mod}(S, F \cup X) \rightarrow N \upharpoonright_{\langle S, F \rangle} / \text{Mod}(S, F)$ determined by the model-reduct functor $_ \upharpoonright_{\langle S, F \rangle}$ is an isomorphism (see e.g. [Codo7] for more details). Representability further requires that the resulting higher-order signatures $\langle S, F \cup X \rangle$ have initial models, a property which holds whenever $\langle S, F \cup X \rangle$ has at least one constant-operation symbol for each type.⁸

PROPOSITION 4.3.8. *All higher-order signature extensions $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ with (finite) sets X of new constant-operation symbols are representable, provided that $F_s \cup X_s \neq \emptyset$ for each type $s \in \vec{S}$.* \square

REMARK 4.3.9. Let $\chi: \Sigma \rightarrow \Sigma(\chi)$ and $\chi': \Sigma' \rightarrow \Sigma'(\chi')$ be a pair of representable signature extensions defined by a quantification space \mathbb{Q} , and let β and

⁸ This property is commonly achieved by assuming that, for each type $s \in \vec{S}$, the set F_s contains an implicit constant-operation symbol.

β' be two functors as depicted below such that $\text{Mod}(\chi') \circ \beta = \beta' \circ \text{Mod}(\chi)$.

$$\begin{array}{ccc}
 \text{Mod}(\Sigma) & \xleftarrow{\beta} & \text{Mod}(\Sigma') \\
 \uparrow -\vdash_{\chi} & \swarrow \perp_{M_{\chi}} & \uparrow -\vdash_{\chi'} \\
 \text{Mod}(\Sigma(\chi)) & \xleftarrow{\beta'} & \text{Mod}(\Sigma'(\chi')) \\
 \searrow i_{\chi} & & \searrow i_{\chi'} \\
 M_{\chi} / \text{Mod}(\Sigma) & \xleftarrow{U} & M_{\chi'} / \text{Mod}(\Sigma')
 \end{array}$$

The composition $i_{\chi'}^{-1} \circ \beta' \circ i_{\chi}$ gives rise to a functor U between the comma categories $M_{\chi'} / \text{Mod}(\Sigma')$ and $M_{\chi} / \text{Mod}(\Sigma)$, where

- for every Σ' -model homomorphism $h': M_{\chi'} \rightarrow M'$, $U(h')$ is the Σ -model homomorphism $(i_{\chi'}^{-1} \circ \beta' \circ i_{\chi})(h')$: $M_{\chi} \rightarrow \beta(M')$, and
- for every arrow $f': h'_1 \rightarrow h'_2$ between model homomorphisms $h'_1: M_{\chi'} \rightarrow M'_1$ and $h'_2: M_{\chi'} \rightarrow M'_2$, $U(f')$ is just the β -reduct of f' , $\beta(f'): \beta(M'_1) \rightarrow \beta(M'_2)$.

When β corresponds to the model-reduct functor $\text{Mod}(\varphi)$ of a signature morphism $\varphi: \Sigma \rightarrow \Sigma'$, and when χ' and β' are χ^{φ} and $\text{Mod}(\varphi^{\chi})$, respectively, we will denote the functor $U: M_{\chi'} / \text{Mod}(\Sigma') \rightarrow M_{\chi} / \text{Mod}(\Sigma)$ by $U_{\varphi, \chi}$. Similarly, when β is the identity of $\text{Mod}(\Sigma)$ and β' is the underlying model functor of a substitution $\psi: \chi \rightarrow \chi'$, we will denote the functor U by U_{ψ} .

Model homomorphisms $h': M_{\chi'} \rightarrow M'$ can be regarded both as objects and as arrows (between $1_{M_{\chi'}}$ and h') in the comma category $M_{\chi'} / \text{Mod}(\Sigma')$. In combination with the definition of $U: M_{\chi'} / \text{Mod}(\Sigma') \rightarrow M_{\chi} / \text{Mod}(\Sigma)$, the arrow view provides us a useful factorization of $U(h')$ as $U(1_{M_{\chi'}}) \circ \beta(h')$.

$$\begin{array}{ccc}
 & M_{\chi'} & \\
 1_{M_{\chi'}} \swarrow & & \searrow h' \\
 M_{\chi'} & \xrightarrow{h'} & M'
 \end{array}
 \mapsto
 \begin{array}{ccc}
 & M_{\chi} & \\
 U(1_{M_{\chi'}}) \swarrow & & \searrow U(h') \\
 \beta(M_{\chi'}) & \xrightarrow{\beta(h')} & \beta(M')
 \end{array}$$

FACT 4.3.10. Under the notation and hypotheses of [Remark 4.3.9](#), for every Σ' -model homomorphism $h': M_{\chi'} \rightarrow M'$, $U(h') = U(1_{M_{\chi'}}) \circ \beta(h')$.

4.3.2 REPRESENTABLE SUBSTITUTIONS

Quantification spaces that have representable extensions of signatures, meaning that every extension $\chi: \Sigma \rightarrow \Sigma(\chi)$ defined by the quantification space is representable, allow us to extend the concept of representability from signature extensions (i.e. signatures of variables) to substitutions, leading to a purely model-theoretic view of the categories of substitutions.

PROPOSITION 4.3.11. *For every signature Σ in an institution equipped with a quantification space \mathbb{Q} , the representation of the extensions of signatures extends to a functor $R_\Sigma^{\mathbb{Q}}: \text{Subst}_\Sigma^{\mathbb{Q}} \rightarrow \text{Mod}(\Sigma)$, where*

- for every extension of signatures $\chi: \Sigma \rightarrow \Sigma(\chi)$ in $|\mathbb{Q}|$, $R_\Sigma^{\mathbb{Q}}(\chi) = M_\chi$, and
- for every substitution $\psi: \chi_1 \rightarrow \chi_2$, $R_\Sigma^{\mathbb{Q}}(\psi) = U_\psi(1_{M_{\chi_2}}): M_{\chi_1} \rightarrow M_{\chi_2}$.

Moreover, for any Σ -substitution ψ , $\text{Mod}_\Sigma(\psi)$ is uniquely determined by $R_\Sigma^{\mathbb{Q}}(\psi)$.

PROOF. The preservation of composition follows from a series of straightforward calculations based on the fact that, for every pair Σ -substitutions $\psi_1: \chi_1 \rightarrow \chi_2$ and $\psi_2: \chi_2 \rightarrow \chi_3$, $\text{Mod}_\Sigma(\psi_1 \circ \psi_2) = \text{Mod}_\Sigma(\psi_2) \circ \text{Mod}_\Sigma(\psi_1)$.

$$\begin{aligned}
R_\Sigma^{\mathbb{Q}}(\psi_1 \circ \psi_2) &= U_{\psi_1 \circ \psi_2}(1_{M_{\chi_3}}) && \text{by the definition of } R_\Sigma^{\mathbb{Q}} \\
&= (i_{\chi_3}^{-1} \circ \text{Mod}_\Sigma(\psi_1 \circ \psi_2) \circ i_{\chi_1})(1_{M_{\chi_3}}) && \text{by the definition of } U_{\psi_1 \circ \psi_2} \\
&= (i_{\chi_3}^{-1} \circ \text{Mod}_\Sigma(\psi_2) \circ \text{Mod}_\Sigma(\psi_1) \circ i_{\chi_1})(1_{M_{\chi_3}}) && \text{by the functoriality of } \text{Mod}_\Sigma \\
&= (i_{\chi_3}^{-1} \circ \text{Mod}_\Sigma(\psi_2) \circ i_{\chi_2} \\
&\quad \circ i_{\chi_2}^{-1} \circ \text{Mod}_\Sigma(\psi_1) \circ i_{\chi_1})(1_{M_{\chi_3}}) && \text{since } i_{\chi_2} \circ i_{\chi_2}^{-1} = 1_{\text{Mod}(\Sigma(\chi_2))} \\
&= U_{\psi_1}(U_{\psi_2}(1_{M_{\chi_3}})) && \text{according to the definitions} \\
&\quad \text{of } U_{\psi_1} \text{ and } U_{\psi_2} \\
&= U_{\psi_1}(1_{M_{\chi_2}}) \circ U_{\psi_2}(1_{M_{\chi_3}}) && \text{by Fact 4.3.10} \\
&= R_\Sigma^{\mathbb{Q}}(\psi_1) \circ R_\Sigma^{\mathbb{Q}}(\psi_2) && \text{by the definition of } R_\Sigma^{\mathbb{Q}}
\end{aligned}$$

In much the same way, based on the fact that $\text{Mod}_\Sigma(1_\chi) = 1_{\text{Mod}(\Sigma(\chi))}$, for every signature of Σ -variables $\chi: \Sigma \rightarrow \Sigma(\chi)$, we obtain $R_\Sigma^{\mathbb{Q}}(1_\chi) = 1_{M_\chi}$.

To prove the second part of the statement, notice that by the definition of U_ψ we have $\text{Mod}_\Sigma(\psi) = i_{\chi_2} \circ U_\psi \circ i_{\chi_1}^{-1}$, where χ_1 and χ_2 correspond to the domain and codomain signatures of variables of the substitution ψ . This allows us to deduce, based on Fact 4.3.10, that $\text{Mod}_\Sigma(\psi)(N_2) = i_{\chi_1}^{-1}(U_\psi(i_{\chi_2}(N_2))) = i_{\chi_1}^{-1}(U_\psi(1_{M_{\chi_2}}) \circ i_{\chi_2}(N_2)) = i_{\chi_1}^{-1}(R_\Sigma^{\mathbb{Q}}(\psi) \circ i_{\chi_2}(N_2))$ for every model N_2 of $\Sigma(\chi_2)$. Furthermore, by Remark 4.3.9, the reduction of $\Sigma(\chi_2)$ -model homomorphisms h_2 is given by $\text{Mod}_\Sigma(\psi)(h_2) = i_{\chi_1}^{-1}(i_{\chi_2}(h_2))$. Therefore, for any substitution $\psi': \chi_1 \rightarrow \chi_2$ such that $R_\Sigma^{\mathbb{Q}}(\psi') = R_\Sigma^{\mathbb{Q}}(\psi)$, $\text{Mod}_\Sigma(\psi') = \text{Mod}_\Sigma(\psi)$. \square

When the quantification space \mathbb{Q} and the signature Σ are clear from the context (as well as the institution under consideration), we may also denote the representation $R_\Sigma^{\mathbb{Q}}(\psi)$ of a substitution $\psi: \chi_1 \rightarrow \chi_2$ by $h_\psi: M_{\chi_1} \rightarrow M_{\chi_2}$.

Note that, in general, the functor $R_\Sigma^{\mathbb{Q}}: \text{Subst}_\Sigma^{\mathbb{Q}} \rightarrow \text{Mod}(\Sigma)$ of Proposition 4.3.11 need be neither full nor faithful. For example, in the case of

QF-FOL_Σ , for every general substitution ψ we can define another substitution ψ' (with the same domain and codomain as ψ) such that, for every atomic sentence $l = r$ that is not ground, $\text{Sen}_\Sigma(\psi')(l = r)$ corresponds to $\text{Sen}_\Sigma(\psi)(r = l)$. In consequence, we can obtain distinct institution-independent substitutions having the same underlying model functor – and thus, the same representation. This is contrary to our intuition concerning first-order substitutions, where, given a signature $\langle S, F \rangle$, every substitution $\psi: X_1 \rightarrow X_2$, i.e. every S -indexed family of maps $\psi_s: X_{1,s} \rightarrow T_{F \cup X_2, s}$, is determined uniquely by its representation $R_{\langle S, F \rangle}^{\text{Q}}(\psi): T_F(X_1) \rightarrow T_F(X_2)$. As we will see later, the full and faithful representation of substitutions as model homomorphisms is essential for translating substitutions along signature morphisms.

DEFINITION 4.3.12 (Representable substitution). Let Σ be a signature in an institution equipped with a quantification space Q . For every subcategory $\text{Subst}_\Sigma \subseteq \text{Subst}_\Sigma^{\text{Q}}$, a substitution $\psi: \chi_1 \rightarrow \chi_2$ in Subst_Σ is said to be *Q-representable* if it is uniquely determined by its image under R_Σ^{Q} . In addition, Subst_Σ forms a category of *Q-representable Σ -substitutions* if the restriction of the functor $R_\Sigma^{\text{Q}}: \text{Subst}_\Sigma^{\text{Q}} \rightarrow \text{Mod}(\Sigma)$ to Subst_Σ is both full and faithful.

EXAMPLE 4.3.13. Let Q be the quantification space for QF-FOL_Σ presented in [Example 4.3.3](#). For every first-order signature $\langle S, F \rangle$, the subcategory $\text{Subst}_{\langle S, F \rangle} \subseteq \text{Subst}_{\langle S, F \rangle}^{\text{Q}}$ whose arrows correspond to the corridors induced by first-order substitution forms a category of *Q-representable substitutions*. A similar property can be formulated for higher-order substitutions.

For the remaining part of this chapter we will assume that \mathcal{J} is an arbitrary but fixed institution $\langle \text{Sig}, \text{Sen}, \text{Mod}, \models \rangle$ equipped with

- *an adequate quantification space $\text{Q} \subseteq \text{Sig}^\rightarrow$ of representable signature extensions,*
- *a broad subcategory $\text{Subst}_\Sigma \subseteq \text{Subst}_\Sigma^{\text{Q}}$ (i.e. with the same objects as $\text{Subst}_\Sigma^{\text{Q}}$), for every signature $\Sigma \in |\text{Sig}|$, of Q-representable Σ -substitutions.*

LEMMA 4.3.14. *Under the above assumptions, for every morphism $\varphi: \Sigma \rightarrow \Sigma'$ in Sig and every signature extension $\chi: \Sigma \rightarrow \Sigma(\chi)$ in $|\text{Q}|$, the homomorphism $U_{\varphi, \chi}(1_{M_{\chi^\varphi}}): M_\chi \rightarrow M_{\chi^\varphi} \upharpoonright_\varphi$ is a universal arrow from M_χ to $\text{Mod}(\varphi)$.*

PROOF. Suppose that $h: M_\chi \rightarrow M' \upharpoonright_\varphi$ is a Σ -model homomorphism from M_χ to the reduct of a Σ' -model M' . Since χ is representable, we know that $i_\chi^{-1}(h) \upharpoonright_\chi = M' \upharpoonright_\varphi$. This allows us to deduce, based on the adequacy of Q , that there exists a unique $\Sigma'(\chi^\varphi)$ -model N' satisfying $N' \upharpoonright_{\chi^\varphi} = M'$ and $N' \upharpoonright_{\varphi\chi} = i_\chi^{-1}(h)$. We thus obtain the homomorphism $i_{\chi^\varphi}(N'): M_{\chi^\varphi} \rightarrow M'$.

To show that $U_{\varphi,\chi}(1_{M_{\chi^\varphi}}) \circ i_{\chi^\varphi}(N') \upharpoonright_\varphi = h$, and thus that $i_{\chi^\varphi}(N') \upharpoonright_\varphi$ is an arrow in $M_\chi / \text{Mod}(\Sigma)$ between $U_{\varphi,\chi}(1_{M_{\chi^\varphi}})$ and h , it suffices to consider $i_{\chi^\varphi}(N')$ as an arrow in $M_{\chi^\varphi} / \text{Mod}(\Sigma')$ between $1_{M_{\chi^\varphi}}$ and $i_{\chi^\varphi}(N')$. In this way, by taking the image of $i_{\chi^\varphi}(N')$ under $U_{\varphi,\chi}$, we obtain the arrow $i_{\chi^\varphi}(N') \upharpoonright_\varphi$ between $U_{\varphi,\chi}(1_{M_{\chi^\varphi}})$ and $U_{\varphi,\chi}(i_{\chi^\varphi}(N')) = i_\chi(N' \upharpoonright_{\varphi^\chi}) = h$.

$$\begin{array}{ccc}
 M_\chi & \xrightarrow{U_{\varphi,\chi}(1_{M_{\chi^\varphi}})} & M_{\chi^\varphi} \upharpoonright_\varphi & & M_{\chi^\varphi} \\
 & \searrow h & \downarrow i_{\chi^\varphi}(N') \upharpoonright_\varphi & & \downarrow i_{\chi^\varphi}(N') \\
 & & M' \upharpoonright_\varphi & & M'
 \end{array}$$

For the ‘uniqueness’ part of the universality of $U_{\varphi,\chi}(1_{M_{\chi^\varphi}})$, let f' be an arbitrary Σ' -homomorphism $M_{\chi^\varphi} \rightarrow M'$ such that $U_{\varphi,\chi}(1_{M_{\chi^\varphi}}) \circ f' \upharpoonright_\varphi = h$. Then $i_{\chi^\varphi}^{-1}(f')$ is a $\Sigma'(\chi^\varphi)$ -model and, in addition, it satisfies $i_{\chi^\varphi}^{-1}(f') \upharpoonright_{\chi^\varphi} = M'$ and $i_{\chi^\varphi}^{-1}(f') \upharpoonright_{\varphi^\chi} = i_\chi^{-1}(h)$; the latter equality follows from the observation that $i_\chi(i_{\chi^\varphi}^{-1}(f') \upharpoonright_{\varphi^\chi}) = U_{\varphi,\chi}(f') = U_{\varphi,\chi}(1_{M_{\chi^\varphi}}) \circ f' \upharpoonright_\varphi = h$. Therefore, $i_{\chi^\varphi}^{-1}(f')$ is the amalgamation of M' and $i_\chi^{-1}(h)$; this further implies that $i_{\chi^\varphi}^{-1}(f') = N'$, because \mathbb{Q} is adequate, thus allowing us to conclude that $f' = i_{\chi^\varphi}(N')$. \square

The lemma above enables us to make use of a well-known construction of adjoint functors from universal arrows [see e.g. [AHS09](#)] to derive translations between categories of substitutions. We thus obtain the following result.

PROPOSITION 4.3.15. *Every morphism of signatures $\varphi: \Sigma \rightarrow \Sigma'$ gives rise to a functor $\Psi_\varphi: \text{Subst}_\Sigma \rightarrow \text{Subst}_{\Sigma'}$ that maps*

$$\begin{array}{ccc}
 \mathcal{J}(\Sigma) & \xrightarrow{\varphi} & \mathcal{J}(\Sigma') \\
 \mathcal{J}(\chi_1) \swarrow & & \swarrow \mathcal{J}(\chi_1^\varphi) \\
 \mathcal{J}(\Sigma(\chi_1)) & \xrightarrow{\mathcal{J}(\varphi^{\chi_1})} & \mathcal{J}(\Sigma'(\chi_1^\varphi)) \\
 \psi \searrow & & \searrow (R_{\Sigma'}^{\mathbb{Q}})^{-1}(h_\psi^\varphi) \\
 & \mathcal{J}(\Sigma(\chi_2)) \xrightarrow{\mathcal{J}(\varphi^{\chi_2})} \mathcal{J}(\Sigma'(\chi_2^\varphi)) &
 \end{array}$$

- every signature extension $\chi: \Sigma \rightarrow \Sigma(\chi)$ to $\chi^\varphi: \Sigma' \rightarrow \Sigma'(\chi^\varphi)$, and
- every Σ -substitution $\psi: \chi_1 \rightarrow \chi_2$ to $\psi^\varphi = (R_{\Sigma'}^{\mathbb{Q}})^{-1}(h_\psi^\varphi)$, where h_ψ^φ is the unique Σ' -homomorphism $M_{\chi_1^\varphi} \rightarrow M_{\chi_2^\varphi}$ for which the diagram below commutes.

$$\begin{array}{ccccc}
 M_{\chi_1} & \xrightarrow{U_{\varphi,\chi_1}(1_{M_{\chi_1^\varphi}})} & M_{\chi_1^\varphi} \upharpoonright_\varphi & & M_{\chi_1^\varphi} \\
 & \searrow h_\psi & \downarrow h_\psi^\varphi \upharpoonright_\varphi & & \downarrow h_\psi^\varphi \\
 & & M_{\chi_2^\varphi} \upharpoonright_\varphi & & M_{\chi_2^\varphi} \\
 & & \uparrow U_{\varphi,\chi_2}(1_{M_{\chi_2^\varphi}}) & &
 \end{array}$$

Moreover, Ψ itself is functorial, in the sense that $\Psi_{\varphi \circ \varphi'} = \Psi_\varphi \circ \Psi_{\varphi'}$ for every composable signature morphisms $\varphi: \Sigma \rightarrow \Sigma'$ and $\varphi': \Sigma' \rightarrow \Sigma''$, and $\Psi_{1_\Sigma} = 1_{\text{Subst}_\Sigma}$.

PROOF. The first part of the statement follows by [Lemma 4.3.14](#) as a direct consequence of the universal property of the homomorphism $U_{\varphi, \chi_1}(1_{M_{\chi_1}^\varphi})$ – note that, since the functor R_Σ^Q is assumed to be both full and faithful, for every signature Σ , it suffices to reason about the representations of substitutions. With respect to the second part of the statement, notice first that, by the definition of quantification spaces, the translation of signature extensions along signature morphisms is functorial (see [Remark 4.3.5](#)). In addition, by [Remark 4.3.9](#), for every signature extension $\chi: \Sigma \rightarrow \Sigma(\chi)$, $U_{\varphi \circ \varphi', \chi} = U_{\varphi', \chi^\varphi} \circ U_{\varphi, \chi}$. This allows us to deduce, according to [Fact 4.3.10](#), that $U_{\varphi \circ \varphi', \chi}(1_{M_{\chi^\varphi}}) = U_{\varphi, \chi}(1_{M_{\chi^\varphi}}) \circ U_{\varphi', \chi^\varphi}(1_{M_{\chi^\varphi \circ \varphi'}})$. Hence, by the general properties of composing universal arrows, we can further conclude that the translation of substitutions along signature morphisms is also functorial. \square

4.3.3 DERIVING GENERALIZED SUBSTITUTION SYSTEMS

For any signature morphism $\varphi: \Sigma \rightarrow \Sigma'$, the functor $\Psi_\varphi: \text{Subst}_\Sigma \rightarrow \text{Subst}_{\Sigma'}$ discussed in [Proposition 4.3.15](#) can be extended in a straightforward manner to a morphism $\langle \Psi_\varphi, \kappa_\varphi, \tau_\varphi \rangle$ between the substitution systems $\mathcal{S}\mathcal{J}_\Sigma$ and $\mathcal{S}\mathcal{J}_{\Sigma'}$ obtained by restricting the functors $\mathcal{S}\mathcal{J}_\Sigma^Q$ and $\mathcal{S}\mathcal{J}_{\Sigma'}^Q$ of [Proposition 4.2.1](#) to the subcategories Subst_Σ and $\text{Subst}_{\Sigma'}$ of Σ - and Σ' -substitutions.

$$\begin{array}{ccc} \text{Subst}_\Sigma & \xrightarrow{\mathcal{S}\mathcal{J}_\Sigma} & \mathcal{J}(\Sigma) / \mathbb{R}\text{oom} \\ \Psi_\varphi \downarrow & \Downarrow \tau_\varphi & \uparrow \mathcal{J}(\varphi) / \mathbb{R}\text{oom} \\ \text{Subst}_{\Sigma'} & \xrightarrow{\mathcal{S}\mathcal{J}_{\Sigma'}} & \mathcal{J}(\Sigma') / \mathbb{R}\text{oom} \end{array}$$

To be more specific, κ_φ is the corridor $\langle \text{Sen}(\varphi), \text{Mod}(\varphi) \rangle$ obtained by taking the image $\mathcal{J}(\varphi)$ of φ under the institution \mathcal{J} , regarded as a functor into $\mathbb{R}\text{oom}$. Furthermore, for every signature extension $\chi: \Sigma \rightarrow \Sigma(\chi)$, the corridor $\tau_{\varphi, \chi}: \mathcal{J}(\Sigma(\chi)) \rightarrow \mathcal{J}(\Sigma'(\chi^\varphi))$ is simply $\mathcal{J}(\varphi^\chi)$. It should be noted, however, that the naturality of τ_φ holds in general only up to semantic equivalence (see [Proposition 4.3.16](#) below): this means that we can only guarantee that $\text{Mod}(\varphi^\chi)$ is natural in χ , and thus that, for every substitution $\psi: \chi_1 \rightarrow \chi_2$ and sentence ρ over $\Sigma(\chi_1)$, $\varphi^{\chi_2}(\psi(\rho)) \vDash \psi^\varphi(\varphi^{\chi_1}(\rho))$.⁹ In concrete cases like QF-FOL_- , the equality $\varphi^{\chi_2}(\psi(\rho)) = \psi^\varphi(\varphi^{\chi_1}(\rho))$ is usually due to the careful choice of the categories of substitutions; other choices, which may involve,

⁹ We recall that two sentences ρ_1 and ρ_2 are *semantically equivalent*, denoted $\rho_1 \vDash \rho_2$, if and only if they are satisfied by the same class of models.

for example, swapping the left- and the right-hand side of non-ground equational atoms, do not necessarily give rise to natural transformations τ_φ . For this reason, in what follows, we will implicitly assume that the categories Subst_Σ of Σ -substitutions are *compatible* with respect to signature morphisms, meaning that $\psi \circ \mathcal{J}(\varphi^{\chi_2}) = \mathcal{J}(\varphi^{\chi_1}) \circ \psi^\varphi$ for every substitution $\psi: \chi_1 \rightarrow \chi_2$.

PROPOSITION 4.3.16. *For every signature morphism $\varphi: \Sigma \rightarrow \Sigma'$ and every Σ -substitution $\psi: \chi_1 \rightarrow \chi_2$, $\text{Mod}(\varphi^{\chi_2}) \circ \text{Mod}_\Sigma(\psi) = \text{Mod}_{\Sigma'}(\psi^\varphi) \circ \text{Mod}(\varphi^{\chi_1})$.*

PROOF. The conclusion follows easily by ‘chasing’ the diagram bellow, provided that the outer square is commutative, that is $U_{\varphi, \chi_2} \circ U_\psi = U_{\psi^\varphi} \circ U_{\varphi, \chi_1}$.

$$\begin{array}{ccccc}
 & & & & U_{\varphi, \chi_1} \\
 & \swarrow & & \searrow & \\
 M_{\chi_1} / \text{Mod}(\Sigma) & \xleftarrow{i_{\chi_1}} & \text{Mod}(\Sigma(\chi_1)) & \xleftarrow{-\uparrow_{\varphi^{\chi_1}}} & \text{Mod}(\Sigma'(\chi_1^\varphi)) & \xrightarrow{i_{\chi_1}^\varphi} & M_{\chi_1} / \text{Mod}(\Sigma') \\
 \uparrow U_\psi & & \uparrow -\uparrow_\psi & & \uparrow -\uparrow_{\psi^\varphi} & & \uparrow U_{\psi^\varphi} \\
 M_{\chi_1} / \text{Mod}(\Sigma) & \xleftarrow{i_{\chi_2}} & \text{Mod}(\Sigma(\chi_2)) & \xleftarrow{-\uparrow_{\varphi^{\chi_2}}} & \text{Mod}(\Sigma'(\chi_2^\varphi)) & \xrightarrow{i_{\chi_2}^\varphi} & M_{\chi_2} / \text{Mod}(\Sigma') \\
 & \swarrow & & \searrow & & & \\
 & & & & & & U_{\varphi, \chi_2}
 \end{array}$$

This equality can also be established without difficulty by noticing that, for every Σ' -homomorphism $h': M_{\chi_2}^\varphi \rightarrow M'$,

$$\begin{aligned}
 & (U_{\varphi, \chi_2} \circ U_\psi)(h') \\
 &= h_\psi \circ U_{\varphi, \chi_2}(1_{M_{\chi_2}^\varphi}) \circ h' \uparrow_\varphi && \text{by Fact 4.3.10 and Proposition 4.3.11} \\
 &= U_{\varphi, \chi_1}(1_{M_{\chi_2}^\varphi}) \circ h_\psi^\varphi \uparrow_\varphi \circ h' \uparrow_\varphi && \text{by Proposition 4.3.15} \\
 &= (U_{\psi^\varphi} \circ U_{\varphi, \chi_1})(h'). && \text{by Fact 4.3.10 and Proposition 4.3.11}
 \end{aligned}$$

Analogously, for every arrow f' in the comma category $M_{\chi_2}^\varphi / \text{Mod}(\Sigma')$, one can check directly that $(U_{\varphi, \chi_2} \circ U_\psi)(f') = f' \uparrow_\varphi = (U_{\psi^\varphi} \circ U_{\varphi, \chi_1})(f')$. \square

We can now conclude the construction of a generalized substitution system $\mathcal{S}\mathcal{J}: \text{Sig} \rightarrow \text{SubstSys}$ from an arbitrary institution $\mathcal{J}: \text{Sig} \rightarrow \text{Room}$ that satisfies the hypotheses laid out in Section 4.3.2 by noticing that, according to Proposition 4.3.15, to the fact that \mathcal{J} is a functor, and to Remark 4.3.5, all components of the morphism of substitution systems $\langle \Psi_\varphi, \kappa_\varphi, \tau_\varphi \rangle$ presented above are functorial in φ . Moreover, since the quantification space of \mathcal{J} is assumed to be adequate, the generalized substitution system $\mathcal{S}\mathcal{J}$ has model amalgamation. This leads us to the first main result of the present chapter.

THEOREM 4.3.17. *For any institution $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$ equipped with an adequate quantification space \mathcal{Q} of representable signature extensions and with compatible categories Subst_Σ of \mathcal{Q} -representable Σ -substitutions, $\mathcal{S}\mathcal{J}: \text{Sig} \rightarrow \text{SubstSys}$ is a generalized substitution system that has model amalgamation. \square*

EXAMPLE 4.3.18. Both institutions $\mathcal{Q}\text{F-FOL}_-$ and $\mathcal{Q}\text{F-HNK}$, in combination with signature extensions with constants and with the first-order and higher-order substitutions outlined in [Example 4.2.2](#), give rise to generalized substitution systems, which we will denote by $\mathcal{Q}\text{F-FOL}_=$ and $\mathcal{Q}\text{F-HNK}$ respectively.

4.4 LOGIC PROGRAMMING OVER AN ARBITRARY INSTITUTION

The view we take here is that the logic programming paradigm can be developed over an arbitrary institution $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$ by considering logic-programming frameworks and languages defined over the generalized substitution system $\mathcal{S}\mathcal{J}: \text{Sig} \rightarrow \text{SubstSys}$ introduced in [Section 4.3.3](#). To this end, we assume that $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}\text{oom}$ is an institution that satisfies the hypotheses of [Theorem 4.3.17](#), and we let \mathcal{L} be a logic-programming language $\langle \text{LP}, \text{Sign}, \text{PMod}, \text{Ax} \rangle$ defined over a framework $\langle \mathcal{S}\mathcal{J}, \mathcal{C}, \mathcal{Q}, \Vdash \rangle$. Since the underlying generalized substitution system of \mathcal{L} is derived from \mathcal{J} , it is often more convenient – especially when discussing concrete examples – to regard \mathcal{C} and \mathcal{Q} as subfunctors of the sentence functor $\text{Sen}: \text{Sig} \rightarrow \text{Set}$ of \mathcal{J} .

4.4.1 PARAMODULATION

Paramodulation originated with the work of Robinson and Wos [[RW83](#)] as a refinement of resolution that is suitable for dealing with clauses and queries defined over first-order logic with equality. The definition of its corresponding goal-directed rules relies on the following notions of context and substitution of a term in a given context [see e.g. [ST11](#); and also [BN99](#)].

For any sort s of a first-order signature $\langle S, F \rangle$ – and similarly, for any type s of a higher-order signature $\langle S, F \rangle$ – a *context for s* is a term c over an extended signature of the form $\langle S, F \cup \{\square: s\} \rangle$ that contains precisely one occurrence of the new variable \square . The *substitution of a term $t \in T_{F,s}$ in c* , denoted $c[t]$, is defined simply as the translation of c along the substitution $(\square \mapsto t): \{\square: s\} \rightarrow \emptyset$ that maps \square to the term t .

CLAUSES AND QUERIES. For every signature $\langle S, F \rangle$ – and, in particular, for signature extensions with constants $\langle S, F \cup X \rangle$ – the local (quantifier-free) *clauses* are defined, as in the relational case, as implications of the form

$\bigwedge H \Rightarrow (l = r)$, where H is a finite set of equational atoms and $l = r$ is an equational atom. In a similar manner, the local *queries* over $\langle S, F \rangle$ are just finite conjunctions $\bigwedge Q$ of equational atoms.

GOAL-DIRECTED RULES. (Local) *paramodulation* is given by rules of the form

$$\bigwedge (\{c[l] = t\} \cup Q), (\bigwedge H \Rightarrow (l = r)) \Vdash_{\langle S, F \cup X \rangle} \bigwedge (\{c[r] = t\} \cup Q \cup H)$$

or

$$\bigwedge (\{t = c[l]\} \cup Q), (\bigwedge H \Rightarrow (l = r)) \Vdash_{\langle S, F \cup X \rangle} \bigwedge (\{t = c[r]\} \cup Q \cup H)$$

where $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ is a signature extension with constants, Q and H are finite sets of atoms over $\langle S, F \cup X \rangle$, l and r are $(F \cup X)$ -terms of some sort s , c is a context for s , and t is an $(F \cup X)$ -term with the same sort as c .

It is easy to verify that the above constructions satisfy all the necessary properties of [Definition 3.3.2](#), thus giving rise to two logic-programming frameworks: $\mathcal{FOL}_=$ – over (many-sorted) first-order equational logic – and \mathcal{HCK} – over higher-order equational logic. Moreover, the abstract concept of resolution captures in this setting an extended version of what is known as paramodulation: a query $\exists X_2 \cdot \bigwedge Q_2$ (over some signature $\langle S, F \rangle$) is derived by paramodulation in one step from another query $\exists X_1 \cdot \bigwedge Q_1$ and a clause $\forall Y_1 \cdot \bigwedge H_1 \Rightarrow (l_1 = r_1)$ using the computed substitution $\theta_1: X_1 \rightarrow X_2$ if and only if there exists a substitution $\psi_1: Y_1 \rightarrow X_2$ such that

$$\bigwedge \theta_1(Q_1), (\bigwedge \psi_1(H_1) \Rightarrow (\psi_1^{\text{tm}}(l_1) = \psi_1^{\text{tm}}(r_1))) \Vdash_{\langle S, F \cup X_2 \rangle} \bigwedge Q_2.$$

The extension is in this case twofold: first, the derivation is not limited to most general unifiers (similarly to the relational variant of logic programming); second, the term $\psi_1^{\text{tm}}(l_1)$ need not be equal with the θ_1 -translation of a subterm in Q_1 , but with a subterm in the θ_1 -translation of Q_1 .

COMPUTED ANSWERS. All that is needed now in order to compute answers to queries defined over $\mathcal{FOL}_=$ or \mathcal{HCK} is an analogue of [Fact 3.3.21](#) providing an adequate characterization of trivial queries. To this end, notice that a first-order signature extension with constants $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ is conservative if and only if there exists at least one F -term for each sort $s \in S$ such that $T_{F \cup X, s}$ is not empty. A similar property holds in general for higher-order logic because we assumed that all signatures of interest have at least one constant-operation symbol for each type. Hence, all higher-order signature extensions with constants that we consider here are conservative.

FACT 4.4.1. Let $\langle S, F \rangle \subseteq \langle S, F \cup Y \rangle$ be a $\mathcal{FOL}_=$ -signature extension such that

$T_{F,s} = \emptyset$ implies $T_{F \cup Y,s} = \emptyset$ for every $s \in S$. Then an $\langle S, F \rangle$ -query $\exists Y \cdot \top$ is trivial if and only if \top is a conjunction of equalities of the form $t = t$.

To illustrate the use of paramodulation, let Γ be the set that consists of the two clauses defined by the module ADD together with the identity clause $\forall \theta \cdot true \Rightarrow (\theta = \theta)$. Then the sum of $s \theta$ and $s \theta$ can be computed according to the following chain of derivations:

$$\begin{aligned}
& \exists\{X_1: \text{Nat}\} \cdot s \theta + s \theta = X_1 \\
& \longrightarrow_{\Gamma, X_1 \mapsto X_2} \exists\{X_2: \text{Nat}\} \cdot s(\theta + s \theta) = X_2 && \text{using the second clause of } \Gamma \text{ and} \\
& && \text{the substitution } M \mapsto \theta, N \mapsto s \theta \\
& \longrightarrow_{\Gamma, X_2 \mapsto X_3} \exists\{X_3: \text{Nat}\} \cdot s s \theta = X_3 && \text{using the first clause of } \Gamma \text{ and the} \\
& && \text{substitution } M \mapsto s \theta \\
& \longrightarrow_{\Gamma, X_3 \mapsto s s \theta} \exists \emptyset \cdot s s \theta = s s \theta. && \text{using the third (and last) clause}
\end{aligned}$$

4.4.2 HERBRAND'S THEOREM REVISITED

Under the additional assumption that, for every signature Σ , the identity 1_Σ is a signature of variables – the ‘empty’ signature of Σ -variables – the general institution-independent versions of Herbrand’s theorem presented in [Diao4; Diao8] can be obtained as more concrete instances of [Theorem 3.3.13](#). In particular, the equivalence [1](#) \Leftrightarrow [2](#) of [Theorem 4.4.5](#) captures the denotational aspect of Herbrand’s theorem – how the problem of checking whether a logic program entails a given query can be reduced from all models of the program to those that are initial; on the other hand, the equivalence [2](#) \Leftrightarrow [3](#) emphasizes the operational aspect of the theorem – the correspondence between those expansions of the program’s initial model that satisfy the underlying (quantifier-free) sentence of the query and the possible solutions to the query, about which we know that can be computed when properties such as query-completeness are satisfied (see [Theorem 3.3.35](#)).

To start with, let us recall that, in any category, an object A is said to be *projective* with respect to an arrow $e: B \rightarrow C$ provided that every arrow $f: A \rightarrow C$ can be factored through e as $f = h \circ e$, for some arrow $h: A \rightarrow B$. For instance, as a consequence of the axiom of choice, for every QF-FOL $_{=}$ -signature extension $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$, the free algebra $T_F(X)$ – the representation of the inclusion $\langle S, F \rangle \subseteq \langle S, F \cup X \rangle$ – is projective with respect to all epimorphisms, and in particular with respect to all quotient homomorphisms $0_{\langle S, F \rangle} \rightarrow 0_{\langle S, F \rangle, \Gamma}$ that correspond to sets of $\langle S, F \rangle$ -clauses Γ .

The concept of *basic sentence* [see [Diao3](#); and also [Tar86](#), where it was studied under the name of *ground positive elementary sentence*] captures categorically the satisfaction of the conjunctions of atomic sentences that are usually involved in defining logic-programming queries.

DEFINITION 4.4.2 (Basic sentence). For any signature Σ , a sentence ρ is said to be *basic* if there exists a model M_ρ such that, for every Σ -model M , $M \vDash_\Sigma \rho$ if and only if there exists a model homomorphism $M_\rho \rightarrow M$.

EXAMPLE 4.4.3. In the institution $\text{QF-FOL}_{\underline{\quad}}$, every (finite) conjunction $\bigwedge E$ of first-order equational atoms forms a basic sentence [see e.g. [Dia08](#)]. This property does not hold in general for QF-HNK , for which one can define higher-order equational atoms of the form $\sigma_1 f = \sigma_2 f$, with $f: s \rightarrow s$ and $\sigma_1, \sigma_2: (s \rightarrow s) \rightarrow s'$, that are not basic [see [Codo7](#); [Dia08](#)].

We also recall from [[Dia03](#)] the following immediate result.

FACT 4.4.4. For any signature Σ , the satisfaction of basic sentences is preserved by all Σ -model homomorphisms: for every basic sentence ρ and model homomorphism $h: M_1 \rightarrow M_2$ such that $M_1 \vDash_\Sigma \rho$, $M_2 \vDash_\Sigma \rho$.

The above preliminaries enable us to recast the institution-independent versions of Herbrand's theorem in the context of (institution-based) abstract logic-programming languages.

THEOREM 4.4.5. Consider a logic program $\langle\langle \Sigma, \Gamma \rangle\rangle$ and a Σ -query $\exists \chi \cdot \rho$ such that

- both the signature Σ and the program $\langle\langle \Sigma, \Gamma \rangle\rangle$ have initial models 0_Σ and $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$,
- M_χ is projective with respect to the unique homomorphism $!_\Gamma: 0_\Sigma \rightarrow 0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$, and
- the $\Sigma(\chi)$ -sentence ρ is basic.

Then the following statements are equivalent:

- 1 $\langle\langle \Sigma, \Gamma \rangle\rangle \vDash_\Sigma^{\text{lp}} \exists \chi \cdot \rho$.
- 2 $0_{\langle\langle \Sigma, \Gamma \rangle\rangle} \vDash_\Sigma^{\text{qs}} \exists \chi \cdot \rho$.
- 3 $\exists \chi \cdot \rho$ admits a $\langle\langle \Sigma, \Gamma \rangle\rangle$ -solution.

PROOF. According to [Theorem 3.3.13](#), it suffices to show that $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ is χ -reachable and that the satisfaction of ρ is preserved by χ -homomorphisms. The latter property is guaranteed by [Fact 4.4.4](#), because ρ is basic. Therefore, we will focus solely on proving that $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ is χ -reachable.

Let $N_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ be a χ -expansion of $0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$. Since χ is a representable extension of signatures (by hypothesis), we know that $i_\chi(N_{\langle\langle \Sigma, \Gamma \rangle\rangle}): M_\chi \rightarrow 0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$ is an object of the comma category $M_\chi / \text{Mod}(\Sigma)$; and because its representation, M_χ , is projective with respect to $!_\Gamma: 0_\Sigma \rightarrow 0_{\langle\langle \Sigma, \Gamma \rangle\rangle}$, it follows that there exists a Σ -model homomorphism $h: M_\chi \rightarrow 0_\Sigma$ such that $h \circ !_\Gamma = i_\chi(N_{\langle\langle \Sigma, \Gamma \rangle\rangle})$.

$$\begin{array}{ccc}
 M_\chi & \xrightarrow{h} & 0_\Sigma \\
 & \searrow & \swarrow \\
 & i_\chi(N_{\langle\langle \Sigma, \Gamma \rangle\rangle}) & !_\Gamma \\
 & & 0_{\langle\langle \Sigma, \Gamma \rangle\rangle}
 \end{array}$$

Moreover, since the identity 1_Σ is a signature of Σ -variables – which, by hypothesis, is also representable – we deduce that 0_Σ is (isomorphic to) the representation M_{1_Σ} of 1_Σ . By the representability of Σ -substitutions, we further obtain the substitution $(R_{\Sigma'}^Q)^{-1}(h): \chi \rightarrow 1_\Sigma$, which we will henceforth denote by ψ . All we need to prove now is that the canonical map $-\uparrow_\Sigma: N_{\langle\langle\Sigma,\Gamma\rangle\rangle} / \text{Mod}_\Sigma(\psi) \rightarrow 0_{\langle\langle\Sigma,\Gamma\rangle\rangle} / \text{Mod}(\Sigma)$ is surjective on objects.

To this end, notice that every Σ -model homomorphism $g: 0_{\langle\langle\Sigma,\Gamma\rangle\rangle} \rightarrow M$ can be viewed as an arrow in $M_\chi / \text{Mod}(\Sigma)$ between $i_\chi(N_{\langle\langle\Sigma,\Gamma\rangle\rangle})$ and $i_\chi(N_{\langle\langle\Sigma,\Gamma\rangle\rangle}) \wp g$, from which we deduce that $i_\chi^{-1}(g)$ is a $\Sigma(\chi)$ -model homomorphism between $N_{\langle\langle\Sigma,\Gamma\rangle\rangle}$ and $N = i_\chi^{-1}(i_\chi(N_{\langle\langle\Sigma,\Gamma\rangle\rangle}) \wp g)$. In addition, by [Proposition 4.3.11](#) and the commutativity of the diagram below, we obtain $M \uparrow_\psi = i_\chi^{-1}(h \wp !_M) = i_\chi^{-1}(i_\chi(N_{\langle\langle\Sigma,\Gamma\rangle\rangle}) \wp g) = N$, thus confirming that $i_\chi^{-1}(g): N_{\langle\langle\Sigma,\Gamma\rangle\rangle} \rightarrow M \uparrow_\psi$ is an object of $N_{\langle\langle\Sigma,\Gamma\rangle\rangle} / \text{Mod}_\Sigma(\psi)$. The conclusion of the theorem follows by observing that $N \uparrow_\Sigma = |i_\chi(N_{\langle\langle\Sigma,\Gamma\rangle\rangle}) \wp g|_{M_\chi} = M$ and $i_\chi^{-1}(g) \uparrow_\Sigma = |g|_{M_\chi} = g$.

$$\begin{array}{ccc}
 M_\chi & \xrightarrow{h} & 0_\Sigma \\
 \searrow^{i_\chi(N_{\langle\langle\Sigma,\Gamma\rangle\rangle})} & & \swarrow^{!_\Gamma} \\
 & 0_{\langle\langle\Sigma,\Gamma\rangle\rangle} & \xrightarrow{g} M \\
 & & \swarrow^{!_M}
 \end{array}$$

□

A LOGIC-PROGRAMMING SEMANTICS OF SERVICES

Building on recent theoretical advancements in the algebraic structures that capture the way services are orchestrated and in the processes that formalize the discovery and binding of services to given client applications by means of logical representations of required and provided services, in the following we develop formal foundations for notions and mechanisms needed to support service-oriented computing. In particular, we show how the denotational and the operational semantics specific to conventional logic programming can be generalized using the theory presented in [Chapter 3](#) to address both static and dynamic aspects of services, thus leading to a service-oriented variant of the paradigm. Our results rely upon a strong analogy between the discovery of a service that can be bound to an application and the search for a clause that can be used for computing an answer to a query; they explore the manner in which requests for external services can be described as service queries, and explain how the computation of their answers can be performed through service-oriented derivatives of unification and resolution, which characterize the binding of services and the reconfiguration of applications.

5.1 ORCHESTRATION SCHEMES

The first step in the development of the particular variant of logic programming that we consider in this chapter consists in determining appropriate categorical abstractions of the structures that support service-oriented computing. These will ultimately allow us to describe the process of service discovery and binding in a way that is independent of any particular formalism (like various forms of automata, transition systems, or process algebras).

Our approach is grounded on two observations: first, that orchestrations can be organized as a category whose arrows, or more precisely, cospans of arrows, can be used to model the composition of service components (as defined, for example, in [[FLBo7](#); [FLB11](#); [FL13a](#)]); second, that the discovery

of a service module to be bound to a given application can be formalized in terms of logical specifications of required and provided services, ensuring that the specification of the provided service refines the specification of the requested service. To this end, we explore the model-theoretic notion of refinement advanced in [ST88b], except that, in the present setting, the structures over which specifications are evaluated are morphisms into ground orchestrations, that is into orchestrations that have no dependencies on external services. The motivation for this choice is that, in general, the semantics of non-ground orchestrations is open: the (observable) behaviour exhibited by non-ground orchestrations varies according to the external services that they may procure at run time. With these remarks in mind, we arrive at the following concept of orchestration scheme.

DEFINITION 5.1.1 (Orchestration scheme). An *orchestration scheme* is a quadruple $\mathcal{O} = \langle \text{Orc}, \text{Spec}, \text{Grc}, \text{Prop} \rangle$ consisting of

- a category Orc of *orchestrations* and *orchestration morphisms*,
- a functor $\text{Spec}: \text{Orc} \rightarrow \text{Set}$ defining, for every orchestration σ , the set $\text{Spec}(\sigma)$ of *service specifications* over σ ,
- a full subcategory $\text{Grc} \subseteq \text{Orc}$ of *ground orchestrations*, and
- a functor $\text{Prop}: \text{Grc} \rightarrow \text{Set}$ defining, for every ground orchestration \mathfrak{g} , the natural subset $\text{Prop}(\mathfrak{g}) \subseteq \text{Spec}(\mathfrak{g})$ ¹ of *properties* of \mathfrak{g} (specifications that are guaranteed to hold when evaluated over \mathfrak{g}).

To illustrate our categorical approach to orchestrations, we consider two main running examples: program expressions as discussed in [Fia12; see also Mor94], which provide a way of constructing structured (sequential) programs through design-time discovery and binding, and the theory of asynchronous relational networks put forward in [FL13b], which emphasizes the role of services as an interface mechanism for software components that can be composed through run-time discovery and binding.

5.1.1 PROGRAM EXPRESSIONS

The view that program expressions can be seen as defining ‘service orchestrations’ through which structured programs can be built in a compositional way originates from [Fia12]. Intuitively, we can see the rules of the Hoare calculus [see Hoa69] as defining ‘clauses’ in the sense of logic programming, where unification is controlled through the refinement of pre/post-

¹ By describing $\text{Prop}(\mathfrak{g})$ as a *natural subset* of $\text{Spec}(\mathfrak{g})$ we mean that the family of inclusions $(\text{Prop}(\mathfrak{g}) \subseteq \text{Spec}(\mathfrak{g}))_{\mathfrak{g} \in |\text{Grc}|}$ defines a natural transformation from Prop to $(\text{Grc} \subseteq \text{Orc}) \natural \text{Spec}$.

conditions as specifications of provided/required services, and resolution binds program statements (terms) to variables in program expressions.

In [Figure 5.1](#), we depict Hoare rules in a notation that is closer to that of service modules, which also brings out their clausal form: the specification (a pair of a pre- and a post-condition) on the left-hand side corresponds to the consequent of the clause (which relates to a ‘provides-point’ of the service), while those on the right-hand side correspond to the antecedent of the clause (i.e. to the ‘requires-points’ of the service) – the specifications of what remains to be discovered and bound to the program expression (the ‘service orchestration’ inside the box) to produce a program. In [Figure 5.2](#), we retrace Hoare’s original example of constructing a program that computes the quotient and the remainder resulting from the division of two natural numbers as an instance of the unification and resolution mechanisms particular to logic programming. We will further discuss these mechanisms in more detail in [Section 5.2.3](#).

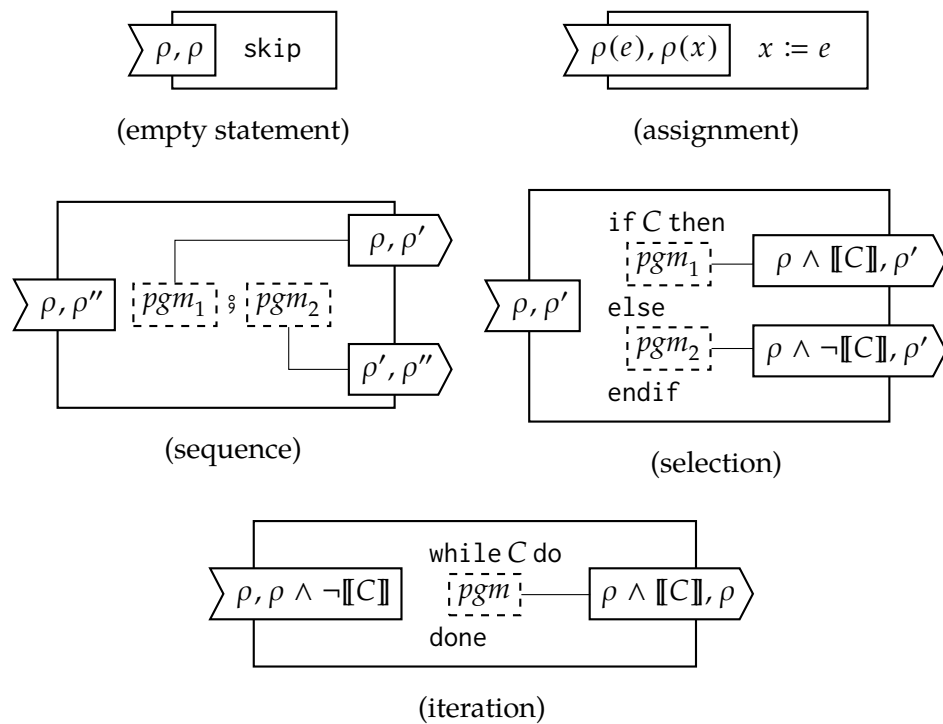


FIGURE 5.1. Program modules

The formal description of program expressions that we consider here follows the presentation given in [\[GM96\]](#) of the algebraic semantics of programs, except that, instead of the theory of many-sorted algebra, we rely on the theory of preordered algebra developed in [\[DF98\]](#), whose institution we denote by [POA](#). In this context, signatures are ordinary algebraic signatures

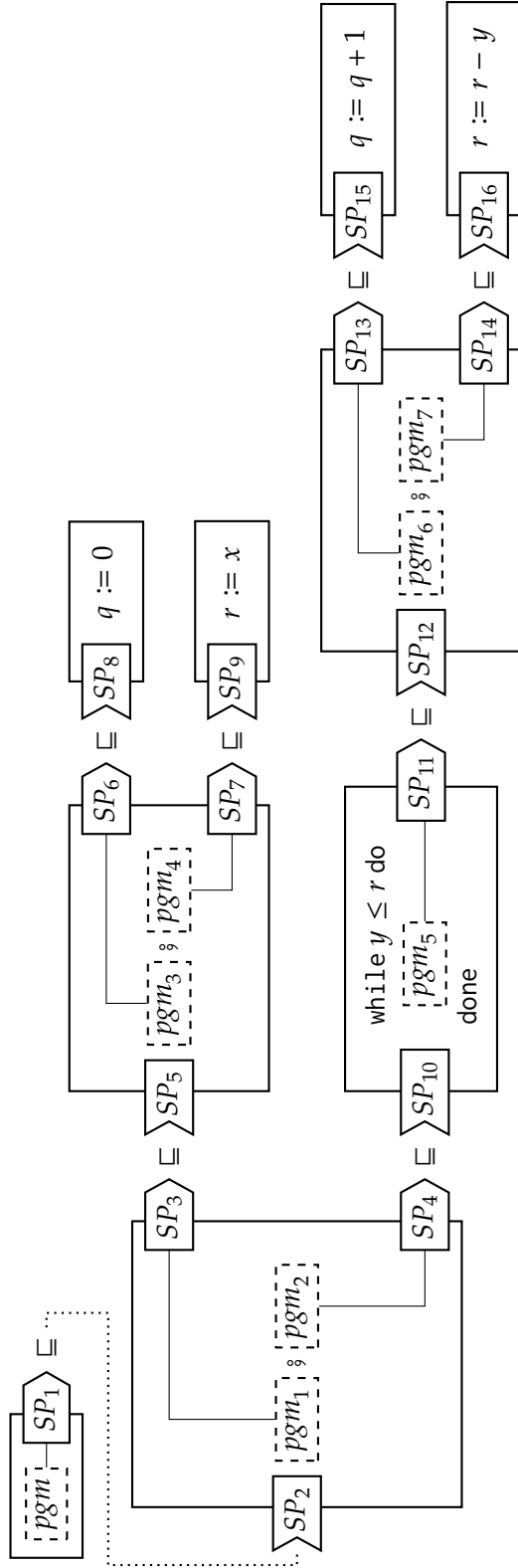


FIGURE 5.2. The derivation of a program that computes the quotient q and the remainder r obtained from the division of x by y

- $SP_1 : true, \llbracket x = q * y + r \rrbracket \wedge \llbracket r < y \rrbracket$
- $SP_2 : true, \llbracket x = q * y + r \rrbracket \wedge \llbracket r < y \rrbracket$
- $SP_3 : true, \llbracket x = q * y + r \rrbracket$
- $SP_4 : \llbracket x = q * y + r \rrbracket, \llbracket x = q * y + r \rrbracket \wedge \llbracket r < y \rrbracket$
- $SP_5 : true, \llbracket x = q * y + r \rrbracket$
- $SP_6 : true, \llbracket x = q * y + x \rrbracket$
- $SP_7 : \llbracket x = q * y + x \rrbracket, \llbracket x = q * y + r \rrbracket$
- $SP_8 : \llbracket x = 0 * y + x \rrbracket, \llbracket x = q * y + x \rrbracket$
- $SP_9 : \llbracket x = q * y + x \rrbracket, \llbracket x = q * y + r \rrbracket$
- $SP_{10} : \llbracket x = q * y + r \rrbracket, \llbracket x = q * y + r \rrbracket \wedge \neg \llbracket y \leq r \rrbracket$
- $SP_{11} : \llbracket x = q * y + r \rrbracket \wedge \llbracket y \leq r \rrbracket, \llbracket x = q * y + r \rrbracket$
- $SP_{12} : \llbracket x = (q + 1) * y + (r - y) \rrbracket, \llbracket x = q * y + r \rrbracket$
- $SP_{13} : \llbracket x = (q + 1) * y + (r - y) \rrbracket, \llbracket x = q * y + (r - y) \rrbracket$
- $SP_{14} : \llbracket x = q * y + (r - y) \rrbracket, \llbracket x = q * y + r \rrbracket$
- $SP_{15} : \llbracket x = (q + 1) * y + (r - y) \rrbracket, \llbracket x = q * y + (r - y) \rrbracket$
- $SP_{16} : \llbracket x = q * y + (r - y) \rrbracket, \llbracket x = q * y + r \rrbracket$

whose denotation is defined over the category of preorders rather than that of sets, with models interpreting the sorts as preordered sets and the operation symbols as monotonic functions. The sentences are built as in first-order logic based on two kinds of atoms: *equational atoms* $l = r$ and *preorder atoms (transitions)* $l \rightarrow r$, where l and r are terms of the same sort; the latter are satisfied by a preordered algebra A if and only if the interpretations of l and r in A belong to the preorder relation of the carrier of their sort.

In order to fully define the orchestration scheme of program expressions we assume that the programming language we have chosen to analyse is specified through a many-sorted signature $\langle S, F \rangle$ equipped with

- a distinguished set of sorts $S^{\text{Pgm}} \subset S$ corresponding to the types of executable expressions supported by the language, and sorts $State, Config \in S \setminus S^{\text{Pgm}}$ capturing the states of the programs and the various configurations that may arise upon their execution, respectively;
- operation symbols $\langle _ \rangle: State \rightarrow Config$ and $\langle _, _ \rangle: eXp\ State \rightarrow Config$, for $eXp \in S^{\text{Pgm}}$, which we regard as constructor operators for the sort $Config$;
- a (sortwise infinite) S^{Pgm} -indexed set Var of program variables together with state variables $st, st': State$, used to refer to the states that precede or result from executions; and
- a preordered $\langle S, F \rangle$ -algebra A that describes the semantics of the programming language through the preorder relation associated with the sort $Config$.²

EXAMPLE 5.1.2. The premises that we consider within this section are weak enough to allow the proposed algebraic framework to accommodate a wide variety of programming languages. For instance, the program expressions underlying the modules depicted in [Figure 5.1](#) are simply terms of sort Pgm that can be formed based on the following five operation symbols (written using the mixfix notation of CAFEOBJ [DF98] and CASL [Mos04]):

(empty statement) $\text{skip}: \rightarrow \text{Pgm}$,
 (assignment) $_ := _: \text{Id AExp} \rightarrow \text{Pgm}$,
 (sequence) $_ \S _: \text{Pgm Pgm} \rightarrow \text{Pgm}$,
 (selection) $\text{if } _ \text{ then } _ \text{ else } _ \text{ endif}: \text{Cond Pgm Pgm} \rightarrow \text{Pgm}$,
 (iteration) $\text{while } _ \text{ do } _ \text{ done}: \text{Cond Pgm} \rightarrow \text{Pgm}$.

In this case, the many-sorted signature of interest can be described as follows:

- Besides the sort Pgm of program expressions, the set S also comprises the

² Alternatively, one could use a theory presentation or a structured specification instead of A .

sorts Id of identifiers, AExp of arithmetic expressions, and Cond of conditions, as well as the two essential sorts State and Config .

- The operation symbols in F are those that correspond to the five types of program expressions presented above (empty statement, assignment, sequence, selection, and iteration) together with the usual operation symbols associated with identifiers (like $\$_{_} : \text{Id} \rightarrow \text{AExp}$, which is used to denote the value bound to a given identifier), with the addition, subtraction and multiplication of arithmetic expressions, and with the atoms and Boolean connectives specific to conditions. In addition, F also contains the two operation symbols $\langle _ \rangle : \text{State} \rightarrow \text{Config}$ and $\langle _, _ \rangle : \text{PgmState} \rightarrow \text{Config}$.

The only expressions considered to be executable are those of sort Pgm , hence S^{Pgm} is just the singleton set $\{\text{Pgm}\}$. Furthermore, the obvious candidates for State and Config are the sorts State and Config , and a similar observation can be stated for the operation symbols $\langle _ \rangle$ and $\langle _, _ \rangle$. As for the preordered algebra A that defines the semantics of the programming language, we note that, apart from State and Config , the interpretations of sorts and operation symbols are straightforward. With respect to states, the carrier A_{State} can be regarded simply as the discrete preorder of partial functions from identifiers to values (natural numbers). Based on this, we can define configurations (elements of sort Config) either as plain states, corresponding to those configurations that are considered to be ‘final’, or as pairs $\langle \text{pgm}, \text{st} \rangle$, which indicate the start of the execution of a program expression pgm in a state st . Finally, the transitions between such configurations can be defined using natural operational-semantics rules as in [Kah87]: for instance, for any two program expressions pgm_1 and pgm_2 , and any two states st_1 and st_2 , if there exists a transition $\langle \text{pgm}_1, \text{st}_1 \rangle \rightarrow \langle \text{st}_2 \rangle$, then there exists a transition $\langle \text{pgm}_1 \ ; \ \text{pgm}_2, \text{st}_1 \rangle \rightarrow \langle \text{pgm}_2, \text{st}_2 \rangle$ as well.

Algebraic signatures having the aforementioned additional structure induce orchestration schemes in a canonical way, as follows.

ORCHESTRATIONS. The *orchestrations* are program expressions, that is terms $\text{pgm} : \text{eXp}$ over the signature $\langle S, F \cup \text{Var} \rangle$, usually denoted simply by pgm if there is no danger of confusion, such that eXp is a sort in S^{Pgm} . The arrows through which they are linked generalize the subterm relations; in this sense, a *morphism* $\langle \psi, \pi \rangle$ between programs $\text{pgm}_1 : \text{eXp}_1$ and $\text{pgm}_2 : \text{eXp}_2$ consists of

- a substitution $\psi : \text{var}(\text{pgm}_1) \rightarrow \text{var}(\text{pgm}_2)$, mapping the variables that occur in pgm_1 to program expressions defined over the variables of pgm_2 , and
- a position π in pgm_2 , that is a sequence of natural numbers that precisely identifies a particular occurrence of a subterm $\text{pgm}_2 \upharpoonright_{\pi}$ of pgm_2 ,

such that $\psi^{\text{tm}}(pgm_1) = pgm_2 \upharpoonright_{\pi}$. Their *composition* is defined componentwise, in a way that ensures the commutativity of the following diagram.

$$\begin{array}{ccccc} pgm_1 : eXp_1 & \xrightarrow{\langle \psi_1, \pi_1 \rangle} & pgm_2 : eXp_2 & \xrightarrow{\langle \psi_2, \pi_2 \rangle} & pgm_3 : eXp_3 \\ & \searrow & \text{---} & \nearrow & \\ & & \langle \psi_1 \circ \psi_2, \pi_2 \circ \pi_1 \rangle & & \end{array}$$

SPECIFICATIONS. For each program expression $pgm : eXp$, a (*program*) *specification* is a triple of the form $\iota : [\rho, \rho']$, where ι is a position in pgm indicating the ‘subprogram’ of pgm whose behaviour is being analysed,³ and ρ and ρ' are *pre-* and *post-conditions* associated with $pgm \upharpoonright_{\iota}$, formalized as (quantifier-free) POA-sentences over the signature $\langle S, F \cup \{st : State\} \rangle$. The intuitive interpretation is the usual one:

Whenever the program $pgm \upharpoonright_{\iota}$ is executed in an initial state that satisfies the pre-condition ρ , and the execution terminates, the resulting final state satisfies the post-condition ρ' .

Note, however, that specifications cannot be evaluated over arbitrary program expressions because, due to the presence of program variables, some of the programs may not support a well-defined notion of execution. We will address this aspect in [Section 5.2](#) by taking into account translations of specifications along morphisms whose codomains are ground program expressions. For now, it suffices to mention that the translation of a program specification $\iota : [\rho, \rho']$ of $pgm_1 : eXp_1$ along a morphism $\langle \psi, \pi \rangle : (pgm_1 : eXp_1) \rightarrow (pgm_2 : eXp_2)$ is defined as $(\pi \cdot \iota) : [\psi(\rho), \psi(\rho')]$ of $pgm_2 : eXp_2$.

GROUND ORCHESTRATIONS AND PROPERTIES. As expected, *ground program expressions* are just expressions that do not contain variables: $\langle S, F \rangle$ -terms $pgm : eXp$ whose sort eXp belongs to S^{pgm} . Consequently, they have a well-defined operational semantics, which means that we can check whether or not they meet the requirements of a given specification.

A specification $\iota : [\rho, \rho']$ is a *property* of a ground program $pgm : eXp$ if the following satisfaction condition holds for the preordered algebra A :

$$A \models^{\text{POA}} \forall \{st, st' : State\} \cdot (\rho(st) \wedge \langle pgm \upharpoonright_{\iota}, st \rangle \rightarrow \langle st' \rangle) \Rightarrow \rho'(st').$$

To keep the notation simple and, at the same time, to emphasize the roles of st and st' , in the above POA-sentence, $\rho(st)$ is used just as another name for

³ The first component of specifications may be encountered in the literature [e.g. in [Mor94](#)] with a different meaning: the frame of the considered specification, that is the set of identifiers whose values may change during the execution of the program.

ρ , while $\rho'(st')$ is the sentence derived from ρ' by replacing the variable st with st' .⁴ The same notational convention is used in Figure 5.1 to represent the specification attached to the assignment expression. In that case, ρ is assumed to be a sentence defined not only over st : State, but also over a variable v : AExp; the sentences $\rho(e)$ and $\rho(x)$ are then derived from ρ by replacing v with e and x (regarded as an atomic arithmetic expression), respectively. Another notation used in Figure 5.1 (and also in Figure 5.2) is $\llbracket C \rrbracket$, where C is a term of sort Cond; this follows Iverson's convention [see Ive62; and also GKP94], and corresponds to an atomic POA-sentence that captures the semantics of the condition C .

We conclude the presentation of orchestrations as program expressions with Proposition 5.1.3 below, which guarantees that properties form natural subsets of the sets of specifications; in other words, that the morphisms of ground programs preserve properties.

PROPOSITION 5.1.3. *Let $\langle \psi, \pi \rangle: (pgm_1: eXp_1) \rightarrow (pgm_2: eXp_2)$ be a morphism between ground programs. For every property $\iota: [\rho, \rho']$ of $pgm_1: eXp_1$, the specification $\text{Spec}(\psi, \pi)(\iota: [\rho, \rho'])$ is a property of $pgm_2: eXp_2$.*

PROOF. By the definition of the translation of specifications along morphisms of program expressions, $\text{Spec}(\psi, \pi)(\iota: [\rho, \rho'])$ is a property of $pgm_2: eXp_2$ if and only if

$$A \models^{\text{POA}} \forall \{st, st': \text{State}\} \cdot \underbrace{(\psi(\rho)(st))}_{\rho(st)} \wedge \underbrace{\langle pgm_2 \upharpoonright_{\pi \cdot \iota}, st \rangle}_{pgm_1 \upharpoonright_{\iota}} \rightarrow \langle st' \rangle \Rightarrow \underbrace{\psi(\rho')(st')}_{\rho'(st')}.$$

To prove this, notice that all morphisms of ground program expressions share the same underlying substitution: the identity of \emptyset . Therefore, $\psi(\rho) = \rho$, $\psi(\rho') = \rho'$, and $pgm_2 \upharpoonright_{\pi \cdot \iota} = pgm_2 \upharpoonright_{\pi} \upharpoonright_{\iota} = \psi^{\text{tm}}(pgm_1) \upharpoonright_{\iota} = pgm_1 \upharpoonright_{\iota}$, from which we immediately deduce that both the evaluation of $\iota: [\rho, \rho']$ in $pgm_1: eXp_1$ and that of $\text{Spec}(\psi, \pi)(\iota: [\rho, \rho'])$ in $pgm_2: eXp_2$ correspond to the satisfaction by A of the same POA-sentence. \square

5.1.2 ASYNCHRONOUS RELATIONAL NETWORKS

Asynchronous relational networks as developed in [FL13b] uphold a significantly different perspective on services: the emphasis is put not on the role of services in addressing design-time organisational aspects of complex, interconnected systems, but rather on their role in managing the run-time

⁴ Formally, the sentences $\rho(st)$ and $\rho'(st')$ are obtained by translating ρ and ρ' along the $\langle S, F \rangle$ -substitutions $\{st\} \rightarrow \{st, st'\}$ given by $st \mapsto st$ and $st \mapsto st'$, respectively.

interactions that are involved in such systems. In this section, we consider a variant of the original theory of asynchronous relational networks that relies on hypergraphs instead of graphs, and uses ω -automata [Tho90; see also PP04] instead of sets of traces as models of behaviour.

The notions discussed in this context depend upon elements of linear temporal logic, and are introduced through syntactic structures that correspond to specific temporal signatures and signature morphisms. However, the proposed theory is largely independent of any logical framework of choice – similarly to the way program expressions can be defined over a variety of algebraic signatures – and can be easily adapted to any institution for which

- 1 *the category of signatures is (finitely) cocomplete,*
- 2 *there exist cofree models along every signature morphism, meaning that the reduct functors determined by signature morphisms admit right adjoints,*
- 3 *the category of models of every signature has (finite) products, and*
- 4 *all model homomorphisms reflect the satisfaction of sentences.*

In addition to the above requirements, we implicitly assume, as in many other works on institutions [see e.g. Dia08; and also ST11, for more details], that the considered logical system is closed under isomorphisms, meaning that the satisfaction of sentences is invariant with respect to isomorphisms of models. This property holds in most institutions; in particular, it holds in the variant of temporal logic that we use here as a basis for the construction of the orchestration scheme of asynchronous relational networks.

LINEAR TEMPORAL LOGIC. In order to capture a more operational notion of service orchestration, we work with an automata-based variant of the institution LTL of linear temporal logic [FC96]. This logical system, denoted ALTL, has the same syntax as LTL, which means that signatures are arbitrary sets of *actions*, and that signature morphisms are just functions. With respect to sentences, for any signature A , the set of A -sentences is defined as the least set containing the actions in A that is closed under standard Boolean connectives and under the temporal operators *next* (\circ) and *until* (\cup). As usual, the derived temporal sentences *eventually* ρ (written $\diamond\rho$) and *always* ρ (written $\square\rho$) stand for $true \cup \rho$ and $\neg(true \cup \neg\rho)$ respectively.

The semantics of ALTL is defined over (non-deterministic) Muller automata [Mul63] instead of the more conventional temporal models. This means that, in the present setting, the models of a signature A are *Muller automata* $\Lambda = \langle Q, \mathcal{P}(A), \Delta, I, \mathcal{F} \rangle$, which consist, besides the (finite) set Q of *states*, the

alphabet $\mathcal{P}(A)$, and the transition relation $\Delta \subseteq Q \times \mathcal{P}(A) \times Q$, of a subset $I \subseteq Q$ of initial states, and of a subset $\mathcal{F} \subseteq \mathcal{P}(Q)$ of (non-empty) final-state sets.

The satisfaction relation is still essentially based upon the satisfaction between traces and (linear) temporal sentences: an automaton Λ satisfies a sentence ρ if every trace accepted by Λ satisfies ρ . To be more precise, let us first recall that a *trace* over A is an (infinite) sequence $\lambda \in \mathcal{P}(A)^\omega$, and that a *run* of an automaton Λ defined as above on a trace λ is a state sequence $\varrho \in Q^\omega$ such that $\varrho(0) \in I$ and $(\varrho(i), \lambda(i), \varrho(i+1)) \in \Delta$ for every $i \in \omega$. A run ϱ is said to be *successful* if its infinity set, $\text{Inf}(\varrho)$, that is the set of states that occur infinitely often in ϱ , is a member of \mathcal{F} . Then, a trace λ is *accepted* by Λ if there exists a successful run of Λ on λ . Finally, given a trace λ (that can be presumed to be accepted by Λ) and $i \in \omega$, we use the notation $\lambda(i..)$ to indicate the suffix of λ that starts at $\lambda(i)$. The satisfaction of temporal sentences by traces can now be defined by structural induction, as follows:

$$\begin{aligned} \lambda \vDash a & \text{ if and only if } a \in \lambda(0), \\ \lambda \vDash \neg\rho & \text{ if and only if } \lambda \not\vDash \rho, \\ \lambda \vDash \bigvee E & \text{ if and only if } \lambda \vDash \rho \text{ for some } \rho \in E, \\ \lambda \vDash \bigcirc\rho & \text{ if and only if } \lambda(1..) \vDash \rho, \text{ and} \\ \lambda \vDash \rho_1 \mathcal{U} \rho_2 & \text{ if and only if } \lambda(i..) \vDash \rho_2 \text{ for some } i \in \omega \\ & \text{ and } \lambda(j..) \vDash \rho_1 \text{ for all } j < i, \end{aligned}$$

where a is an action in A , ρ , ρ_1 , and ρ_2 are A -sentences, and E is a finite set of A -sentences.

One can easily see that the first of the hypotheses 1–4 that form the basis of the present study of asynchronous relational networks is satisfied by ALTL, as it corresponds to a well-known result about the existence of small colimits in Set . In order to check that the remaining three properties hold as well, let us first recall that a *homomorphism* $h: \Lambda_1 \rightarrow \Lambda_2$ between Muller automata $\Lambda_1 = \langle Q_1, \mathcal{P}(A), \Delta_1, I_1, \mathcal{F}_1 \rangle$ and $\Lambda_2 = \langle Q_2, \mathcal{P}(A), \Delta_2, I_2, \mathcal{F}_2 \rangle$ (over the same alphabet) is a function $h: Q_1 \rightarrow Q_2$ such that $(h(p), \alpha, h(q)) \in \Delta_2$ whenever $(p, \alpha, q) \in \Delta_1$, $h(I_1) \subseteq I_2$, and $h(\mathcal{F}_1) \subseteq \mathcal{F}_2$. We also note that for any map $\sigma: A \rightarrow A'$ (i.e. for any signature morphism) and any Muller automaton $\Lambda' = \langle Q', \mathcal{P}(A'), \Delta', I', \mathcal{F}' \rangle$, the *reduct* $\Lambda' \upharpoonright_\sigma$ is the automaton $\langle Q', \mathcal{P}(A), \Delta' \upharpoonright_\sigma, I', \mathcal{F}' \rangle$ with the same states, initial states, and final-state sets as Λ' , and with the transition relation given by $\Delta' \upharpoonright_\sigma = \{(p', \sigma^{-1}(\alpha'), q') \mid (p', \alpha', q') \in \Delta'\}$.

The following results enable us to use the institution ALTL as a foundation for the subsequent development of asynchronous relational networks. In

particular, [Proposition 5.1.4](#) ensures the existence of cofree Muller automata along signature morphisms; [Proposition 5.1.5](#) allows us to form products of Muller automata based on a straightforward categorical interpretation of the fact that the sets of traces accepted by Muller automata, that is the regular ω -languages, are closed under intersection [see e.g. [Thogo](#)]; and finally, [Proposition 5.1.6](#) guarantees that all model homomorphisms reflect the satisfaction of temporal sentences.

PROPOSITION 5.1.4. *For every morphism of ALTL -signatures $\sigma: A \rightarrow A'$, the reduct functor $_ \upharpoonright_\sigma: \text{Mod}^{\text{ALTL}}(A') \rightarrow \text{Mod}^{\text{ALTL}}(A)$ admits a right adjoint, which we denote by $(_)^\sigma$.*

PROOF. According to a general category-theoretic result concerning adjoint functors, it suffices to show that for any automaton Λ over the alphabet $\mathcal{P}(A)$ there exists a universal arrow from $_ \upharpoonright_\sigma$ to Λ . Let us thus consider a Muller automaton $\Lambda = \langle Q, \mathcal{P}(A), \Delta, I, \mathcal{F} \rangle$ over $\mathcal{P}(A)$. We define the automaton $\Lambda^\sigma = \langle Q, \mathcal{P}(A'), \Delta^\sigma, I, \mathcal{F} \rangle$ over the alphabet $\mathcal{P}(A')$ by

$$\Delta^\sigma = \{(p, \alpha', q) \mid (p, \sigma^{-1}(\alpha'), q) \in \Delta\}.$$

It is straightforward to verify that the identity map 1_Q defines a homomorphism of automata $\Lambda^\sigma \upharpoonright_\sigma \rightarrow \Lambda$: for any transition $(p, \alpha, q) \in \Delta^\sigma \upharpoonright_\sigma$, by the definition of the reduct functor $_ \upharpoonright_\sigma$, there exists a subset $\alpha' \subseteq A'$ such that $\sigma^{-1}(\alpha') = \alpha$ and $(p, \alpha', q) \in \Delta^\sigma$; given the definition above of Δ^σ , it follows that $(p, \sigma^{-1}(\alpha'), q) \in \Delta$, and hence $(p, \alpha, q) \in \Delta$.

$$\begin{array}{ccc} \Lambda & \xleftarrow{1_Q} & \Lambda^\sigma \upharpoonright_\sigma \\ & \searrow h & \uparrow h \\ & & \Lambda' \upharpoonright_\sigma \end{array} \qquad \begin{array}{c} \Lambda^\sigma \\ \uparrow h \\ \Lambda' \end{array}$$

Let us now assume that $h: \Lambda' \upharpoonright_\sigma \rightarrow \Lambda$ is another homomorphism of automata, with $\Lambda' = \langle Q', \mathcal{P}(A'), \Delta', I', \mathcal{F}' \rangle$. Then, for any transition (p', α', q') of Λ' , by the definition of the functor $_ \upharpoonright_\sigma$, we have $(p', \sigma^{-1}(\alpha'), q') \in \Delta' \upharpoonright_\sigma$. Since h is a homomorphism, $(h(p'), \sigma^{-1}(\alpha'), h(q')) \in \Delta$; this further implies, by the definition of Δ^σ , that $(h(p'), \alpha', h(q')) \in \Delta^\sigma$. As a result, the map h is also a homomorphism of automata $\Lambda' \rightarrow \Lambda^\sigma$. Even more, it is obviously the unique homomorphism $\Lambda' \rightarrow \Lambda^\sigma$ (in the category of automata over $\mathcal{P}(A')$) such that $h \circ 1_Q = h$ in the category of automata over $\mathcal{P}(A)$. \square

PROPOSITION 5.1.5. *For any set of actions A , the category $\text{Mod}^{\text{ALTL}}(A)$ of Muller automata defined over the alphabet $\mathcal{P}(A)$ admits (finite) products.*

PROOF. Let $(\Lambda_i)_{i \in J}$ be a (finite) family of Muller automata over the alphabet $\mathcal{P}(A)$, with Λ_i given by $\langle Q_i, \mathcal{P}(A), \Delta_i, I_i, \mathcal{F}_i \rangle$ for every $i \in J$. We define the automaton $\Lambda = \langle Q, \mathcal{P}(A), \Delta, I, \mathcal{F} \rangle$ by

$$\begin{aligned} Q &= \prod_{i \in J} Q_i, \\ \Delta &= \{(p, \alpha, q) \mid (p(i), \alpha, q(i)) \in \Delta_i \text{ for all } i \in J\}, \\ I &= \prod_{i \in J} I_i, \text{ and} \\ \mathcal{F} &= \{S \subseteq Q \mid \pi_i(S) \in \mathcal{F}_i \text{ for all } i \in J\}, \end{aligned}$$

where the functions $\pi_i: Q \rightarrow Q_i$ are the corresponding projections of the Cartesian product $\prod_{i \in J} Q_i$. By construction, it immediately follows that for every $i \in J$, the map π_i defines a homomorphism of automata $\Lambda \rightarrow \Lambda_i$. Even more, one can easily see that for any other family of homomorphisms $(h_i: \Lambda' \rightarrow \Lambda_i)_{i \in J}$, with Λ' given by $\langle Q', \mathcal{P}(A'), \Delta', I', \mathcal{F}' \rangle$, the unique map $h: Q' \rightarrow Q$ such that $h \circ \pi_i = h_i$, for all $i \in J$, defines a homomorphism of automata as well. Therefore, the automaton Λ , together with the projections $(\pi_i)_{i \in J}$, form the product of $(\Lambda_i)_{i \in J}$. \square

PROPOSITION 5.1.6. *Let $h: \Lambda_1 \rightarrow \Lambda_2$ be a homomorphism between automata Λ_1 and Λ_2 defined over an alphabet $\mathcal{P}(A)$. Every temporal sentence over A that is satisfied by Λ_2 is also satisfied by Λ_1 .*

PROOF. Suppose that the automaton Λ_i is given by $\langle Q_i, \mathcal{P}(A), \Delta_i, I_i, \mathcal{F}_i \rangle$, for $i \in \{1, 2\}$. Since the map $h: Q_1 \rightarrow Q_2$ defines a homomorphism of automata, for every successful run $\rho \in Q_1^\omega$ of Λ_1 on a trace $\lambda \in \mathcal{P}(A)^\omega$, the composition $\rho \circ h$ yields a successful run of Λ_2 on λ . As a result, the automaton Λ_2 accepts all the traces accepted by Λ_1 , which further implies that Λ_1 satisfies all temporal sentences that are satisfied by Λ_2 . \square

SERVICE COMPONENTS. Following [FL13b], we regard service components as networks of processes that interact asynchronously by exchanging messages through communication channels. Messages are considered to be atomic units of communication. They can be grouped either into sets of messages that correspond to processes or channels, or into specific structures, called ports, through which processes and channels can be interconnected.

The ports can be viewed as sets of messages with attached polarities. As in [BZ83; BCT06] we distinguish between outgoing or published messages (labelled with a minus sign), and incoming or delivered messages (labelled with a plus sign).

DEFINITION 5.1.7 (Port). A port M is a pair $\langle M^-, M^+ \rangle$ of disjoint (finite) sets of *published* and *delivered messages*. The set of all *messages* of M is given by $M^- \cup M^+$ and is often denoted simply by M . Every port M defines the set of *actions* A_M as the union $A_{M^-} \cup A_{M^+}$, where

- A_{M^-} is the set $\{m! \mid m \in M^-\}$ of *publication actions* and
- A_{M^+} is the set $\{m; \mid m \in M^+\}$ of *delivery actions*.

Processes are defined by sets of interaction points labelled with ports and by automata that describe their behaviour in terms of observable publication and delivery actions.

DEFINITION 5.1.8 (Process). A process $\langle X, (M_x)_{x \in X}, \Lambda \rangle$ consists of a (finite) set X of *interaction points*, each point $x \in X$ being labelled with a port M_x , and a Muller automaton Λ over $\mathcal{P}(A_M)$, where M is the port given by

$$M^\mp = \bigsqcup_{x \in X} M_x^\mp = \{x.m \mid x \in X, m \in M_x^\mp\}.$$

EXAMPLE 5.1.9. In [Figure 5.3](#), we depict a process named JP (for Journey Planner) that provides directions from a source to a target location. The process interacts with the environment by means of two ports, JP₁ and JP₂. The first of them is used to communicate with potential client applications; the request for directions (including the source and the target locations) is encoded into the incoming message `planJourney`, while the response is represented by the outgoing message `directions`. The second port defines messages that JP exchanges with other processes in order to complete its task; the outgoing message `getRoutes` can be seen as a query for all possible routes between the specified source and target locations, while the incoming messages `routes` and `timetables` define the result of the query and the timetables of the available transport services for the selected routes.

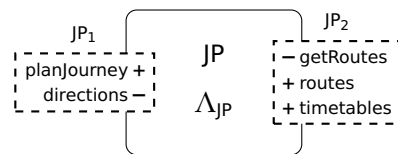


FIGURE 5.3. The process JP

The behaviour of JP is given by the Muller automaton depicted in [Figure 5.4](#), whose final-state sets contain q_0 whenever they contain q_5 . We can describe it informally as follows: whenever the process JP receives a request `planJourney`, it immediately initiates the search for the available routes by sending the

message `getRoutes`; it then waits for the delivery of the routes and of the corresponding timetables (given here by the messages `routes` and `timetables`), and, once it receives both, it compiles the directions and replies to the client.

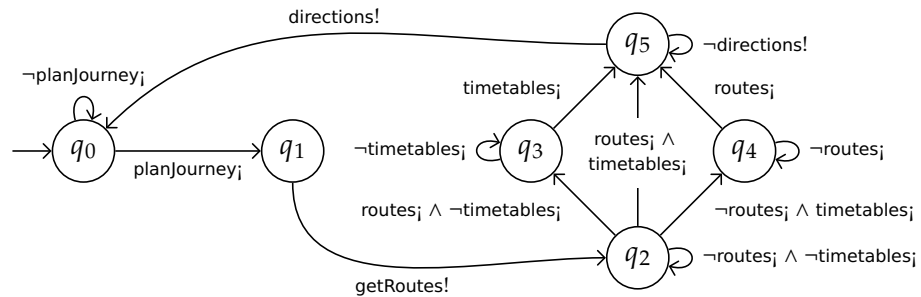


FIGURE 5.4. The automaton $\Lambda_{\mathcal{P}}^5$

REMARK 5.1.10. To generalize Definition 5.1.8 to an arbitrary institution (subject to the four technical assumptions listed at the beginning of the section), we first observe that every polarity-preserving map θ between ports M and M' defines a function $A_\theta: A_M \rightarrow A_{M'}$ – i.e. a morphism of ALTL-signatures – usually denoted simply by θ , that maps every publication action $m!$ to $\theta(m)!$ and every delivery action m_i to $\theta(m)_i$. Moreover, for any process $\langle X, (M_x)_{x \in X}, \Lambda \rangle$, the injections $(x._: A_{M_x} \rightarrow A_M)_{x \in X}$ define a coproduct in the category of ALTL-signatures. This allows us to introduce an abstract notion of process as a triple $\langle X, (\iota_x: \Sigma_x \rightarrow \Sigma)_{x \in X}, \Lambda \rangle$ that consists of a set X of *interaction points*, each point $x \in X$ being labelled with a *port signature* Σ_x , a *process signature* Σ together with morphisms $\iota_x: \Sigma_x \rightarrow \Sigma$ for $x \in X$ (usually defining a coproduct), and a model Λ of Σ .

Processes communicate by transmitting messages through channels. As in [BZ83; FL13b], channels are bidirectional: they may transmit both incoming and outgoing messages.

DEFINITION 5.1.11 (Channel). A *channel* is a pair $\langle M, \Lambda \rangle$ that consists of a (finite) set M of *messages* and a Muller automaton Λ over the alphabet $\mathcal{P}(A_M)$, where A_M is given by the union $A_M^- \cup A_M^+$ of the sets of actions $A_M^- = \{m! \mid m \in M\}$ and $A_M^+ = \{m_i \mid m \in M\}$.

Note that channels do not provide any information about the communicating entities. In order to enable given processes to exchange messages, channels need to be attached to their ports, thus forming connections.

⁵ In the graphical representation, transitions are labelled with propositional sentences, as in [AS87]; this means that there exists a transition for any propositional model (i.e. set of actions) of the considered sentence.

DEFINITION 5.1.12 (Connection). A *connection* $\langle M, \Lambda, (\mu_x : M \rightarrow M_x)_{x \in X} \rangle$ between the ports M_x , for $x \in X$, consists of a channel $\langle M, \Lambda \rangle$ and a (finite) family of partial *attachment injections* $(\mu_x : M \rightarrow M_x)_{x \in X}$ such that the set M corresponds to the union $\bigcup_{x \in X} \text{dom}(\mu_x)$ and, for any point $x \in X$,

$$\mu_x^{-1}(M_x^{\mp}) \subseteq \bigcup_{y \in X \setminus \{x\}} \mu_y^{-1}(M_y^{\pm}).$$

This notion of connection generalizes the one found in [FL13b] so that messages can be transmitted between more than two ports. The additional condition ensures in this case that messages are well paired: every published message of M_x , for $x \in X$, is paired with a delivered message of M_y , for $y \in X \setminus \{x\}$, and vice versa. One can also see that for any binary connection the attachment injections have to be total functions; therefore, any binary connection is also a connection in the sense of [FL13b].

EXAMPLE 5.1.13. To illustrate how the process JP can send or receive messages, we consider the connection C depicted in Figure 5.5, which moderates the flow of messages between the port JP₂ and two other ports, R₁ and R₂.

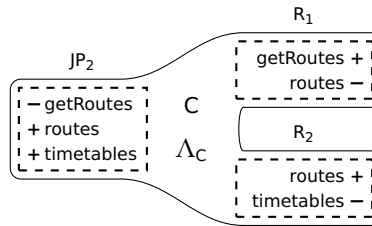


FIGURE 5.5. The Journey Planner's connection

The underlying channel of C is given by the set of messages $M = \{g, r, t\}$ together with the automaton Λ_C that specifies the delivery of all published messages without any delay; Λ_C can be built as the product of the automata Λ_m , for $m \in M$, whose transition map is depicted in Figure 5.6, and whose sets of states are all marked as final.

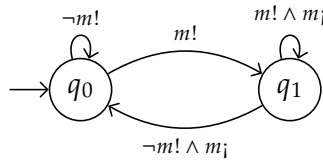


FIGURE 5.6. The automaton Λ_m

The channel is attached to the ports JP₂, R₁, and R₂ through the injections

- $\mu_{JP_2} : M \rightarrow M_{JP_2}$ given by $g \mapsto \text{getRoutes}$, $r \mapsto \text{routes}$ and $t \mapsto \text{timetables}$,

- $\mu_{R_1} : M \rightarrow M_{R_1}$ given by $g \mapsto \text{getRoutes}$ and $r \mapsto \text{routes}$, and
- $\mu_{R_2} : M \rightarrow M_{R_2}$ given by $r \mapsto \text{routes}$ and $t \mapsto \text{timetables}$.

These injections specify the actual senders and receivers of messages. For instance, the message g is delivered only to the port R_1 (because μ_{R_2} is not defined on g), whereas r is simultaneously delivered to both JP_2 and R_2 .

As already suggested in [Examples 5.1.9](#) and [5.1.13](#), processes and connections have dual roles, and they interpret the polarities of messages accordingly. In this sense, processes are responsible for publishing messages (i.e. they regard delivered messages as inputs and published messages as outputs), while connections are responsible for delivering messages. This dual nature of connections can be made explicit by taking into account, for every connection $\langle M, \Lambda, (\mu_x : M \rightarrow M_x)_{x \in X} \rangle$, partial translations $(A_{\mu_x} : A_M \rightarrow A_{M_x})_{x \in X}$ of the actions defined by the channel into actions defined by the ports, as follows:

$$\begin{aligned} \text{dom}(A_{\mu_x}) &= \{m! \mid m \in \mu_x^{-1}(M_x^-)\} \cup \{m_i \mid m \in \mu_x^{-1}(M_x^+)\}, \\ A_{\mu_x}(m!) &= \mu_x(m)! \text{ for all messages } m \in \mu_x^{-1}(M_x^-), \\ A_{\mu_x}(m_i) &= \mu_x(m)_i \text{ for all messages } m \in \mu_x^{-1}(M_x^+). \end{aligned}$$

When there is no danger of confusion, we may also designate the partial translations $A_{\mu_x} : A_M \rightarrow A_{M_x}$ simply by μ_x .

REMARK 5.1.14. Just as in the case of processes, we can define connections based on an arbitrary logical system, without relying on messages. To achieve this goal, note that every connection $\langle M, \Lambda, (\mu_x : M \rightarrow M_x)_{x \in X} \rangle$ determines a family of spans of ALTL-signature morphisms

$$A_M \xleftarrow{\supseteq} \text{dom}(\mu_x) \xrightarrow{\mu_x} A_{M_x}$$

indexed by points $x \in X$. This allows us to consider connections more generally as triples $\langle \Sigma, \Lambda, (\iota_x : \Sigma'_x \rightarrow \Sigma, \mu_x : \Sigma'_x \rightarrow \Sigma_x)_{x \in X} \rangle$ in which the signature Σ and the model Λ of Σ abstract the *channel* component, while the spans $(\iota_x, \mu_x)_{x \in X}$ capture the attachment of port signatures to the channel.

We can now define asynchronous networks of processes as hypergraphs having vertices labelled with ports and hyperedges labelled with processes or connections.

DEFINITION 5.1.15 (Hypergraph). A *hypergraph* $\langle X, E, \gamma \rangle$ consists of a set X of *vertices* or *nodes*, a set E of *hyperedges*, disjoint from X , and an *incidence map*

$\gamma: E \rightarrow \mathcal{P}(X)$ defining, for every hyperedge $e \in E$, a non-empty set $\gamma_e \subseteq X$ of vertices it is incident with.

A hypergraph $\langle X, E, \gamma \rangle$ is said to be *edge-bipartite* if it admits a distinguished partition of E into subsets F and G such that no adjacent hyperedges belong to the same part, meaning that for every two hyperedges $e_1, e_2 \in E$ such that $\gamma_{e_1} \cap \gamma_{e_2} \neq \emptyset$, either $e_1 \in F$ and $e_2 \in G$, or $e_1 \in G$ and $e_2 \in F$.

Hypergraphs have been used extensively in the context of graph-rewriting-based approaches to concurrency, including service-oriented computing [e.g. Fer+05; BGLLo9]. We use them instead of graphs [see FL13b] because they offer a more flexible mathematical framework for handling the notions of variable and variable binding required in Section 5.2.

DEFINITION 5.1.16 (Asynchronous relational network – ARN). An *asynchronous relational network* $\mathfrak{N} = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$ consists of a (finite) edge-bipartite hypergraph $\langle X, P, C, \gamma \rangle$ of *points* $x \in X$, *computation hyperedges* $p \in P$ and *communication hyperedges* $c \in C$, together with

- a port M_x for every point $x \in X$,
- a process $\langle \gamma_p, (M_x)_{x \in \gamma_p}, \Lambda_p \rangle$ for every hyperedge $p \in P$, and
- a connection $\langle M_c, \Lambda_c, (\mu_x^c: M_c \multimap M_x)_{x \in \gamma_c} \rangle$ for every hyperedge $c \in C$.

EXAMPLE 5.1.17. By putting together the process and the connection presented in Examples 5.1.9 and 5.1.13, we obtain the ARN JourneyPlanner depicted in Figure 5.7. Its underlying hypergraph consists of the points $JP_1, JP_2, R_1,$ and R_2 , the computation hyperedge JP , the communication hyperedge C , and the incidence map γ given by $\gamma_{JP} = \{JP_1, JP_2\}$ and $\gamma_C = \{JP_2, R_1, R_2\}$.

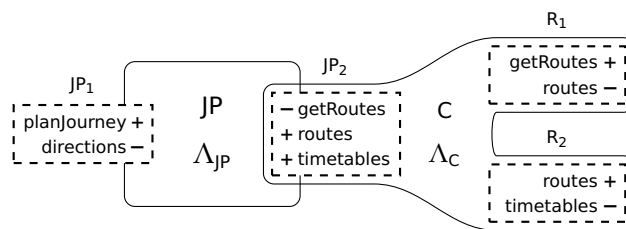


FIGURE 5.7. The ARN JourneyPlanner

THE ORCHESTRATION SCHEME OF ASYNCHRONOUS RELATIONAL NETWORKS.

Let us now focus on the manner in which ARNs can be organized to form an orchestration scheme. This provides a convenient mechanism for dealing with the added complexity of ARNs (of a different nature than the component abstraction considered in [FL13b]) that will prove to be particularly suited

for the present investigation of the logic-programming semantics of services. We begin with a brief discussion on the types of points of ARNS, which will enable us to introduce notions of morphism of ARNS and ground ARN.

An *interaction point* of an ARN \mathfrak{N} is a point of \mathfrak{N} that is not bound to both computation and communication hyperedges. We distinguish between two types of interaction points, called *requires-* and *provides-*points, as follows.

DEFINITION 5.1.18 (Requires- and provides-point). A *requires-point* of an ARN \mathfrak{N} is a point of \mathfrak{N} that is incident only with a communication hyperedge. Similarly, a *provides-point* of \mathfrak{N} is incident only with a computation hyperedge.

For the ARN JourneyPlanner depicted in Figure 5.7, the points R_1 and R_2 are *requires-points* (incident with the communication hyperedge C), whereas JP_1 is a *provides-point* (incident with the computation hyperedge JP).

ORCHESTRATIONS. In order to describe ARNS as orchestrations we first need to equip them with appropriate notions of morphism and composition of morphisms. Morphisms of ARNS correspond to injective homomorphisms between their underlying hypergraphs, and are required to preserve all labels, except those associated with points that, like the *requires-points*, are not incident with computation hyperedges.

DEFINITION 5.1.19 (Homomorphism of hypergraphs). A *homomorphism* h between hypergraphs $\langle X_1, E_1, \gamma^1 \rangle$ and $\langle X_2, E_2, \gamma^2 \rangle$ consists of a pair of functions $h: X_1 \rightarrow X_2$ and $h: E_1 \rightarrow E_2$ ⁶ such that for any vertex $x \in X_1$ and hyperedge $e \in E_1$, $x \in \gamma_e^1$ if and only if $h(x) \in \gamma_{h(e)}^2$.

DEFINITION 5.1.20 (Morphism of ARNS). A *morphism* θ between ARNS $\mathfrak{N}_1 = \langle X_1, P_1, C_1, \gamma^1, M^1, \mu^1, \Lambda^1 \rangle$ and $\mathfrak{N}_2 = \langle X_2, P_2, C_2, \gamma^2, M^2, \mu^2, \Lambda^2 \rangle$ consists of

- an injective homomorphism θ between the hypergraphs $\langle X_1, P_1, C_1, \gamma^1 \rangle$ and $\langle X_2, P_2, C_2, \gamma^2 \rangle$ such that $\theta(P_1) \subseteq P_2$ and $\theta(C_1) \subseteq C_2$, and
- a family θ^{pt} of polarity-preserving injections $\theta_x^{\text{pt}}: M_x^1 \rightarrow M_{\theta(x)}^2$, for $x \in X_1$,

such that

- for every point $x \in X_1$ incident with a computation hyperedge, $\theta_x^{\text{pt}} = 1_{M_x^1}$,
- for every computation hyperedge $p \in P_1$, $\Lambda_p^1 = \Lambda_{\theta(p)}^2$, and

⁶ To simplify the notation, we denote both the translation of vertices and the translation of hyperedges simply by h .

- for every communication hyperedge $c \in C_1$, $M_c^1 = M_{\theta(c)}^2$, $\Lambda_c^1 = \Lambda_{\theta(c)}^2$ and the following diagram commutes, for every point $x \in \gamma_c^1$.

$$\begin{array}{ccc}
 M_c^1 = M_{\theta(c)}^2 & \xrightarrow{\mu_x^{1,c}} & M_x^1 \\
 & \searrow \mu_{\theta(x)}^{2,\theta(c)} & \downarrow \theta_x^{\text{pt}} \\
 & & M_{\theta(x)}^2
 \end{array}$$

It is straightforward to verify that the morphisms of ARNs can be composed in terms of their components. Their composition is associative and has left and right identities given by morphisms that consist solely of set-theoretic identities. We obtain in this way the first result supporting the construction of an orchestration scheme of ARNs .

PROPOSITION 5.1.21. *The morphisms of ARNs form a category, denoted \mathbb{ARN} . \square*

SPECIFICATIONS. To define specifications over given ARNs , we label their points with linear temporal sentences, much in the way we used pre- and post-conditions as labels for positions in terms when defining specifications of program expressions.

DEFINITION 5.1.22 (Specification over an ARN). For any ARN \mathfrak{N} , the set $\text{Spec}(\mathfrak{N})$ of \mathfrak{N} -specifications is the set of pairs $\langle x, \rho \rangle$, usually denoted $@_x \rho$, where x is a point of \mathfrak{N} and ρ is an $\underline{\text{ALTL}}$ -sentence over A_{M_x} , that is over the set of actions defined by the port that labels x .

The *translation* of specifications along morphisms of ARNs presents no difficulties: for every morphism $\theta: \mathfrak{N} \rightarrow \mathfrak{N}'$, the map $\text{Spec}(\theta): \text{Spec}(\mathfrak{N}) \rightarrow \text{Spec}(\mathfrak{N}')$ is given by

$$\text{Spec}(\theta)(@_x \rho) = @_{\theta(x)} \text{Sen}^{\underline{\text{ALTL}}}(\theta_x^{\text{pt}})(\rho)$$

for each point x of \mathfrak{N} and each $\underline{\text{ALTL}}$ -sentence ρ over the actions of x . Furthermore, it can be easily seen that it inherits the functoriality of the translation of $\underline{\text{ALTL}}$ -sentences, thus giving rise to the needed functor $\text{Spec}: \mathbb{ARN} \rightarrow \text{Set}$.

GROUND ORCHESTRATIONS. Morphisms of ARNs can also be regarded as refinements, because they formalize the embedding of networks with an intuitively simpler behaviour into networks that are more complex. This is achieved essentially by mapping each of the requires-points of the source ARN to a potentially non-requires-point of the target ARN , a point which can be looked at as the ‘root’ of a particular subnetwork of the target ARN . To

explain this aspect in more detail we introduce the notions of dependency and ARN defined by a point.

DEFINITION 5.1.23 (Dependency). Let x and y be two points of an ARN \mathfrak{N} . The point x is said to be *dependent* on y if there exists a path from x to y that begins with a computation hyperedge, that is if there exists an alternating sequence $x e_1 x_1 \dots e_n y$ of (distinct) points and hyperedges of the underlying hypergraph $\langle X, P, C, \gamma \rangle$ of \mathfrak{N} such that $x \in \gamma_{e_1}$, $y \in \gamma_{e_n}$, $x_i \in \gamma_{e_i} \cap \gamma_{e_{i+1}}$ for every $1 \leq i < n$, and $e_1 \in P$.

DEFINITION 5.1.24 (Network defined by a point). For any ARN \mathfrak{N} and any point x of \mathfrak{N} , the *network defined by x* (relative to \mathfrak{N}) is the full sub-ARN \mathfrak{N}_x of \mathfrak{N} determined by x and the points on which x is dependent.

One can now see that any morphism of ARNs $\theta: \mathfrak{N}_1 \rightarrow \mathfrak{N}_2$ assigns to each requires-point x of \mathfrak{N}_1 the sub-ARN $\mathfrak{N}_{2,\theta(x)}$ of \mathfrak{N}_2 defined by $\theta(x)$.

EXAMPLE 5.1.25. In Figure 5.8, we outline an extension of the network JourneyPlanner discussed in Example 5.1.17 that is obtained by attaching the processes MS (for Map Services) and TS (for Transport System) to the requires-points R_1 and R_2 of JourneyPlanner. Formally, the link between JourneyPlanner and the resulting ARN JourneyPlannerNet is given by a morphism $\theta: \text{JourneyPlanner} \rightarrow \text{JourneyPlannerNet}$ that preserves all the labels, points and hyperedges of JourneyPlanner, with the exception of the requires-points R_1 and R_2 , which are mapped to MS_1 and TS_1 , respectively.

In this case, the point MS_1 only depends on itself, hence the sub-ARN of JourneyPlannerNet defined by MS_1 , that is the ARN assigned to the requires-point R_1 of JourneyPlanner, is given by the process MS and its port, MS_1 . In contrast, the point JP_1 depends on all the other points of JourneyPlannerNet, and thus it defines the entire ARN JourneyPlannerNet.

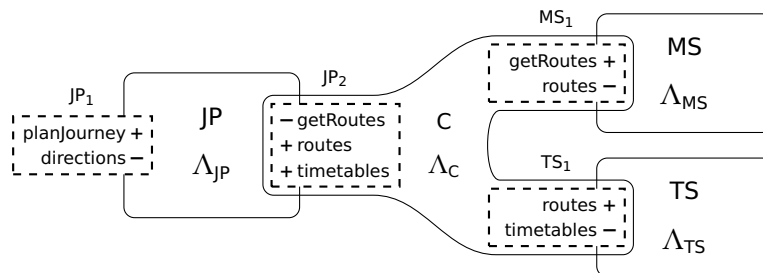


FIGURE 5.8. The ARN JourneyPlannerNet

In view of the above observation, we may consider the requires-points of asynchronous relational networks as counterparts of the (program) variables

used in program expressions, and their morphisms as substitutions. This leads us to the following definition of ground ARNS.

DEFINITION 5.1.26. An ARN is said to be *ground* if it has no requires-points.

PROPERTIES. The evaluation of specifications with respect to ground ARNS relies on the concepts of diagram of a network and automaton (i.e. ALTL -model) defined by a point, whose purpose is to describe the observable behaviour of a ground ARN through one of its points. We start by extending [Remarks 5.1.10](#) and [5.1.14](#) from processes and connections to ARNS.

FACT 5.1.27 (Diagram of an ARN). Every ARN $\mathfrak{N} = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$ defines a diagram $D_{\mathfrak{N}} : \mathbb{J}_{\mathfrak{N}} \rightarrow \text{Sig}^{\text{ALTL}}$ as follows:

- $\mathbb{J}_{\mathfrak{N}}$ is the free preordered category given by the set of objects

$$X \cup P \cup C \cup \{\langle c, x \rangle_{\mathfrak{N}} \mid c \in C, x \in \gamma_c\}$$

and the arrows

- $\{x \rightarrow p \mid p \in P, x \in \gamma_p\}$ for computation hyperedges, and
- $\{c \leftarrow \langle c, x \rangle_{\mathfrak{N}} \rightarrow x \mid c \in C, x \in \gamma_c\}$ for communication hyperedges;
- $D_{\mathfrak{N}}$ is the functor that provides the sets of actions of ports, processes and channels, together with the appropriate mappings between them. For example, given a communication hyperedge $c \in C$ and a point $x \in \gamma_c$,
 - $D_{\mathfrak{N}}(c) = A_{M_c}$, $D_{\mathfrak{N}}(\langle c, x \rangle_{\mathfrak{N}}) = \text{dom}(\mu_x^c)$, $D_{\mathfrak{N}}(x) = A_{M_x}$,
 - $D_{\mathfrak{N}}(\langle c, x \rangle_{\mathfrak{N}} \rightarrow c) = (\text{dom}(\mu_x^c) \subseteq A_{M_c})$, and
 - $D_{\mathfrak{N}}(\langle c, x \rangle_{\mathfrak{N}} \rightarrow x) = \mu_x^c$.

We define the signature of an ARN by taking the colimit of its diagram, which is guaranteed to exist because Sig^{ALTL} (i.e. Set) is finitely cocomplete.

DEFINITION 5.1.28 (Signature of an ARN). The *signature* of an ARN \mathfrak{N} is the colimiting cocone $\xi : D_{\mathfrak{N}} \Rightarrow A_{\mathfrak{N}}$ of the diagram $D_{\mathfrak{N}}$.

The most important construction that allows us to define properties of ground ARNS is the one that defines the observed behaviour of a (ground) network at one of its points.

DEFINITION 5.1.29 (Automaton defined by a point). Let x be a point of a ground ARN \mathfrak{G} . The *observed automaton* Λ_x at x is the reduct $\Lambda_{\mathfrak{G}_x} \upharpoonright_{\xi_x}$, where

- $\mathfrak{G}_x = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$ is the sub-ARN of \mathfrak{G} defined by x ,
- $\xi : D_{\mathfrak{G}_x} \Rightarrow A_{\mathfrak{G}_x}$ is the signature of \mathfrak{G}_x ,

- $\Lambda_{\mathfrak{G}_x}$ is the product automaton $\prod_{e \in P \cup C} \Lambda_e^{\mathfrak{G}_x}$, and
- $\Lambda_e^{\mathfrak{G}_x}$ is the cofree expansion of Λ_e along ξ_e , for any hyperedge $e \in P \cup C$.

EXAMPLE 5.1.30. Consider once again the (ground) ARN represented in [Figure 5.8](#). The automaton defined by the point MS_1 is just $\Lambda_{MS} \upharpoonright_{A_{MS_1}}$; this follows from the observation that the ARN defined by MS_1 consists exclusively of the process MS and the port MS_1 . On the other hand, to obtain the automaton defined by the provides-point JP_1 , one needs to compute the product of the cofree expansions of all four automata Λ_{JP} , Λ_C , Λ_{MS} , and Λ_{TS} . Based on [Propositions 5.1.4](#) and [5.1.5](#), the resulting automaton has to accept precisely the projections to $A_{M_{JP_1}}$ of those traces accepted by Λ_{JP} that are compatible with traces accepted by Λ_C , Λ_{MS} , and Λ_{TS} , in the sense that together they give rise, by amalgamation, to traces over the alphabet of the network.

We now have all the necessary concepts for defining properties of ARNs.

DEFINITION 5.1.31 (Property of an ARN). Let $@_x \rho$ be a specification over a ground ARN \mathfrak{G} . Then $@_x \rho$ is a *property* of \mathfrak{G} if and only if the automaton Λ_x observed at the point x in \mathfrak{G} satisfies (according to the definition of satisfaction in [ALTL](#)) the temporal sentence ρ .

$$\Lambda_x \models^{\text{ALTL}} \rho$$

REMARK 5.1.32. It is important to notice that not only the signature of an ARN, but also the various cofree expansions and products considered in [Definition 5.1.29](#) are unique only up to an isomorphism. Consequently, the automaton defined by a point of a ground ARN is also unique only up to an isomorphism, which means that the closure of [ALTL](#) under isomorphisms plays a crucial role in ensuring that the evaluation of specifications with respect to ground ARNs is well defined.

All we need now in order to complete the construction of the orchestration scheme of asynchronous relational networks is to show that the morphisms of ground ARNs preserve properties. This result depends upon the last of the four hypotheses we introduced at the beginning of the section: the reflection of the satisfaction of sentences by the model homomorphisms of the institution used as foundation for the construction of ARNs.

PROPOSITION 5.1.33. *For every morphism of ground ARNs $\theta: \mathfrak{G}_1 \rightarrow \mathfrak{G}_2$ and every property $@_x \rho$ of \mathfrak{G}_1 , the specification $\text{Spec}(\theta)(@_x \rho)$ is a property of \mathfrak{G}_2 .*

PROOF. Let \mathfrak{G}_1^x and \mathfrak{G}_2^x be the sub-ARNs of \mathfrak{G}_1 and \mathfrak{G}_2 determined by x and $\theta(x)$ respectively, and let us also assume that $\mathfrak{G}_i^x = \langle X_i, P_i, C_i, \gamma^i, M^i, \mu^i, \Lambda^i \rangle$

and that $\xi^i: D_{\mathfrak{G}_i^x} \Rightarrow A_{\mathfrak{G}_i^x}$ is the signature of \mathfrak{G}_i^x , for $i \in \{1, 2\}$. Since $@_x \rho$ is a property of \mathfrak{G}_1 , we know that the automaton Λ_x^1 observed at x in \mathfrak{G}_1 satisfies ρ . We also know that $\theta: \mathfrak{G}_1 \rightarrow \mathfrak{G}_2$ defines the ALTL-signature morphism $\theta_x^{\text{pt}}: A_{M_{1,x}} \rightarrow A_{M_{2,\theta(x)}}$ as the identity of $A_{M_{1,x}}$ (because \mathfrak{G}_1 is ground); hence, the automaton $\Lambda_{\theta(x)}^2$ observed at $\theta(x)$ in \mathfrak{G}_2 is also a model of $A_{M_{1,x}}$.

By [Proposition 5.1.6](#), ALTL-model homomorphisms reflect the satisfaction of sentences; therefore, in order to prove that $\Lambda_{\theta(x)}^2$ satisfies ρ – and in this way, that $\text{Spec}(\theta)(@_x \rho)$ is a property of \mathfrak{G}_2 – it suffices to determine the existence of a homomorphism $\Lambda_{\theta(x)}^2 \rightarrow \Lambda_x^1$. To this end, recall that the automata Λ_x^1 and $\Lambda_{\theta(x)}^2$ correspond to the reducts $\Lambda_{\mathfrak{G}_1^x} \upharpoonright_{\xi_x^1}$ and $\Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\xi_{\theta(x)}^2}$, where, for $i \in \{1, 2\}$,

- $\Lambda_{\mathfrak{G}_i^x}$ is the product $\prod_{e \in P_i \cup C_i} \Lambda_e^{\mathfrak{G}_i^x}$, equipped with projections $\pi_e^i: \Lambda_{\mathfrak{G}_i^x} \rightarrow \Lambda_e^{\mathfrak{G}_i^x}$,
- $\Lambda_e^{\mathfrak{G}_i^x}$, for $e \in P_i \cup C_i$, is the cofree expansion of Λ_e^i along ξ_e^i , for which we denote the universal morphism from $_ \upharpoonright_{\xi_e^i}$ to Λ_e^i by $\varepsilon_e^i: \Lambda_e^{\mathfrak{G}_i^x} \upharpoonright_{\xi_e^i} \rightarrow \Lambda_e^i$.

According to the description of the ARNS defined by given points, we can restrict θ to a morphism of ARNS from \mathfrak{G}_1^x to \mathfrak{G}_2^x . Since \mathfrak{G}_1^x is ground, we further obtain, based on this restriction, a functor $F: \mathbb{J}_{\mathfrak{G}_1^x} \rightarrow \mathbb{J}_{\mathfrak{G}_2^x}$ that makes the following diagram commutative.

$$\begin{array}{ccc} \mathbb{J}_{\mathfrak{G}_1^x} & \xrightarrow{D_{\mathfrak{G}_1^x}} & \text{Sig}^{\text{ALTL}} \\ F \downarrow & & \nearrow D_{\mathfrak{G}_2^x} \\ \mathbb{J}_{\mathfrak{G}_2^x} & & \end{array}$$

This allows us to define the derived cocone $F \cdot \xi^2: D_{\mathfrak{G}_1^x} \Rightarrow A_{\mathfrak{G}_2^x}$, whose components are given, for example, by $(F \cdot \xi^2)_x = \xi_{\theta(x)}^2$. Since ξ^1 is the colimit of $D_{\mathfrak{G}_1^x}$ it follows that there exists a (unique) morphism of cocones $\sigma: \xi^1 \rightarrow F \cdot \xi^2$, that is an ALTL-signature morphism $\sigma: A_{\mathfrak{G}_1^x} \rightarrow A_{\mathfrak{G}_2^x}$ that satisfies, in particular, $\xi_e^1 \circ \sigma = \xi_{\theta(e)}^2$ for every hyperedge $e \in P_1 \cup C_1$.

We obtain in this way, for every hyperedge $e \in P_1 \cup C_1$, the composite morphism $\pi_{\theta(e)}^2 \upharpoonright_{\xi_{\theta(e)}^2} \circ \varepsilon_{\theta(e)}^2$ from $\Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\xi_{\theta(e)}^2} = \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma} \upharpoonright_{\xi_e^1}$ to $\Lambda_e^1 = \Lambda_{\theta(e)}^2$.

$$\begin{array}{ccccc} \Lambda_e^1 = \Lambda_{\theta(e)}^2 & \xleftarrow{\varepsilon_e^1} & \Lambda_e^{\mathfrak{G}_1^x} \upharpoonright_{\xi_e^1} & & \Lambda_e^{\mathfrak{G}_1^x} \\ \varepsilon_{\theta(e)}^2 \uparrow & & \uparrow h_e \upharpoonright_{\xi_e^1} & & \uparrow h_e \\ \Lambda_{\theta(e)}^{\mathfrak{G}_2^x} \upharpoonright_{\xi_{\theta(e)}^2} & \xleftarrow{\pi_{\theta(e)}^2 \upharpoonright_{\xi_{\theta(e)}^2}} & \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\xi_{\theta(e)}^2} = \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma} \upharpoonright_{\xi_e^1} & & \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma} \end{array}$$

Given that $\Lambda_e^{\mathfrak{G}_1^x}$ is the cofree expansion of Λ_e^1 along ξ_e^1 , we deduce that there

exists a (unique) morphism $h_e : \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma} \rightarrow \Lambda_e^{\mathfrak{G}_1^x}$ such that the above diagram is commutative. This implies, by the universal property of $\Lambda_{\mathfrak{G}_1^x}$, the existence of a (unique) morphism $h : \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma} \rightarrow \Lambda_{\mathfrak{G}_1^x}$ such that $h \circ \pi_e^1 = h_e$ for $e \in P_1 \cup C_1$.

$$\begin{array}{ccc}
 \Lambda_e^1 & \xleftarrow{\pi_e^1} & \Lambda_{\mathfrak{G}_1^x} \\
 & \swarrow h_e & \uparrow h \\
 & & \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma}
 \end{array}$$

It follows that the reduct $h \upharpoonright_{\xi_x^1}$ is a morphism from $\Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma} \upharpoonright_{\xi_x^1}$ to $\Lambda_{\mathfrak{G}_1^x} \upharpoonright_{\xi_x^1}$. Then, to complete the proof, we only need to notice that $\Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\sigma} \upharpoonright_{\xi_x^1} = \Lambda_{\mathfrak{G}_2^x} \upharpoonright_{\xi_{\theta(x)}^2} = \Lambda_{\theta(x)}^2$ and $\Lambda_{\mathfrak{G}_1^x} \upharpoonright_{\xi_x^1} = \Lambda_x^1$. \square

5.2 A LOGICAL VIEW ON SERVICE DISCOVERY AND BINDING

Building on the results of [Section 5.1](#), let us now investigate how the semantics of the service overlay can be characterized using fundamental computational aspects of logic-programming such as unification and resolution. Our approach is founded upon a simple and intuitive analogy between concepts of service-oriented computing like service module and client application [see e.g. [FLB11](#)], and concepts such as clause and query that are specific to the relational variant of logic programming. In order to clarify this analogy, we rely on the institutional framework that we put forward in [Chapter 3](#) to address the model-theoretic foundations of logic programming.

For the purpose of this section, it suffices to consider *definite clauses*, that is clauses of the form $\forall X \cdot C \leftarrow H$, where X is a signature of variables (of a given generalized substitution system), and C and H correspond to ‘atomic’ sentences and sets of ‘atomic’ sentences over X respectively. This enables us to make explicit – at the level of logic-programming frameworks – the layered construction of clauses and queries from the sentences of an arbitrary generalized substitution system (regarded as ‘atomic’) by adding the necessary implications and conjunctions similarly to the way in which we added quantifiers in [Section 3.2.1](#); this type of extension of a given generalized substitution system follows the well-known institution-independent semantics of Boolean connectives introduced in [[Tar86](#); see also [Bar74](#)]. On that account, we will rely on a slightly different (and simpler) logical foundation for conventional logic programming, based on the atomic fragment of first-order logic – whose generalized substitution system we denote by AFOL_{\neq}^1 – in-

stead of the quantifier-free fragment. Note that every single-sorted first-order signature $\langle F, P \rangle$ determines a substitution system

$$(\text{AFOL}_{\neq}^1)_{\langle F, P \rangle} : \text{Subst}_{\langle F, P \rangle} \rightarrow \underline{\text{AFOL}}_{\neq}^1(F, P) / \mathbb{R}\text{oom}^7$$

in which $\text{Subst}_{\langle F, P \rangle}$ is simply the category whose objects are sets of variables defined over $\langle F, P \rangle$, and whose arrows are first-order substitutions (see Section 3.1.1). The room $\underline{\text{AFOL}}_{\neq}^1(F, P)$ accounts for the (ground) atomic sentences given by $\langle F, P \rangle$, the models of $\langle F, P \rangle$, as well as the standard satisfaction relation between them. Finally, the functor $(\text{AFOL}_{\neq}^1)_{\langle F, P \rangle}$ maps every signature (i.e. every set) of variables X to the corridor $\langle \alpha_{\langle F, P \rangle, X}, \beta_{\langle F, P \rangle, X} \rangle$,

$$\begin{array}{ccc} & \alpha_{\langle F, P \rangle, X} & \\ \text{Sen}(F, P), \text{Mod}(F, P), \vDash_{\langle F, P \rangle} & \xrightarrow{\quad} & \text{Sen}(F \cup X, P), \text{Mod}(F \cup X, P), \vDash_{\langle F \cup X, P \rangle} \\ & \beta_{\langle F, P \rangle, X} & \end{array}$$

where $\alpha_{\langle F, P \rangle, X}$ and $\beta_{\langle F, P \rangle, X}$ are the translation of sentences and the reduction of models that correspond to the inclusion of signatures $\langle F, P \rangle \subseteq \langle F \cup X, P \rangle$.

5.2.1 A GENERALIZED SUBSTITUTION SYSTEM OF ORCHESTRATION SCHEMES

What is essential about orchestration schemes with respect to the development of the service-oriented variant of logic programming is that they can be organized as a category OS from which there exists a functor OrcScheme into SubstSys that allows us to capture some of the most basic aspects of service-oriented computing by means of logic-programming constructs. More precisely, orchestration schemes form the signatures of a generalized substitution system

$$\text{OrcScheme} : \text{OS} \rightarrow \text{SubstSys}$$

through which the notions of service module, application, discovery, and binding emerge as particular instances of the abstract notions of clause, query, unification, and resolution. In this sense, OrcScheme and AFOL_{\neq}^1 can be regarded as structures having the same role in the description of service-oriented and relational logic programming, respectively.

Morphisms of orchestration schemes are, intuitively, a way of encoding orchestrations. In order to understand how they arise in practice, let us

⁷ Through $\underline{\text{AFOL}}_{\neq}^1$ we refer to the institution that corresponds to the atomic fragment of the single-sorted variant of first-order logic without equality.

consider a morphism φ between two algebraic signatures Σ and Σ' used in defining program expressions. For instance, we may assume Σ to be the signature of structured programs discussed in [Example 5.1.2](#), and $\varphi: \Sigma \rightarrow \Sigma'$ its extension with a new operation symbol $\text{repeat_until_}: \text{PgmCond} \rightarrow \text{Pgm}$. Then, it is easy to notice that the φ -translation of Σ -terms (defined over a given set of program variables) generalizes to a functor F between the categories of program expressions defined over Σ and Σ' . Moreover, the choice of φ enables us to define a second functor U , from program expressions over Σ' to program expression over Σ , based on the derived signature morphism [see e.g. [ST11](#)] $\Sigma' \rightarrow \Sigma$ that encodes the repeat_until_ operation as the term $\underline{1} \ ; \ \text{while not } \underline{2} \ \text{do } \underline{1} \ \text{done}$.⁸

The functor U is clearly a right inverse of F with respect to ground program expressions, whereas in general, for every program expression pgm over Σ we actually obtain a morphism $\eta_{\text{pgm}}: \text{pgm} \rightarrow U(F(\text{pgm}))$ as a result of the potential renaming of program variables; thus, the morphism η_{pgm} accounts for the translation of the program variables of pgm along $F \ ; \ U$. Furthermore, for every program expression pgm' over Σ' , the translation of Σ -sentences determined by φ extends to a map between the specifications over $U(\text{pgm}')$ and the specifications over pgm' , which, as we will see, can be used to define a translation of the specifications over a program expression pgm (given by the signature Σ) to specifications over $F(\text{pgm})$. With respect to the semantics, it is natural to expect that every program expression pgm over Σ has the same behaviour as $F(\text{pgm})$ and, even more, that every program expression pgm' over Σ' (that may be built using repeat_until_), behaves in the same way as $U(\text{pgm}')$. These observations lead us to the following formalization of the notion of morphism of orchestration schemes.

DEFINITION 5.2.1 (Morphism of orchestration schemes). *A morphism between orchestration schemes $\langle \text{Orc}, \text{Spec}, \text{Grc}, \text{Prop} \rangle$ and $\langle \text{Orc}', \text{Spec}', \text{Grc}', \text{Prop}' \rangle$ is a tuple $\langle F, U, \eta, \sigma \rangle$, where*

$$\text{Orc} \begin{array}{c} \xrightarrow{F} \\ \xleftarrow{U} \end{array} \text{Orc}'$$

- F and U are functors as above such that $F(\text{Grc}) \subseteq \text{Grc}'$ and $U(\text{Grc}') \subseteq \text{Grc}$,
- η is a natural transformation $1_{\text{Orc}} \Rightarrow F \ ; \ U$ such that $\eta_{\mathfrak{g}} = 1_{\mathfrak{g}}$ for $\mathfrak{g} \in |\text{Grc}|$,
- σ is a natural transformation $U \ ; \ \text{Spec} \Rightarrow \text{Spec}'$ such that for every ground

⁸ In this context, $\underline{1}: \text{Pgm}$ and $\underline{2}: \text{Cond}$ are variables that correspond to the arguments of the derived operation $\underline{1} \ ; \ \text{while not } \underline{2} \ \text{do } \underline{1} \ \text{done}$.

orchestration $g' \in |\mathbf{Grc}'|$ and specification $\rho \in \text{Spec}(U(g'))$,

$$\sigma_{g'}(\rho) \in \text{Prop}'(g') \quad \text{if and only if} \quad \rho \in \text{Prop}(U(g')).$$

EXAMPLE 5.2.2. Let $\langle \Upsilon, \alpha, \beta \rangle$ be a morphism [see GB92; GR02] between two institutions $\mathcal{J}' = \langle \text{Sig}', \text{Sen}', \text{Mod}', \varepsilon' \rangle$ and $\mathcal{J} = \langle \text{Sig}, \text{Sen}, \text{Mod}, \varepsilon \rangle$ that are suitable for defining orchestration schemes of ARNS (according to the hypotheses introduced in Section 5.1.2). If (a) the signature functor $\Upsilon: \text{Sig}' \rightarrow \text{Sig}$ is cocontinuous, (b) the natural transformation $\beta: \text{Mod}' \Rightarrow \Upsilon^{\text{op}} \circ \text{Mod}$ preserves cofree expansions and products, and, moreover, if (c) both Υ and β admit sections, in the sense that there exist a functor $\Phi: \text{Sig} \rightarrow \text{Sig}'$ such that $\Phi \circ \Upsilon = 1_{\text{Sig}}$ and a natural transformation $\tau: \text{Mod} \Rightarrow \Phi^{\text{op}} \circ \text{Mod}'$ such that $\tau \circ (\Phi^{\text{op}} \cdot \beta) = 1_{\text{Mod}}$, then $\langle \Upsilon, \alpha, \beta \rangle$ gives rise to a morphism $\langle F, U, \eta, \sigma \rangle$ between the orchestration schemes of ARNS defined over \mathcal{J} and \mathcal{J}' .

In particular, the functor F maps the diagram and the models that label an ARN defined over \mathcal{J} to their images under Φ and τ ; similarly, U maps ARNS defined over \mathcal{J}' according to Υ and β ; the natural transformation η is just an identity, and σ extends the α -translation of sentences to specifications. The additional properties of Υ and β are essential for ensuring that the observable behaviour of ground networks is preserved.

One may consider, for instance, the extension of ALTL (in the role of \mathcal{J}) with new temporal modalities such as *previous* and *since*, as in [Kna+10]; this naturally leads to a morphism of orchestration schemes for which both Υ and β would be identities. Alternatively, one may explore the correspondence between deterministic weak ω -automata – which form a subclass of Muller automata – and sets of traces that are both Büchi and co-Büchi deterministically recognizable – for which a minimal automaton can be shown to exist [see e.g. MS97; Lödo1]. In this case, in the roles of \mathcal{J} and \mathcal{J}' we could consider variants of ALTL with models given by sets of traces and deterministic weak automata, respectively;⁹ Υ and α would be identities, β would define the language recognized by a given automaton, and τ would capture the construction of minimal automata.

It is easy to see that the morphisms of orchestration schemes compose in a natural way in terms of their components, thus giving rise to a category of orchestration schemes.

⁹ Note that, to ensure that model reducts are well defined for deterministic automata, one may need to restrict signature morphisms to injective maps.

PROPOSITION 5.2.3. *Orchestration-scheme morphisms can be composed as follows:*

$$\langle F, U, \eta, \sigma \rangle \circ \langle F', U', \eta', \sigma' \rangle = \langle F \circ F', U' \circ U, \eta \circ (F \cdot \eta' \cdot U), (U' \cdot \sigma) \circ \sigma' \rangle.$$

Under this composition, morphisms of orchestration schemes form a category OS.

□

The definition of the functor `OrcScheme` is grounded on two simple ideas:

- 1 Orchestration schemes can be regarded as signatures of variables; they provide sentences in the form of specifications, and models as morphisms into ground orchestrations – which can also be seen, in the case of `ARNs`, for example, as collections of ground networks assigned to the ‘variables’ of the considered orchestration. In addition, we can define a satisfaction relation between the models and the sentences of an orchestration based on the evaluation of specifications with respect to ground orchestrations. In this way, every orchestration scheme yields an institution whose composition resembles that of the so-called *institutions of extended models* [see [SMLo4](#)].
- 2 There is a one-to-one correspondence between institutions and substitution systems defined over the initial room $\langle \emptyset, \mathbb{1}, \emptyset \rangle$ – the room given by the empty set of sentences, the terminal category $\mathbb{1}$, and the empty satisfaction relation. The effect of this is that a clause can be described as ‘correct’ whenever it is satisfied by the sole model of $\langle \emptyset, \mathbb{1}, \emptyset \rangle$; therefore, we obtain precisely the notion of correctness of a service module [[FLB11](#)]: all models of the underlying signature of variables, that is of the orchestration, that satisfy the antecedent of the clause satisfy its consequent as well.

Formally, the generalized substitution system `OrcScheme` results from the composition of two functors, `Ins`: `OS` \rightarrow `colIns` and `SS`: `colIns` \rightarrow `SubstSys`, that implement the general constructions outlined above.

$$\begin{array}{ccc} & \text{OrcScheme} & \\ & \curvearrowright & \\ \text{OS} & \xrightarrow{\text{Ins}} & \text{colIns} \xrightarrow{\text{SS}} \text{SubstSys} \end{array}$$

The functor `Ins` carries most of the complexity of `OrcScheme`, and is discussed in detail in [Theorem 5.2.4](#). Concerning `SS`, we recall from [Section 2.3](#) that the category `colIns` of institution comorphisms can also be described as the category $[_ \rightarrow \mathbb{R}\text{oom}]^\sharp$ of functors into `Room`, and that any functor $G: \mathbb{K} \rightarrow \mathbb{K}'$ can be extended to a functor $[_ \rightarrow \mathbb{K}]^\sharp \rightarrow [_ \rightarrow \mathbb{K}']^\sharp$ that is given essentially by the right-composition with G . In particular, the isomorphism `Room` \rightarrow $\langle \emptyset, \mathbb{1}, \emptyset \rangle / \mathbb{R}\text{oom}$ that maps every room $\langle S, \mathbb{M}, \varepsilon \rangle$ to the unique cor-

ridor $\langle \emptyset, \mathbb{1}, \emptyset \rangle \rightarrow \langle S, \mathbb{M}, \varepsilon \rangle$ generates an isomorphism of categories between $[_ \rightarrow \mathbb{R}oom]^\sharp$, that is coIns , and $[_ \rightarrow \langle \emptyset, \mathbb{1}, \emptyset \rangle / \mathbb{R}oom]^\sharp$. The latter is further embedded into SubstSys , defining in this way, by composition, the required functor SS . To sum up, SS maps every institution $\mathcal{J}: \text{Sig} \rightarrow \mathbb{R}oom$ to the substitution system $\mathcal{S}: \text{Sig} \rightarrow \langle \emptyset, \mathbb{1}, \emptyset \rangle / \mathbb{R}oom$ for which $\mathcal{S}(\Sigma)$, for every signature $\Sigma \in |\text{Sig}|$, is the unique corridor between $\langle \emptyset, \mathbb{1}, \emptyset \rangle$ and $\mathcal{J}(\Sigma)$.

THEOREM 5.2.4. *The map $\text{Ins}: \text{OS} \rightarrow \text{coIns}$ is a functor, where*

- for any orchestration scheme $\mathcal{O} = \langle \text{Orc}, \text{Spec}, \text{Grc}, \text{Prop} \rangle$, $\text{Ins}(\mathcal{O})$ is the institution $\langle \text{Orc}, \text{Spec}, _ / \text{Grc}, \varepsilon \rangle$ whose family of satisfaction relations is given by

$$(\delta: \mathfrak{o} \rightarrow \mathfrak{g}) \varepsilon_{\mathfrak{o}} SP \quad \text{if and only if} \quad \text{Spec}(\delta)(SP) \in \text{Prop}(\mathfrak{g})$$

for every orchestration \mathfrak{o} , every \mathfrak{o} -model δ (i.e. every morphism of orchestrations $\delta: \mathfrak{o} \rightarrow \mathfrak{g}$ such that \mathfrak{g} is ground), and every specification SP over \mathfrak{o} ;¹⁰

- for any morphism of orchestration schemes $\langle F, U, \eta, \sigma \rangle: \mathcal{O} \rightarrow \mathcal{O}'$, with \mathcal{O} as above, and \mathcal{O}' given by $\langle \text{Orc}', \text{Spec}', \text{Grc}', \text{Prop}' \rangle$, $\text{Ins}(F, U, \eta, \sigma)$ is the comorphism of institutions $\langle F, \alpha, \beta \rangle: \text{Ins}(\mathcal{O}) \rightarrow \text{Ins}(\mathcal{O}')$ defined by

$$\alpha_{\mathfrak{o}} = \text{Spec}(\eta_{\mathfrak{o}}) \circ \sigma_{F(\mathfrak{o})} \quad \text{and} \quad \beta_{\mathfrak{o}} = v_{F(\mathfrak{o})} \circ (\eta_{\mathfrak{o}} / \text{Grc})$$

for every orchestration $\mathfrak{o} \in |\text{Orc}|$, where $v: (_ / \text{Grc}') \Rightarrow U^{\text{op}} \circ (_ / \text{Grc})$ is the natural transformation given by $v_{\mathfrak{o}'}(x) = U(x)$ for every orchestration $\mathfrak{o}' \in |\text{Orc}'|$ and every object or arrow x of the comma category $\mathfrak{o}' / \text{Grc}'$.

PROOF. For the first part, all we need to show is that the satisfaction condition holds; but this follows easily since for every morphism of orchestrations $\theta: \mathfrak{o}_1 \rightarrow \mathfrak{o}_2$, every \mathfrak{o}_1 -specification SP , and every \mathfrak{o}_2 -model $\delta: \mathfrak{o}_2 \rightarrow \mathfrak{g}$,

$$\begin{aligned} \delta \varepsilon_{\mathfrak{o}_2} \text{Spec}(\theta)(SP) & \quad \text{if and only if} \quad \text{Spec}(\theta \circ \delta)(SP) \in \text{Prop}(\mathfrak{g}) \\ & \quad \text{if and only if} \quad (\theta / \text{Grc})(\delta) = \theta \circ \delta \varepsilon_{\mathfrak{o}_2} SP. \end{aligned}$$

As regards the second part of the statement, notice that α and β are the natural transformations $(\eta \cdot \text{Spec}) \circ (F \cdot \sigma)$ and $(\eta^{\text{op}} \cdot (_ / \text{Grc})) \circ (F^{\text{op}} \cdot v)$, respectively. In order to verify that $\langle F, \alpha, \beta \rangle$ is indeed a comorphism between $\text{Ins}(\mathcal{O})$ and $\text{Ins}(\mathcal{O}')$, consider an orchestration \mathfrak{o} in Orc , a model $\delta': F(\mathfrak{o}) \rightarrow \mathfrak{g}'$ of $F(\mathfrak{o})$, and a specification SP over \mathfrak{o} . Assuming that ε' is the family of

¹⁰ Moreover, the institution $\text{Ins}(\mathcal{O})$ is exact, because the functor $_ / \text{Grc}: \text{Orc}^{\text{op}} \rightarrow \text{Cat}$ is continuous (see e.g. [Mes89]).

satisfaction relations of $\text{Ins}(\mathcal{O}')$, we deduce that

$$\begin{aligned}
& \delta' \vDash'_{F(\mathfrak{o})} \alpha_{\mathfrak{o}}(SP) \\
& \text{iff } \text{Spec}'(\delta')(\alpha_{\mathfrak{o}}(SP)) \in \text{Prop}'(\mathfrak{g}') && \text{by the definition of } \vDash'_{F(\mathfrak{o})} \\
& \text{iff } \text{Spec}'(\delta')(\sigma_{F(\mathfrak{o})}(\text{Spec}(\eta_{\mathfrak{o}})(SP))) \in \text{Prop}'(\mathfrak{g}') && \text{by the definition of } \alpha_{\mathfrak{o}} \\
& \text{iff } \sigma_{\mathfrak{g}'}(\text{Spec}(\eta_{\mathfrak{o}} \circledast U(\delta'))(SP)) \in \text{Prop}'(\mathfrak{g}') && \text{by the naturality of } \sigma \\
& \text{iff } \text{Spec}(\eta_{\mathfrak{o}} \circledast U(\delta'))(SP) \in \text{Prop}(U(\mathfrak{g}')) && \text{since } \text{Prop}(U(\mathfrak{g}')) \\
& && \qquad \qquad \qquad = \sigma_{\mathfrak{g}'}^{-1}(\text{Prop}'(\mathfrak{g}')) \\
& \text{iff } \eta_{\mathfrak{o}} \circledast U(\delta') \vDash_{\mathfrak{o}} SP && \text{by the definition of } \vDash_{\mathfrak{o}} \\
& \text{iff } \beta_{\mathfrak{o}}(\delta') \vDash_{\mathfrak{o}} SP. && \text{by the definition of } \beta_{\mathfrak{o}}
\end{aligned}$$

Finally, it is easy to see that Ins preserves identities. To prove that it also preserves composition, let $\langle F, U, \eta, \sigma \rangle$ and $\langle F', U', \eta', \sigma' \rangle$ be morphisms of orchestration schemes as below, and suppose that $\text{Ins}(F, U, \eta, \sigma) = \langle F, \alpha, \beta \rangle$ and $\text{Ins}(F', U', \eta', \sigma') = \langle F', \alpha', \beta' \rangle$.

$$\begin{array}{ccc}
& \langle F, U, \eta, \sigma \rangle & \xrightarrow{\quad} & \langle \text{Orc}', \text{Spec}', \text{Grc}', \text{Prop}' \rangle & \xrightarrow{\quad} & \langle F', U', \eta', \sigma' \rangle \\
& \searrow & & \downarrow & & \downarrow \\
& \langle \text{Orc}, \text{Spec}, \text{Grc}, \text{Prop} \rangle & & & & \langle \text{Orc}'', \text{Spec}'', \text{Grc}'', \text{Prop}'' \rangle \\
& \swarrow & & \downarrow & & \downarrow \\
& & & \langle F \circledast F', U' \circledast U, \eta \circledast (F \cdot \eta' \cdot U), (U' \cdot \sigma) \circledast \sigma' \rangle & &
\end{array}$$

In addition, let $v: (_ / \text{Grc}') \Rightarrow U^{\text{op}} \circledast (_ / \text{Grc})$ and $v': (_ / \text{Grc}'') \Rightarrow U'^{\text{op}} \circledast (_ / \text{Grc}')$ be the natural transformations involved in the definitions of β and β' , respectively. Based on the composition of morphisms of orchestration schemes and on the definition of Ins , it follows that $\text{Ins}(\langle F, U, \eta, \sigma \rangle \circledast \langle F', U', \eta', \sigma' \rangle)$ is a comorphism of institutions of the form $\langle F \circledast F', \alpha'', \beta'' \rangle$, where α'' and β'' are given by

$$\begin{aligned}
\alpha''_{\mathfrak{o}} &= \text{Spec}((\eta \circledast (F \cdot \eta' \cdot U))_{\mathfrak{o}}) \circledast ((U' \cdot \sigma) \circledast \sigma')_{(F \circledast F')(\mathfrak{o})} \\
\beta''_{\mathfrak{o}} &= (v' \circledast (U'^{\text{op}} \cdot v))_{(F \circledast F')(\mathfrak{o})} \circledast ((\eta \circledast (F \cdot \eta' \cdot U))_{\mathfrak{o}} / \text{Grc}).
\end{aligned}$$

In order to complete the proof, we need to show that $\alpha'' = \alpha \circledast (F \cdot \alpha')$ and $\beta'' = (F \cdot \beta') \circledast \beta$. Each of these equalities follows from a sequence of straightforward calculations that relies on the naturality of σ (in the case of α''), or on the naturality of v (in the case of β'').

$$\begin{aligned}
\alpha''_{\mathfrak{o}} &= \text{Spec}(\eta_{\mathfrak{o}}) \circledast \underbrace{\text{Spec}(U(\eta'_{F(\mathfrak{o})}))}_{\sigma_{(F \circledast F')(\mathfrak{o})}} \circledast \sigma'_{(F \circledast F')(\mathfrak{o})} \\
&= \text{Spec}(\eta_{\mathfrak{o}}) \circledast \sigma_{F(\mathfrak{o})} \circledast \text{Spec}'(\eta'_{F(\mathfrak{o})}) \circledast \sigma'_{(F \circledast F')(\mathfrak{o})} = \alpha_{\mathfrak{o}} \circledast \alpha'_{F(\mathfrak{o})}
\end{aligned}$$

$$\begin{aligned}
\beta''_{\mathfrak{o}} &= v'_{(F \circledast F')(\mathfrak{o})} \circledast v_{(F \circledast F' \circledast U')(\mathfrak{o})} \circledast \underbrace{(U(\eta'_{F(\mathfrak{o})}) / \mathbf{Grc})}_{\text{}} \circledast (\eta_{\mathfrak{o}} / \mathbf{Grc}) \\
&= v'_{(F \circledast F')(\mathfrak{o})} \circledast (\eta'_{F(\mathfrak{o})} / \mathbf{Grc}') \circledast v_{F(\mathfrak{o})} \circledast (\eta_{\mathfrak{o}} / \mathbf{Grc}) = \beta'_{F(\mathfrak{o})} \circledast \beta_{\mathfrak{o}} \quad \square
\end{aligned}$$

COROLLARY 5.2.5. *Together with the category of orchestration schemes, the functor $\text{OrcScheme}: \text{OS} \rightarrow \text{SubstSys}$ defines a generalized substitution system.* \square

We recall from [Section 3.3](#) that, in order to be used as semantic frameworks for logic programming, generalized substitution systems need to ensure a weak model-amalgamation property between the models that are ground and those that are defined by signatures of variables. This property entails that the satisfaction of quantified sentences (and, in particular, of clauses and queries) is invariant under change of notation. In the case of OrcScheme , this means that the correctness property of service modules does not depend on the actual orchestration scheme over which the modules are defined.

PROPOSITION 5.2.6. *The generalized substitution system $\langle \text{OS}, \text{OrcScheme} \rangle$ has weak model amalgamation.*

PROOF. Let φ be a morphism $\langle F, U, \eta, \sigma \rangle$ between orchestration schemes \mathcal{O} and \mathcal{O}' as in [Definition 5.2.1](#), and let \mathfrak{o} be an orchestration of \mathcal{O} . Since orchestrations define substitution systems over the initial room $\langle \emptyset, \mathbb{1}, \emptyset \rangle$, we can redraw the diagram of interest as follows:

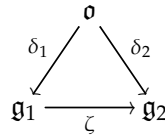
$$\begin{array}{ccc}
|\mathbb{1}| & \xleftarrow{-\uparrow\varphi} & |\mathbb{1}| \\
-\uparrow\mathfrak{o} \uparrow & & \uparrow -\uparrow\mathfrak{o}' \\
|\mathfrak{o} / \mathbf{Grc}| & \xleftarrow{\beta_{\varphi, \mathfrak{o}}} & |F(\mathfrak{o}) / \mathbf{Grc}'|
\end{array}$$

It is easy to see that the above diagram depicts a weak pullback if and only if $\beta_{\varphi, \mathfrak{o}}$ is surjective on objects. By [Theorem 5.2.4](#), we know that $\beta_{\varphi, \mathfrak{o}}(\delta') = \eta_{\mathfrak{o}} \circledast U(\delta')$ for every object $\delta': F(\mathfrak{o}) \rightarrow \mathfrak{g}'$ of the comma category $F(\mathfrak{o}) / \mathbf{Grc}'$. Therefore, for every $\delta: \mathfrak{o} \rightarrow \mathfrak{g}$ in $|\mathfrak{o} / \mathbf{Grc}|$ we obtain

$$\begin{aligned}
\beta_{\varphi, \mathfrak{o}}(F(\delta)) &= \eta_{\mathfrak{o}} \circledast U(F(\delta)) && \text{by Theorem 5.2.4} \\
&= \delta \circledast \eta_{\mathfrak{g}} && \text{by the naturality of } \eta \\
&= \delta. && \text{because, by definition, } \eta_{\mathfrak{g}} \text{ is an identity} \quad \square
\end{aligned}$$

REMARK 5.2.7. In addition to model amalgamation, it is important to notice that, similarly to AFOL_{\neq}^1 , in OrcScheme the satisfaction of sentences is preserved by model homomorphisms. This is an immediate consequence of the

fact that, in every orchestration scheme, the morphisms of ground orchestrations preserve properties: given an orchestration σ , a specification SP over σ , and a homomorphism ζ between σ -models δ_1 and δ_2 as depicted below, if $\text{Spec}(\delta_1)(SP)$ is a property of \mathfrak{g}_1 then $\text{Spec}(\delta_2)(SP) = \text{Spec}(\zeta)(\text{Spec}(\delta_1)(SP))$ is a property of \mathfrak{g}_2 ; therefore, $\delta_1 \models^{\text{OrcScheme}} SP$ implies $\delta_2 \models^{\text{OrcScheme}} SP$.



5.2.2 THE CLAUSAL STRUCTURE OF SERVICES

Given the above constructions, we can now consider a service-oriented notion of clause, defined over the generalized substitution system OrcScheme rather than AFOL_{\neq}^1 . Intuitively, this means that we replace first-order signatures with orchestration schemes, sets of variables with orchestrations, and first-order sentences (over given sets of variables) with specifications. Furthermore, certain orchestration schemes allow us to identify structures that correspond to finer-grained notions like variable and term: in the case of program expressions, variables and terms have their usual meaning (although we only take into account executable expressions), whereas in the case of ARNS , variables and terms materialize as requires-points and sub- ARNS defined by provides-points.

The following notion of service clause corresponds to the concepts of service module and of orchestrated interface discussed in [FLB11] and [FL13b].

DEFINITION 5.2.8 (Service clause). A (definite) service-oriented clause over an orchestration scheme \mathcal{O} is a structure $\forall \sigma \cdot P \leftarrow R$, also denoted

$$P \leftarrow_{\sigma} R$$

where σ is an orchestration of \mathcal{O} , P is a specification over σ – called the *provides-interface* of the clause – and R is a finite set of specifications over σ – the *requires-interface* of the clause.

The semantics of service-oriented clauses is defined just as the semantics of first-order clauses, except that they are evaluated within the generalized substitution system OrcScheme instead of AFOL_{\neq}^1 . As mentioned before, this means that we can only distinguish whether or not a clause is correct.

DEFINITION 5.2.9 (Correct clause). A service-oriented clause $\forall \sigma \cdot P \leftarrow R$ is said to be *correct* if for every morphism of orchestrations $\delta: \sigma \rightarrow \mathfrak{g}$ such that

g is a ground orchestration and $\text{Spec}(\delta)(R)$ consists only of properties of g , the specification $\text{Spec}(\delta)(P)$ is also a property of g .

In other words, a service-oriented clause is correct if its provides-interface is ensured by its orchestration and the specifications of its requires-interface.

EXAMPLE 5.2.10. We have already encountered several instances of service clauses in the form of the program modules depicted in Figure 5.1. Their provides-interfaces and requires-interfaces are placed on the left-hand and right-hand side of their orchestrations, and are represented using symbolic forms that are traditionally associated with services.

To illustrate how service modules can be defined as clauses over ARNS, notice that the network JourneyPlanner introduced in Example 5.1.17 can orchestrate a module named Journey Planner that consistently delivers the requested directions, provided that the routes and the timetables can be obtained whenever they are needed. This can be described in logical terms through the following (correct) service-oriented clause:

$$@_{JP_1} \rho^{JP} \xleftarrow{\text{JourneyPlanner}} \{ @_{R_1} \rho_1^{JP}, @_{R_2} \rho_2^{JP} \}$$

where ρ^{JP} , ρ_1^{JP} , and ρ_2^{JP} are the ALTL-sentences $\Box(\text{planJourney}_i \Rightarrow \Diamond \text{directions!})$, $\Box(\text{getRoutes}_i \Rightarrow \Diamond \text{routes!})$, and $\Box(\text{routes}_i \Rightarrow \Diamond \text{timetables!})$, respectively.

Client applications are captured in the present setting by service-oriented queries. They are defined similarly to service clauses, but their semantics is based on an existential quantification, not on a universal one.

DEFINITION 5.2.11 (Service query). A *service-oriented query* over an orchestration scheme \mathcal{O} is a structure $\exists \mathfrak{o} \cdot Q$, also written

$$\frac{}{\mathfrak{o}} Q$$

such that \mathfrak{o} is an orchestration of \mathcal{O} , and Q is a finite set of specifications over \mathfrak{o} that defines the *requires-interface* of the query.

DEFINITION 5.2.12 (Satisfiable query). A service-oriented query $\exists \mathfrak{o} \cdot Q$ is *satisfiable* if there exists a morphism of orchestrations $\delta: \mathfrak{o} \rightarrow g$ such that g is ground and all specifications in $\text{Spec}(\delta)(Q)$ are properties of g .

EXAMPLE 5.2.13. In Figure 5.9, we outline the ARN of a possible client application for the service module Journey Planner discussed in Example 5.2.10. We specify the actual application, called Traveller, through the service query

$$\frac{}{\text{Traveller}} \{ @_{R_1} \rho_1^T \}$$

given by the ALTL-sentence $\Box(\text{getRoute}_j \Rightarrow \Diamond \text{route!})$.

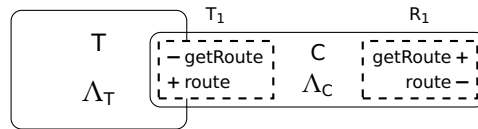


FIGURE 5.9. The ARN Traveller

5.2.3 RESOLUTION AS SERVICE DISCOVERY AND BINDING

Let us now turn our attention to the dynamic aspects of service-oriented computing that result from the process of service discovery and binding [FLB11]. *Service discovery* represents, as in conventional logic programming, the search for a module (service clause) that can be bound to a given application (service query) in order to take it one step closer to a possible solution, that is to a ‘complete’ application capable of fulfilling its goal.

From a technical point of view, both discovery and binding are subject to *matching* the requires-interface of the application, or more precisely, one of its specifications, with the provides-interface of the module under consideration. This is usually achieved through a suitable notion of *refinement* of specifications. For instance, in the case of program expressions, given specifications $\iota_1: [\rho_1, \rho'_1]$ and $\iota_2: [\rho_2, \rho'_2]$ over programs $\text{pgm}_1: eXp_1$ and $\text{pgm}_2: eXp_2$, respectively, $\iota_2: [\rho_2, \rho'_2]$ refines $\iota_1: [\rho_1, \rho'_1]$ up to a cospan

$$\text{pgm}_1: eXp_1 \xrightarrow{\langle \psi_1, \pi_1 \rangle} \text{pgm}: eXp \xleftarrow{\langle \psi_2, \pi_2 \rangle} \text{pgm}_2: eXp_2$$

if by translation we obtain specifications that refer to the same position of $\text{pgm}: eXp$, that is $\pi_1 \cdot \iota_1 = \pi_2 \cdot \iota_2$, such that the pre-condition $\psi_2(\rho_2)$ is weaker than $\psi_1(\rho_1)$, and the post-condition $\psi_2(\rho'_2)$ is stronger than $\psi_1(\rho'_1)$:

$$\psi_1(\rho_1) \models^{\text{POA}} \psi_2(\rho_2) \quad \text{and} \quad \psi_2(\rho'_2) \models^{\text{POA}} \psi_1(\rho'_1).$$

This reflects the consequence rules introduced in [Hoa69] (see also [Mor94], whence we adopt the notation $\iota_1: [\rho_1, \rho'_1] \sqsubseteq \iota_2: [\rho_2, \rho'_2]$ used in Figure 5.2).

In a similar manner, in the case of ARNS, a specification $@_{x_1} \rho_1$ over a network \mathfrak{N}_1 is refined by another specification $@_{x_2} \rho_2$ over a network \mathfrak{N}_2 up to a cospan of morphisms of ARNS $\langle \theta_1: \mathfrak{N}_1 \rightarrow \mathfrak{N}, \theta_2: \mathfrak{N}_2 \rightarrow \mathfrak{N} \rangle$ when $\theta_1(x_1) = \theta_2(x_2)$ and $\theta_{2,x_2}^{\text{pt}}(\rho_2) \models^{\text{ALTL}} \theta_{1,x_1}^{\text{pt}}(\rho_1)$ [see TF13]. Both of these notions of refinement generalize to the following concept of unification.

DEFINITION 5.2.14 (Service-oriented unification). Let SP_1 and SP_2 be specifications defined over orchestrations σ_1 and σ_2 , respectively, of an arbitrary but fixed orchestration scheme. We say that the ordered pair $\langle SP_1, SP_2 \rangle$ is *unifiable* if there exists a cospan of morphisms of orchestrations as below,

$$\sigma_1 \xrightarrow{\theta_1} \sigma \xleftarrow{\theta_2} \sigma_2$$

called the *unifier* of SP_1 and SP_2 , such that $\theta_2(SP_2) \vDash^{\text{OrcScheme}} \theta_1(SP_1)$.

Therefore, $\langle \theta_1, \theta_2 \rangle$ is a unifier of SP_1 and SP_2 if and only if, for every morphism of orchestrations $\delta: \sigma \rightarrow g$ such that g is a ground orchestration, if $\text{Spec}(\theta_2 \circ \delta)(SP_2)$ is a property of g then so is $\text{Spec}(\theta_1 \circ \delta)(SP_1)$.

In conventional logic programming, the resolution inference rule simplifies the current goal and at the same time, through unification, yields computed substitutions that could eventually deliver a solution to the initial query. This process is accurately reflected in the case of service-oriented computing by *service binding*. Unlike relational logic programming, however, the emphasis is put not on the computed morphisms of orchestrations (i.e. on substitutions), but on the dynamic reconfiguration of the orchestrations (i.e. of the signatures of variables) that underlie the considered applications.

DEFINITION 5.2.15 (Service-oriented resolution). Let $\exists \sigma_1 \cdot Q_1$ be a query and $\forall \sigma_2 \cdot P_2 \leftarrow R_2$ a clause defined over an arbitrary but fixed orchestration scheme. A query $\exists \sigma \cdot Q$ is said to be *derived by resolution* from $\exists \sigma_1 \cdot Q_1$ and $\forall \sigma_2 \cdot P_2 \leftarrow R_2$ using the *computed morphism* $\theta_1: \sigma_1 \rightarrow \sigma$ when

$$\frac{\frac{\vdash_{\sigma_1} Q_1 \quad P_2 \leftarrow_{\sigma_2} R_2}{\vdash_{\sigma} \theta_1(Q_1 \setminus \{SP_1\}) \cup \theta_2(R_2)} \theta_1}{\vdash_{\sigma} \theta_1(Q_1 \setminus \{SP_1\}) \cup \theta_2(R_2)} \theta_1$$

- θ_1 can be extended to a unifier $\langle \theta_1, \theta_2 \rangle$ of a specification $SP_1 \in Q_1$ and P_2 ,
- Q is the set of specifications given by the translation along θ_1 and θ_2 of the specifications in $Q_1 \setminus \{SP_1\}$ and R_2 .

EXAMPLE 5.2.16. Consider the service-oriented query and the service clause detailed in [Examples 5.2.10](#) and [5.2.13](#). One can easily see that the single specification $@_{R_1} \rho_1^T$ of the requires-interface of the application Traveller and the provides-interface $@_{P_1} \rho_1^P$ of the module Journey Planner form a unifiable pair: they admit, for instance, the unifier $\langle \theta_1, \theta_2 \rangle$ given by

$$\text{Traveller} \xrightarrow{\theta_1} \text{JourneyPlannerApp} \xleftarrow{\theta_2} \text{JourneyPlanner}$$

- the ARN JourneyPlannerApp depicted in [Figure 5.10](#),
- the morphism θ_1 that maps the point R_1 to JP_1 , the communication hyperedge C to CJP and the messages $getRoute$ and $route$ of M_{R_1} to $planJourney$ and $directions$, while preserving all the remaining elements of $Traveller$, and
- the inclusion θ_2 of $JourneyPlanner$ into $JourneyPlannerApp$.

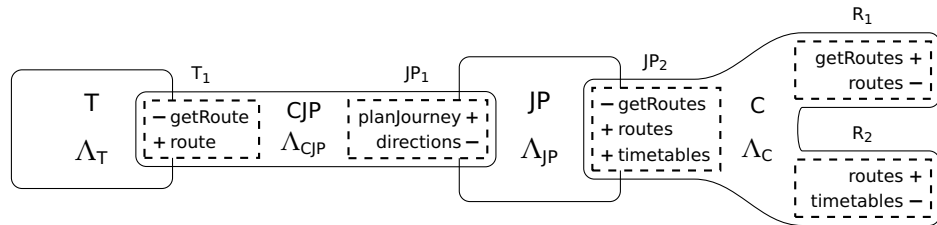


FIGURE 5.10. The ARN JourneyPlannerApp

It follows that we can derive by resolution a new service query given by the ARN JourneyPlannerApp and the requires-specifications $@_{R_1} \rho_1^{JP}$ and $@_{R_2} \rho_2^{JP}$.

$$\frac{\frac{\frac{\text{Traveller} \vdash \{ @_{R_1} \rho_1^T \}}{\text{Traveller}} \quad \frac{\text{JourneyPlanner} \vdash \{ @_{R_1} \rho_1^{JP}, @_{R_2} \rho_2^{JP} \}}{\text{JourneyPlanner}}}{\text{JourneyPlannerApp} \vdash \{ @_{R_1} \rho_1^{JP}, @_{R_2} \rho_2^{JP} \}}}{\theta_1}$$

THE LOGIC-PROGRAMMING FRAMEWORK OF SERVICES. The crucial property of the above notions of service-oriented clause, query, and resolution is that, together with the generalized substitution system $OrcScheme$ used to define them, they give rise to a logic-programming framework. The construction is to a great extent self-evident, and it requires little additional consideration apart from the fact that, from a technical point of view, in order to define clauses and queries as quantified sentences, we need to extend $OrcScheme$ by closing the sets of sentences that it defines under propositional connectives such as implication and conjunction. It should be noted, however, that the properties that guarantee the well-definedness of the resulting logic-programming framework such as the fact that its underlying generalized substitution system has weak model amalgamation (ensured by [Proposition 5.2.6](#)), and also the fact that the satisfaction of specifications is preserved by model homomorphisms (detailed in [Remark 5.2.7](#)), are far from trivial, especially when taking into account particular orchestration schemes (see e.g. [Proposition 5.1.33](#)).

By describing service discovery and binding as instances of unification and resolution (specific to the logic-programming framework of services)

we obtain not only a rigorously defined analogy between service-oriented computing and relational logic programming, but also a way to apply the general theory of logic programming developed in [Section 3.3](#) to the particular case of services. For example, we gain a concept of solution to a service query that reflects the rather intuitive service-oriented notion of solution and, moreover, through Herbrand's theorem, a characterization of satisfiable queries as queries that admit solutions.

DEFINITION 5.2.17 (Solution). A *solution* to a service-oriented query $\exists \sigma \cdot Q$ consists of a morphism of orchestrations $\psi : \sigma \rightarrow \sigma'$ such that σ' has models, and every one of them satisfies the ψ -translations of the specifications in Q .

PROPOSITION 5.2.18. A service query is satisfiable if and only if it has a solution. \square

Even more significant is the fact that logic programming provides us with a general search procedure that can be used to compute solutions to queries. The search is triggered by a query $\exists \sigma \cdot Q$ and consists in the iterated application of resolution, that is of service discovery and binding, until the requires-interface of the derived service query consists solely of trivial specifications (tautologies); these are specifications whose translation along morphisms into ground orchestrations always gives rise to properties. Thus, whenever the search procedure successfully terminates we obtain a *computed answer* to the original query by sequentially composing the resulting computed morphisms. This is the process that led, for example, to the derivation of the program that calculates the quotient and the remainder obtained on dividing two natural numbers illustrated in [Figure 5.2](#). The computed answer is given in this case by the sequence of substitutions

$$\begin{aligned} \text{pgm} &\mapsto \text{pgm}_1 \ ; \ \text{pgm}_2 \mapsto (\text{pgm}_3 \ ; \ \text{pgm}_4) \ ; \ \text{pgm}_2 \mapsto \dots \\ &\mapsto (q := 0 \ ; \ r := x) \ ; \ \text{while } y \leq r \ \text{do} \\ &\quad q := q + 1 \ ; \ r := r - y \\ &\quad \text{done.} \end{aligned}$$

In a similar manner, we can continue [Example 5.2.16](#) towards the derivation of an answer to the Traveller application. To this purpose, we assume that Map Services and Transport System are two additional service modules that correspond to the processes MS and TS used in [Example 5.1.25](#), and whose provides-interfaces meet the requires-specifications of the module Journey Planner. We obtain in this way the construction outlined in [Figure 5.11](#).

The soundness of resolution, detailed in [Proposition 5.2.19](#) below, entails

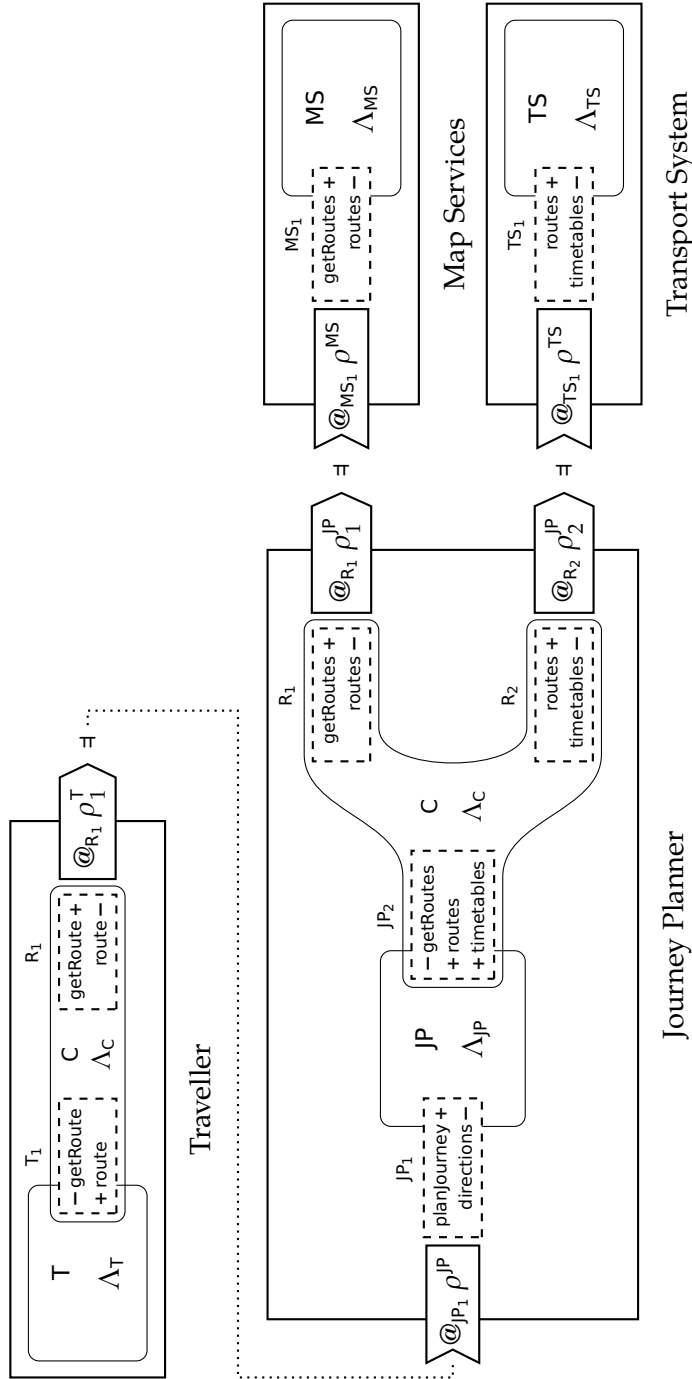


FIGURE 5.11. The derivation of an answer to the Traveller application

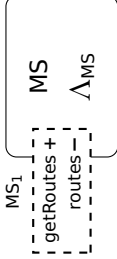
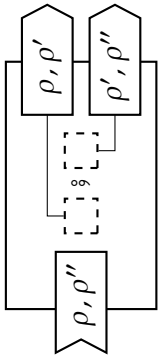
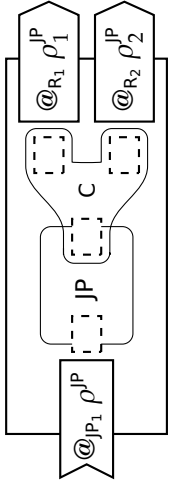
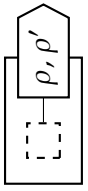
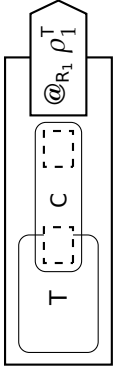
$$\begin{aligned}
 \rho_1^T &: \square(\text{getRoute}_j \Rightarrow \diamond \text{route}_j!) & \rho_2^P &: \square(\text{routes}_j \Rightarrow \diamond \text{timetables}_j!) \\
 \rho^P &: \square(\text{planJourney}_j \Rightarrow \diamond \text{directions}_j!) & \rho^{MS} &: \square(\text{getRoutes}_j \Rightarrow \diamond \text{routes}_j!) \\
 \rho_1^P &: \square(\text{getRoutes}_j \Rightarrow \diamond \text{routes}_j!) & \rho^{TS} &: \square(\text{routes}_j \Rightarrow \diamond \text{timetables}_j!)
 \end{aligned}$$

that the search for solutions is sound as well, in the sense that every computed answer to $\exists \mathfrak{o} \cdot Q$ is also a solution to $\exists \mathfrak{o} \cdot Q$. This fundamental result follows from [Theorem 3.3.24](#) and ensures, in combination with [Proposition 5.2.18](#), that the operational semantics of the service overlay given by discovery and binding is sound with respect to the notion of satisfiability of a service query.

PROPOSITION 5.2.19. *Let $\exists \mathfrak{o} \cdot Q$ be a service query derived by resolution from $\exists \mathfrak{o}_1 \cdot Q_1$ and $\forall \mathfrak{o}_2 \cdot P_2 \leftarrow R_2$ using the morphism $\theta_1: \mathfrak{o}_1 \rightarrow \mathfrak{o}$. If $\forall \mathfrak{o}_2 \cdot P_2 \leftarrow R_2$ is correct then, for any solution ψ to $\exists \mathfrak{o} \cdot Q$, $\theta_1 \circ \psi$ is a solution to $\exists \mathfrak{o}_1 \cdot Q_1$. \square*

The analogy between service-oriented computing and conventional logic programming that we have established in this chapter is presented in a condensed form in [Table 5.1](#). To summarize, our approach to the logic-programming semantics of services is based on the identification of the binding of terms to variables in logic programming with the binding of orchestrations of services to those of software applications in service-oriented computing; the answer to a service query – the request for external services – is obtained through resolution using service clauses – orchestrated service interfaces – that are available from a repository. This departs from other works on the logic-programming semantics of service-oriented computing such as [\[KBG07\]](#) that actually considered implementations of the service discovery and binding mechanisms based on constraint logic programming.

TABLE 5.1. Correspondence between concepts of relational and service-oriented logic programming

CONCEPT	RELATIONAL LOGIC PROGRAMMING	SERVICE-ORIENTED LOGIC PROGRAMMING	OVER RELATIONAL NETWORKS
	OVER A SIGNATURE $\langle F, P \rangle$	OVER PROGRAM EXPRESSIONS	OVER RELATIONAL NETWORKS
Variable	pair (x, F_0)	program variable $pgm : eXp$	requires-point $x \in X$
Term	syntactic structure $\sigma(t_1, \dots, t_n)$	program statement $\text{while } C \text{ do}$ $\quad \vdash pgm$ done	subnetwork determined by a point 
Clause	universally quantified implication $C \leftarrow \overline{x} H$	program module 	service module 
Query	existentially quantified conjunction $\vdash \overline{x} Q$	program query 	client application 
Unification and resolution	term unification and first-order resolution	program discovery and binding (see Figure 5.2)	service discovery and binding (see Figure 5.11)

6

SOLVING QUERIES OVER MODULAR LOGIC PROGRAMS

As in many other areas of computer science, in logic programming, modularization or structuring techniques are an essential element in managing the inherent complexity of large software systems – in this case, logic programs – at various stages of their development. This chapter is devoted to the study of modularization over the model-theoretic framework of abstract logic programming advanced in [Section 3.3](#). More specifically, we investigate a series of properties related to the preservation and the reflection of both solutions and computed answers along morphisms of programs. We show that the preservation of solutions holds in general, thus ensuring that the general procedure for computing answers to queries is sound with respect to the modularization of logic programs, and identify a set of sufficient conditions under which the reflection property holds as well.

To start with, let us briefly examine how the specification-building operators of [\[ST88a\]](#) – which correspond to one of the most important general directions within the institution-independent studies of modularization – can be used in constructing modular logic programs. This leads to an alternative to the theory-oriented approach to structuring programs that we discussed in [Example 3.3.6](#) in the context of relational logic programming. Both structuring mechanisms are particular instances of the general axiomatic approach proposed in [Section 3.3](#), hence we can treat them uniformly by studying modularization over abstract logic-programming languages.

6.1 PROGRAM MODULES

In order to accommodate both the so-called loose semantics specific to relational logic programming and the free semantics necessary for specifying abstract data types, the equational logic programs presented in [Section 4.1](#) are defined by means of modules like NAT and ADD that are built from finite presentations over $\mathcal{FOL}_=$ by iteration of *structuring operators* such as union,

translation, and free semantics. Various other operators dedicated, for instance, to the derivation or the extension of modules (often parameterized by classes of signature morphisms or homomorphisms) have been considered in the specification literature [see [Boro2](#); [ST11](#); [DT11](#)]. To keep the presentation simple, we focus here only on the three aforementioned operators.

The equational logic-programming language $\mathcal{FOL}_{=}^{\text{struct}}$ defines *programs* as ‘terms’ formed from the programs of $\mathcal{FOL}_{=}^{\text{pres}}$ by repeated applications of the union, translation, and free-semantics operators listed below – together with the appropriate images of the resulting program modules under the functors Sign , Ax , and PMod . Similarly to the case of presentations, the *morphisms of programs* $\nu: P \rightarrow P'$ are morphisms of signatures $\nu: \text{Sign}(P) \rightarrow \text{Sign}(P')$ such that $M' \upharpoonright_{\nu} \in |\text{PMod}(P)|$ for every model $M' \in |\text{PMod}(P')|$.

UNION. For any two program modules P_1 and P_2 having the same signature Σ , the union $P_1 \cup P_2$ is also a program module, with

$$\text{Sign}(P_1 \cup P_2) = \Sigma$$

$$\text{Ax}(P_1 \cup P_2) = \text{Ax}(P_1) \cup \text{Ax}(P_2)$$

$$\text{PMod}(P_1 \cup P_2) = \text{PMod}(P_1) \cap \text{PMod}(P_2).$$

TRANSLATION. For any program module P and any signature morphism $\varphi: \text{Sign}(P) \rightarrow \Sigma'$, translate P by φ is also a program module, with

$$\text{Sign}(\text{translate } P \text{ by } \varphi) = \Sigma'$$

$$\text{Ax}(\text{translate } P \text{ by } \varphi) = \varphi(\text{Ax}(P))$$

$$\text{PMod}(\text{translate } P \text{ by } \varphi) = \text{Mod}(\varphi)^{-1}(\text{PMod}(P)).$$

FREE SEMANTICS. For any two program modules P and P' , and any signature morphism $\varphi: \text{Sign}(P) \rightarrow \text{Sign}(P')$, free P' over P through φ is also a program module, with

$$\text{Sign}(\text{free } P' \text{ over } P \text{ through } \varphi) = \text{Sign}(P')$$

$$\text{Ax}(\text{free } P' \text{ over } P \text{ through } \varphi) = \text{Ax}(P')$$

$$\text{PMod}(\text{free } P' \text{ over } P \text{ through } \varphi) = \text{the full subcategory of } \text{PMod}(P')$$

given by those P' -models that are free
with respect to φ over some P -model.

REMARK 6.1.1. To describe the modules **NAT** and **ADD**, note that, for any program module P , free P is an abbreviation for free P over \emptyset through ι , where

ι is the inclusion $\emptyset \subseteq \text{Sign}(P)$, and that, for any other program module P' such that $\text{Sign}(P)$ is a subsignature of $\text{Sign}(P')$, P then P' is an abbreviation for (translate P by ι) $\cup P'$, where ι is the inclusion $\text{Sign}(P) \subseteq \text{Sign}(P')$.

6.2 FOUNDATIONS OF MODULAR LOGIC PROGRAMMING

Throughout this chapter, we will assume \mathcal{L} to be an arbitrary but fixed logic-programming language $\langle \text{LIP}, \text{Sign}, \text{PMod}, \text{Ax} \rangle$ over a framework $\mathcal{F} = \langle \mathcal{GS}, \text{C}, \text{Q}, \text{I} \rangle$.

Our first result concerns the preservation of queries and solutions (the denotational concept of answer to a query) along morphisms of programs.

PROPOSITION 6.2.1. *Let $\exists X \cdot \rho$ be a query over a signature Σ .*

- 1 *For every morphism of signatures $\nu: \Sigma \rightarrow \Sigma'$, $\nu(\exists X \cdot \rho)$ is a query over Σ' .*
- 2 *For every morphism of programs $\nu: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$ and every $\langle \Sigma, \Gamma \rangle$ -solution $\psi: X \rightarrow Y$ to $\exists X \cdot \rho$, $\Psi_\nu(\psi)$ is a $\langle \Sigma', \Gamma' \rangle$ -solution to $\nu(\exists X \cdot \rho)$.*

PROOF. For the first part, note that, by the definition of the translation of quantified sentences (see Section 3.2.1), $\nu(\exists X \cdot \rho) = \exists \Psi_\nu(X) \cdot \alpha_{\nu, X}(\rho)$. Since Q is a generalized subfunctor of LSen , it follows that $\alpha_{\nu, X}(\rho) \in \text{Q}_{\Sigma'}(\Psi_\nu(X))$. Therefore, the quantified sentence $\exists \Psi_\nu(X) \cdot \alpha_{\nu, X}(\rho)$ is a Σ' -query.

For the second part, according to Definition 3.3.8, we need to check that $\Psi_\nu(Y)$ is conservative and that $\langle \Sigma', \Gamma' \rangle \models_{\Sigma'}^{\text{lp}} \forall \Psi_\nu(Y) \cdot \Psi_\nu(\psi)(\alpha_{\nu, X}(\rho))$. Let us thus suppose that M' is a Σ' -model. Since Y is conservative (because ψ is a solution), there exists a Y -expansion N of $M' \upharpoonright_\nu$. It follows that M' and N are models of Σ' and Y , respectively, such that $M' \upharpoonright_\nu = N \upharpoonright_\Sigma$. This further implies, by the weak model-amalgamation property of \mathcal{GS} , that there exists a $\Psi_\nu(Y)$ -expansion N' of M' such that $\beta_{\nu, Y}(N') = N$.

To prove that $\langle \Sigma', \Gamma' \rangle$ entails $\forall \Psi_\nu(Y) \cdot \Psi_\nu(\psi)(\alpha_{\nu, X}(\rho))$, note that, by the satisfaction condition for the institution of quantified sentences over \mathcal{GS} , $\langle \Sigma, \Gamma \rangle \models_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho)$ implies $\langle \Sigma', \Gamma' \rangle \models_{\Sigma'}^{\text{lp}} \nu(\forall Y \cdot \psi(\rho))$.¹ This means that $\langle \Sigma', \Gamma' \rangle \models_{\Sigma'}^{\text{lp}} \forall \Psi_\nu(Y) \cdot \alpha_{\nu, Y}(\psi(\rho))$ because, by the definition of the ν -translation of quantified sentences, $\nu(\forall Y \cdot \psi(\rho)) = \forall \Psi_\nu(Y) \cdot \alpha_{\nu, Y}(\psi(\rho))$. Moreover, according to the general properties of the morphism of substitution systems $\langle \Psi_\nu, \kappa_\nu, \tau_\nu \rangle$ induced by ν , the square depicted below is commutative, from

¹ Recall that, since ν is an LIP-morphism, the ν -reduct of any $\langle \Sigma', \Gamma' \rangle$ -model is a $\langle \Sigma, \Gamma \rangle$ -model.

which we deduce that $\alpha_{v,Y}(\psi(\rho)) = \Psi_v(\psi)(\alpha_{v,X}(\rho))$.

$$\begin{array}{ccc} \text{Sen}_{\Sigma}(X) & \xrightarrow{\alpha_{v,X}} & \text{Sen}_{\Sigma'}(\Psi_v(X)) \\ \psi(_) \downarrow & & \downarrow \Psi_v(\psi)(_) \\ \text{Sen}_{\Sigma}(Y) & \xrightarrow{\alpha_{v,Y}} & \text{Sen}_{\Sigma'}(\Psi_v(Y)) \end{array}$$

We thus conclude that $\langle\langle \Sigma', \Gamma' \rangle\rangle \models_{\Sigma'}^{\text{lp}} \forall \Psi_v(Y) \cdot \Psi_v(\psi)(\alpha_{v,X}(\rho))$. \square

Unlike the preservation property of [Proposition 6.2.1](#), the reflection of solutions can only be guaranteed for those morphisms $\nu: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ that are *conservative* – in the sense that every model of the program $\langle\langle \Sigma, \Gamma \rangle\rangle$ admits a $\langle\langle \Sigma', \Gamma' \rangle\rangle$ -expansion along ν – and that, moreover, *lift substitutions*.

DEFINITION 6.2.2 (Lifting substitutions). A morphism of logic programs $\nu: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ is said to *lift substitutions* if for every Σ' -substitution $\psi': \Psi_v(X) \rightarrow Y'$ there is a Σ -substitution $\psi: X \rightarrow Y$ such that $\Psi_v(\psi) = \psi'$.

Although the use of conservativity to support modularization is often considered as one of the fundamental principles of structuring logic programs or specifications [see e.g. [ST11](#)], this property is in general difficult to verify, as it is well known that even in the case of first-order logic it has no recursively axiomatized complete calculus [see [MAHo1](#)]. Nevertheless, we can easily find additional assumptions that guarantee conservativeness. For instance, in the logic-programming language $\mathcal{F}^{\text{pres}}$, for any morphism of programs $\nu: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ such that Γ' and $\nu(\Gamma)$ are semantically equivalent (or, even more, equal), the property is independent of the set of clauses, and thus it suffices to consider ν as a signature morphism.² In relational first-order logic, this means that all we have to check is that ν is injective on both operation and relation symbols [see e.g. [Diao8](#)]. The lifting-substitutions property further requires that ν is surjective on operation symbols.

FACT 6.2.3. In the generalized substitution system QF-FOL_{\neq}^1 , a signature morphism $\nu: \langle\langle F, P \rangle\rangle \rightarrow \langle\langle F', P' \rangle\rangle$ is conservative and lifts substitutions if and only if it is bijective on operation symbols and injective on relation symbols.

A similar result can be obtained for equational logic programming.

FACT 6.2.4. In $\text{QF-FOL}_{=}$, a signature morphism $\nu: \langle\langle S, F \rangle\rangle \rightarrow \langle\langle S', F' \rangle\rangle$ is conservative and lifts substitutions if and only if it is injective on all symbols and, for every sort $s \in S$ and operation symbol $\sigma' \in F'_{w' \rightarrow \nu^{\text{st}}(s)}$, there exists an operation symbol $\sigma \in F_{w \rightarrow s}$ such that $\nu^{\text{st}}(w) = w'$ and $\nu_{w \rightarrow s}^{\text{op}}(\sigma) = \sigma'$.

² Alternatively, one could consider $\mathcal{F}^{\text{struc}}$ -morphisms of logic programs $\nu: P \rightarrow P'$ whose codomain P' is semantically equivalent with $\text{translate } P \text{ by } \text{Sign}(\nu)$.

Note that, even though the conditions above may seem overly restrictive, they are often the best that can be achieved. For example, assume that ν is the inclusion of first-order signatures $\langle \{\emptyset: 0\}, \{\text{add}: 3\} \rangle \subseteq \langle F_{\text{NAT}}, P_{\text{NAT}} \rangle$, where $\langle F_{\text{NAT}}, P_{\text{NAT}} \rangle$ is the signature of natural numbers discussed in Section 3.1.1, that Γ is the singleton set containing the clause $\forall \{M\} \cdot \text{true} \Rightarrow \text{add}(M, \emptyset, M)$, Γ' is $\nu(\Gamma)$, and consider the query $\exists \{X\} \cdot \text{add}(X, \emptyset, X)$. Then the morphism ν is conservative, and yet we may obtain solutions to the translated query $\nu(\exists \{X\} \cdot \text{add}(X, \emptyset, X))$, such as $X \mapsto s0$, that are not in the image of Ψ_ν .

PROPOSITION 6.2.5. *Consider a morphism $\nu: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$ and a Σ -query $\exists X \cdot \rho$. If ν is conservative and lifts substitutions, then for every $\langle \Sigma', \Gamma' \rangle$ -solution ψ' to $\nu(\exists X \cdot \rho)$ there exists a $\langle \Sigma, \Gamma \rangle$ -solution ψ to $\exists X \cdot \rho$ such that $\Psi_\nu(\psi) = \psi'$.*

PROOF. Let $\psi': \Psi_\nu(X) \rightarrow Y'$ be a $\langle \Sigma', \Gamma' \rangle$ -solution to $\exists \Psi_\nu(X) \cdot \alpha_{\nu, X}(\rho)$, that is to the ν -translation of the query $\exists X \cdot \rho$. Since the morphism ν lifts substitutions, we know that there exists a Σ -substitution $\psi: X \rightarrow Y$ such that $\Psi_\nu(Y) = Y'$ and $\Psi_\nu(\psi) = \psi'$. Therefore, all we need to establish is that the signature of Σ -variables Y is conservative and that $\langle \Sigma, \Gamma \rangle \models_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho)$.

The conservativeness of Y is an immediate consequence of the conservativeness of ν and $\Psi_\nu(Y)$: every Σ -model M admits a ν -expansion M' (because ν is conservative), which further admits a $\Psi_\nu(Y)$ -expansion N' (because $\Psi_\nu(Y)$ is conservative); then, based on the commutativity of the diagram below, $\beta_{\nu, Y}(N')$ is a Y -expansion of M .

$$\begin{array}{ccc} \text{Mod}(\Sigma) & \xleftarrow{-\uparrow_\nu} & \text{Mod}(\Sigma') \\ \uparrow -\uparrow_\Sigma & & \uparrow -\uparrow_{\Sigma'} \\ \text{Mod}_\Sigma(Y) & \xleftarrow{\beta_{\nu, Y}} & \text{Mod}_{\Sigma'}(\Psi_\nu(Y)) \end{array}$$

For the second property, note that, since $\Psi_\nu(\psi)$ is a $\langle \Sigma', \Gamma' \rangle$ -solution to the query $\exists \Psi_\nu(X) \cdot \alpha_{\nu, X}(\rho)$, we have $\langle \Sigma', \Gamma' \rangle \models_{\Sigma'}^{\text{lp}} \forall \Psi_\nu(Y) \cdot \Psi_\nu(\psi)(\alpha_{\nu, X}(\rho))$. This means that $\forall \Psi_\nu(Y) \cdot \alpha_{\nu, Y}(\psi(\rho))$ is a semantic consequence of $\langle \Sigma', \Gamma' \rangle$, because the sentence translations $\alpha_{\nu, X} \circ \Psi_\nu(\psi)$ and $\psi \circ \alpha_{\nu, Y}$ are equal. Thus, by the definition of the translation of quantified sentences, we deduce that

$$\langle \Sigma', \Gamma' \rangle \models_{\Sigma'}^{\text{lp}} \nu(\forall Y \cdot \psi(\rho)).$$

Suppose now that M is a model of the program $\langle \Sigma, \Gamma \rangle$. Since the morphism ν is conservative, there exists a ν -expansion M' of M such that $M' \models_{\Sigma'}^{\text{lp}} \langle \Sigma', \Gamma' \rangle$. As a result, $M' \models_{\Sigma'}^{\text{qs}} \nu(\forall Y \cdot \psi(\rho))$, which further implies, by the satisfaction condition for ν , that $M \models_{\Sigma}^{\text{qs}} \forall Y \cdot \psi(\rho)$. Therefore, $\langle \Sigma, \Gamma \rangle \models_{\Sigma}^{\text{lp}} \forall Y \cdot \psi(\rho)$. \square

6.3 COMPUTING ANSWER SUBSTITUTIONS

From an operational point of view, it is also useful to investigate the preservation and the reflection of computed answers (to given queries) along morphisms of programs, as this may enable us to overcome some of the limitations imposed by query-completeness or conservativeness. For this purpose, we will only consider morphisms that are *simple*, that is morphisms of logic programs $\nu: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$ for which $\nu(\Gamma) \subseteq \Gamma'$.

Let us first notice that trivial queries are both preserved and reflected (under mild additional assumptions) along every signature morphism φ ; this follows from the satisfaction condition for the morphism of substitution systems $\langle \Psi_\varphi, \kappa_\varphi, \tau_\varphi \rangle$ determined by φ in combination with the model-amalgamation property of the generalised substitution system \mathcal{GS} or, in the case of reflection, with the conservativeness of φ .

FACT 6.3.1. For every morphism of signatures $\varphi: \Sigma \rightarrow \Sigma'$, the translation of every trivial Σ -query $\exists Y \cdot \top$ is a trivial query as well. Moreover, if φ is conservative, then a query $\exists Y \cdot \top$ is trivial whenever $\varphi(\exists Y \cdot \top)$ is trivial.

LEMMA 6.3.2. For every simple morphism of programs $\nu: \langle \Sigma, \Gamma \rangle \rightarrow \langle \Sigma', \Gamma' \rangle$, every (one-step) Γ -derivation $\exists X_1 \cdot \rho_1 \xrightarrow{\Gamma, \theta_1} \exists X_2 \cdot \rho_2$ determines a Γ' -derivation

$$\nu(\exists X_1 \cdot \rho_1) \xrightarrow{\Gamma', \Psi_\nu(\theta_1)} \nu(\exists X_2 \cdot \rho_2).$$

PROOF. Assume that $\exists X_2 \cdot \rho_2$ is a query derived by resolution from $\exists X_1 \cdot \rho_1$ and Γ using the computed substitution $\theta_1: X_1 \rightarrow X_2$. It follows that there exists a clause $\forall Y_1 \cdot \gamma_1 \in \Gamma$ and a substitution $\psi_1: Y_1 \rightarrow X_2$ such that $\theta_1(\rho_1), \psi_1(\gamma_1) \Vdash_{\Sigma, X_2} \rho_2$. Then, by the functoriality of \Vdash , we know that $\alpha_{\nu, X_2}(\theta_1(\rho_1)), \alpha_{\nu, X_2}(\psi_1(\gamma_1)) \Vdash_{\Sigma', \Psi_\nu(X_2)} \alpha_{\nu, X_2}(\rho_2)$. This allows us to deduce, based on the naturality of α_ν , that

$$\Psi_\nu(\theta_1)(\alpha_{\nu, X_1}(\rho_1)), \Psi_\nu(\psi_1)(\alpha_{\nu, Y_1}(\gamma_1)) \Vdash_{\Sigma', \Psi_\nu(X_2)} \alpha_{\nu, X_2}(\rho_2).$$

Therefore, the translated query $\exists \Psi_\nu(X_2) \cdot \alpha_{\nu, X_2}(\rho_2)$ can be derived by resolution from $\exists \Psi_\nu(X_1) \cdot \alpha_{\nu, X_1}(\rho_1)$ and $\forall \Psi_\nu(Y_1) \cdot \alpha_{\nu, Y_1}(\gamma_1)$ using the computed substitution $\Psi_\nu(\theta_1)$. By the definition of the translation of quantified sentences along signature morphisms, we further deduce that $\nu(\exists X_2 \cdot \rho_2)$ can be derived by resolution from $\nu(\exists X_1 \cdot \rho_1)$ and $\nu(\forall Y_1 \cdot \gamma_1)$ using the computed morphism $\Psi_\nu(\theta_1)$. We can thus conclude, based on the fact that ν is simple, that $\nu(\exists X_2 \cdot \rho_2)$ can be derived from $\nu(\exists X_1 \cdot \rho_1)$ and Γ' . \square

The preservation of computed answers along simple morphisms of logic

programs follows by repeated applications of [Lemma 6.3.2](#) and by [Fact 6.3.1](#).

PROPOSITION 6.3.3. *Let $\nu: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ be a simple morphism of logic programs and $\theta: X \rightarrow Y$ a computed $\langle\langle \Sigma, \Gamma \rangle\rangle$ -answer to a Σ -query $\exists X \cdot \rho$. Then the substitution $\Psi_\nu(\theta)$ is a computed $\langle\langle \Sigma', \Gamma' \rangle\rangle$ -answer to $\nu(\exists X \cdot \rho)$. \square*

As in the case of solutions, the reflection of computed answers relies on additional hypotheses, which correspond here to the reflection of final rules.

DEFINITION 6.3.4 (Reflection of final rules). A simple morphism of programs $\nu: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ reflects final (goal-directed) rules if for every conservative signature of Σ -variables X , X -query ρ , and trivial $\Psi_\nu(X)$ -query \top' such that

$$\alpha_{\nu, X}(\rho), \Psi_\nu(X)(\Gamma') \Vdash_{\Sigma', \Psi_\nu(X)}^* \top'$$

there exists a trivial X -query \top such that

$$\rho, X(\Gamma) \Vdash_{\Sigma, X}^* \top.$$

It is rather straightforward to see that the reflection of final rules is satisfied by most simple morphisms of logic programs used for structuring purposes. For instance, in $(\mathcal{FOL}_\neq^1)^{\text{pres}}$, a simple morphism of logic programs $\nu: \langle\langle F, P \rangle, \Gamma \rangle \rightarrow \langle\langle F', P' \rangle, \Gamma' \rangle$ reflects final goal-directed rules if it does not introduce new information concerning the old predicates: for every clause $\forall Y' \cdot \bigwedge H' \Rightarrow \pi'(t'_1, \dots, t'_n)$ in $\Gamma' \setminus \nu(\Gamma)$, π' is a relation symbol in $P'_n \setminus \nu_n^{\text{rel}}(P_n)$.

LEMMA 6.3.5. *Let $\nu: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ be a simple morphism of programs such that $\text{Sign}(\nu): \Sigma \rightarrow \Sigma'$ is conservative. The following statements are equivalent:*

- 1 For every Σ -query $\exists X \cdot \rho$ and substitution $\theta: X \rightarrow Y$, θ is a computed $\langle\langle \Sigma, \Gamma \rangle\rangle$ -answer to $\exists X \cdot \rho$ whenever $\Psi_\nu(\theta)$ is a computed $\langle\langle \Sigma', \Gamma' \rangle\rangle$ -answer to $\nu(\exists X \cdot \rho)$.
- 2 The morphism ν reflects final goal-directed rules.

PROOF.

1 \Rightarrow 2. Suppose X is a conservative signature of Σ -variables, ρ is an X -query, and \top' is a trivial $\Psi_\nu(X)$ -query such that $\alpha_{\nu, X}(\rho), \Psi_\nu(X)(\Gamma') \Vdash_{\Sigma', \Psi_\nu(X)}^* \top'$. By [Proposition 3.3.30](#), it follows that $\exists \Psi_\nu(X) \cdot \alpha_{\nu, X}(\rho) \twoheadrightarrow_{\Gamma', 1_{\Psi_\nu(X)}}^* \exists \Psi_\nu(X) \cdot \top'$. Since X is conservative and \mathcal{GS} has weak model-amalgamation, we deduce that $\Psi_\nu(X)$ is also conservative, and thus that $\exists \Psi_\nu(X) \cdot \top'$ is trivial. Therefore, by hypothesis, there exists a trivial X -query \top such that $\exists X \cdot \rho \twoheadrightarrow_{\Gamma, 1_X}^* \exists X \cdot \top$. As a result, by [Proposition 3.3.30](#), we obtain $\rho, X(\Gamma) \Vdash_{\Sigma, X}^* \top$.

2 \Rightarrow 1. Let us now consider $\exists X \cdot \rho$ to be a Σ -query and θ to be a substitution for which $\Psi_v(\theta)$ is a computed answer to $v(\exists X \cdot \rho)$, meaning that there exists a trivial $\Psi_v(Y)$ -query \top' such that $v(\exists X \cdot \rho) \rightarrow_{\Gamma', \Psi_v(\theta)}^* \exists \Psi_v(Y) \cdot \top'$. By [Proposition 3.3.30](#), $\Psi_v(\theta)(\alpha_{v,X}(\rho)), \Psi_v(Y)(\Gamma') \Vdash_{\Sigma', \Psi_v(Y)}^* \top'$, from which we deduce, by the naturality of α_v , that $\alpha_{v,Y}(\theta(\rho)), \Psi_v(Y)(\Gamma') \Vdash_{\Sigma', \Psi_v(Y)}^* \top'$. Moreover, since both $\text{Sign}(v)$ and $\Psi_v(Y)$ are conservative, it follows that Y is conservative as well. This means that we can use the reflection of final goal-directed rules to derive the existence of a trivial Y -query \top such that $\theta(\rho), Y(\Gamma) \Vdash_{\Sigma, Y}^* \top$. By applying [Proposition 3.3.30](#) one last time, we conclude that θ is a computed $\langle\langle \Sigma, \Gamma \rangle\rangle$ -answer to $\exists X \cdot \rho$. \square

Note that, as opposed to solutions, in the case of computed answers we are able to fully characterize their reflection along morphisms of logic programs.

PROPOSITION 6.3.6. *Let $v: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ be a simple morphism of logic programs, $\exists X \cdot \rho$ a Σ -query, and θ' a computed $\langle\langle \Sigma', \Gamma' \rangle\rangle$ -answer to $v(\exists X \cdot \rho)$. If $\text{Sign}(v)$ is conservative, and if v lifts substitutions and reflects final goal-directed rules, there is a computed $\langle\langle \Sigma, \Gamma \rangle\rangle$ -answer θ to $\exists X \cdot \rho$ such that $\Psi_v(\theta) = \theta'$. \square*

[Propositions 6.3.3](#) and [6.3.6](#) enable the computation of all solutions to certain queries even in situations in which the given logic program is not query-complete, or in which it constitutes the result of an extension of logic programs that is not known to be conservative.

COROLLARY 6.3.7. *Consider a simple morphism $v: \langle\langle \Sigma, \Gamma \rangle\rangle \rightarrow \langle\langle \Sigma', \Gamma' \rangle\rangle$ between programs $\langle\langle \Sigma, \Gamma \rangle\rangle$ and $\langle\langle \Sigma', \Gamma' \rangle\rangle$, and a Σ -query $\exists X \cdot \rho$ that admits an identity in Γ .*

- *If v is conservative and lifts substitutions, and if $\langle\langle \Sigma, \Gamma \rangle\rangle$ is query-complete, every $\langle\langle \Sigma', \Gamma' \rangle\rangle$ -solution to $v(\exists X \cdot \rho)$ is also a computed $\langle\langle \Sigma', \Gamma' \rangle\rangle$ -answer to $v(\exists X \cdot \rho)$.*
- *If $\langle\langle \Sigma', \Gamma' \rangle\rangle$ is query-complete, $\text{Sign}(v)$ is conservative, and v lifts substitutions and reflects final goal-directed rules, then every $\langle\langle \Sigma', \Gamma' \rangle\rangle$ -solution to $v(\exists X \cdot \rho)$ is the image under v of a $\langle\langle \Sigma, \Gamma \rangle\rangle$ -solution to $\exists X \cdot \rho$. \square*

The schematic representation in [Figure 6.1](#) summarizes the various relationships between solutions and computed answers in connection with morphisms of logic programs. To this end, we consider v to be a morphism between logic programs $\langle\langle \Sigma, \Gamma \rangle\rangle$ and $\langle\langle \Sigma', \Gamma' \rangle\rangle$, and $\exists X \cdot \rho$ to be a Σ -query.

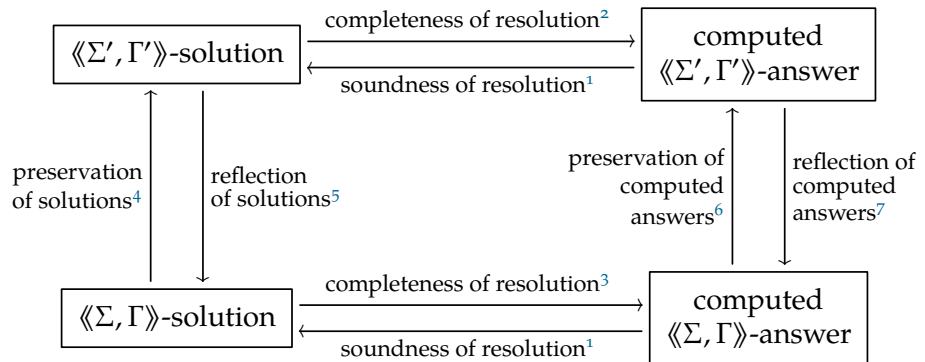


FIGURE 6.1. Relationships between solutions and computed answers

- 1 by Theorem 3.3.24
- 2 by Theorem 3.3.35, based on the query-completeness of $\langle\langle \Sigma', \Gamma' \rangle\rangle$ and on the assumption that the translated query $\nu(\exists X \cdot \rho)$ admits an identity in Γ'
- 3 by Theorem 3.3.35, based on the query-completeness of $\langle\langle \Sigma, \Gamma \rangle\rangle$ and on the assumption that the original query $\exists X \cdot \rho$ has an identity in Γ
- 4 by Proposition 6.2.1
- 5 by Proposition 6.2.5, based on the conservativeness of ν and on the lifting of substitutions
- 6 by Proposition 6.3.3, based on the assumption that ν is simple
- 7 by Proposition 6.3.6, based on the assumptions that ν is simple, it lifts substitutions and reflects final goal-directed rules, and its underlying signature morphism is conservative

CONCLUSIONS AND FURTHER WORK

In this thesis, we have advanced an abstract axiomatic theory of logic programming by identifying and examining in an institution-theoretic setting two of the most basic principles of the paradigm: (a) the fact that each logic program has a rigorous mathematical semantics given by a class of models (and often by a ‘standard’ model of that class) that determines the specific set of queries that can be positively answered, and (b) the existence of a sound (and in some cases complete) goal-directed procedure for computing answer-substitutions that confirm the validity of the positive answer received by a query. In this way, the present study unifies the relational and the equational variants of logic programming, and we can expect it to integrate with ease many other derived forms of the paradigm, such as constraint [see [Diao0](#)] or behavioural [see [GMK02](#)] logic programming, which share many similarities with the equational variant presented in [Section 4.1](#).

The logic-programming semantics of services discussed in [Section 5.2](#) provides us a significantly different form of logic programming – even though the logic programs (known in this case as repositories) and their corresponding operational semantics are defined similarly to their relational counterparts. This is mainly due to the fact that the signatures of variables (formalized as labelled hypergraphs) can no longer be treated as extensions of signatures (which provide the structures to be used as labels for the hypergraphs that underlie the signatures of variables). Consequently, service-oriented substitutions cannot be captured by the institution-independent concept of substitution considered in [[Diao4](#)], nor can they be expressed as generalized forms of signature morphisms as in [[GFO12](#); [MKMip](#)].

To be more precise, the theory of services that we have developed in this thesis is grounded on a declarative semantics of service clauses defined over a novel logical system of orchestration schemes. The structure of the sentences and of the models of this logical system varies according to the orchestration scheme under consideration. For example, when orchestrations are defined as asynchronous relational networks over the institution $\underline{\text{ALTL}}$, we obtain

sentences as linear-temporal-logic sentences expressing properties observed at given interaction points of a network, and models in the form of ground orchestrations of Muller automata. Other logics (with corresponding model theory) could have been used instead of the automata-based variant of linear temporal logic, more specifically any institution such that (a) the category of signatures is (finitely) cocomplete; (b) there exist cofree models along every signature morphism; (c) the category of models of every signature has (finite) products; and (d) model homomorphisms reflect the satisfaction of sentences. Moreover, the formalism used in defining orchestrations can change by means of morphisms of orchestration schemes. We could consider, for instance, an encoding of the hypergraphs of processes and connections discussed in [Section 5.1.2](#) into graph-based structures that are similar to those presented in [\[FL13a\]](#); or we could change their underlying institution by adding new temporal modalities (along the lines of [Example 5.2.2](#)) or by considering other classes of automata, like the closed reduced Büchi automata used in [\[AS87; FL13b\]](#). This encourages us to further investigate aspects related to the heterogeneous foundations of service-oriented computing based on the proposed logical system of orchestration schemes.

In the more general context of abstract logic-programming languages, our efforts have focused mainly on the development of a simple yet sufficiently rich theoretical framework to allow the investigation of properties related to the satisfaction of quantified sentences, the generalization of Herbrand's theorem and, moreover, the definition of a sound and (conditionally) complete procedure for computing solutions to queries. This procedure relies on exploring a potentially infinite search space of queries, related through substitutions computed by means of resolution, in search of queries that are trivial, i.e. known to admit the simplest possible solutions. Its implementation is not complete because we do not commit to any particular search strategy, which could make use, for example, of most general unifiers and backtracking. However, it should be noted that under the present formalization of the search space, any such strategy would be sound, and even complete if it ensured that the reachability of all trivial queries is maintained.

Thus, apart from the obvious need to consider various other forms of logic programming, a very interesting direction for future research is to further formalize and examine the notion of strategy, especially in connection with the modularization of logic programs. To that end, one would have to take into account the fact that the sets of clauses associated with logic programs are often structured, in the sense that certain clauses take precedence over others in the computation of the next (derived) query to be solved. Moreover,

given strategies could make use only of most general unifiers, hence imposing additional restrictions on computed substitutions. All these aspects could be dealt with abstractly by narrowing the search space of queries. In that case, the challenge lies in determining the basic conditions under which properties such as the preservation of computed answers still hold.

Another open problem of practical importance is the development of an appropriate concept of map of logic-programming languages to capture, for example, the encoding of relational logic programming into its equational correspondent based on the representation of relations as Boolean-valued operations [see e.g. [Dia08](#)]. Even though from a denotational perspective we obtain an immediate answer in the form of the notion of morphism of generalized substitution systems (suggested in [Definition 3.2.6](#)), from an operational point of view the answer does not appear to be equally obvious since different logic-programming frameworks may be founded on highly different kinds of goal-directed rules.

BIBLIOGRAPHY

- [AHS09] Jiri Adámek, Horst Herrlich and George Strecker. *Abstract and Concrete Categories: The Joy of Cats*. Dover books on mathematics. reprint. Dover Publications, 2009 (cited on pages [16](#), [64](#)).
- [Alo+04] Gustavo Alonso et al. *Web Services: Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, 2004 (cited on page [10](#)).
- [AS87] Bowen Alpern and Fred B. Schneider. ‘Recognizing safety and liveness’. In: *Distributed Computing* 2.3 (1987), pages 117–126 (cited on pages [85](#), [122](#)).
- [Av82] Krzysztof R. Apt and Maarten H. van Emden. ‘Contributions to the theory of logic programming’. In: *Journal of the ACM* 29.3 (1982), pages 841–862 (cited on page [47](#)).
- [BN99] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999 (cited on page [67](#)).
- [Bar74] K. Jon Barwise. ‘Axioms for abstract model theory’. In: *Annals of Mathematical Logic* 7.2–3 (1974), pages 221–265 (cited on page [95](#)).
- [BCT06] Boualem Benatallah, Fabio Casati and Farouk Toumani. ‘Representing, analysing and managing Web service protocols’. In: *Data & Knowledge Engineering* 58.3 (2006), pages 327–357 (cited on page [83](#)).
- [BPP85] Edward K. Blum and Francesco Parisi-Prisicce. ‘The semantics of shared submodules specifications’. In: *Theory and Practice of Software Development*. Edited by Hartmut Ehrig et al. Volume 185. Lecture Notes in Computer Science. Springer, 1985, pages 359–373 (cited on page [34](#)).
- [Bor02] Tomasz Borzyszkowski. ‘Logical systems for structured specifications’. In: *Theoretical Computer Science* 286.2 (2002), pages 197–245 (cited on pages [12](#), [34](#), [113](#)).

- [BZ83] Daniel Brand and Pitro Zafiropulo. ‘On communicating finite-state machines’. In: *Journal of the ACM* 30.2 (1983), pages 323–342 (cited on pages 83, 85).
- [BGLL09] Roberto Bruni, Fabio Gadducci and Alberto Lluch-Lafuente. ‘A graph syntax for processes and services’. In: *Web Services and Formal Methods*. Edited by Cosimo Laneve and Jianwen Su. Volume 6194. Lecture Notes in Computer Science. Springer, 2009, pages 46–60 (cited on page 88).
- [BLM94] Michele Bugliesi, Evelina Lamma and Paola Mello. ‘Modularity in logic programming’. In: *Journal of Logic Programming* 19/20 (1994), pages 443–502 (cited on page 11).
- [BG79] Rod M. Burstall and Joseph A. Goguen. ‘The semantics of Clear, a specification language’. In: *Abstract Software Specifications*. Edited by Dines Bjørner. Volume 86. Lecture Notes in Computer Science. Springer, 1979, pages 292–332 (cited on page 18).
- [Chu40] Alonzo Church. ‘A formulation of the simple theory of types’. In: *The Journal of Symbolic Logic* 5.2 (1940), pages 56–68 (cited on page 54).
- [Cod07] Mihai Codescu. ‘The model theory of higher order logic’. Master’s thesis. Școala Normală Superioară București, 2007 (cited on pages 57, 59, 60, 70).
- [CG08] Mihai Codescu and Daniel Găină. ‘Birkhoff completeness in institutions’. In: *Logica Universalis* 2.2 (2008), pages 277–309 (cited on page 29).
- [Col+73] Alain Colmerauer et al. *Un système de communication homme-machine en Français*. Technical report. Université de Aix-Marseille II, Marseille: Groupe de Intelligence Artificielle, Faculté des Sciences de Luminy, 1973 (cited on page 7).
- [Dia95] Răzvan Diaconescu. ‘Completeness of category-based equational deduction’. In: *Mathematical Structures in Computer Science* 5.1 (1995), pages 9–40 (cited on pages 8, 20).
- [Dia00] Răzvan Diaconescu. ‘Category-based constraint logic’. In: *Mathematical Structures in Computer Science* 10.3 (2000), pages 373–407 (cited on pages 8, 9, 20, 121).
- [Dia03] Răzvan Diaconescu. ‘Institution-independent ultraproducts’. In: *Fundamenta Informaticae* 55.3–4 (2003), pages 321–348 (cited on pages 60, 69, 70).

- [Dia04] Răzvan Diaconescu. ‘Herbrand theorems in arbitrary institutions’. In: *Information Processing Letters* 90.1 (2004), pages 29–37 (cited on pages 8, 9, 27, 29, 30, 41, 42, 55, 69, 121).
- [Dia08] Răzvan Diaconescu. *Institution-Independent Model Theory*. Studies in Universal Logic. Birkhäuser, 2008 (cited on pages 9, 20, 29, 35, 53, 55, 57, 60, 69, 70, 80, 115, 123).
- [Dia10] Răzvan Diaconescu. ‘Quasi-Boolean encodings and conditionals in algebraic specification’. In: *Journal of Logic and Algebraic Programming* 79.2 (2010), pages 174–188 (cited on page 57).
- [Dia11] Răzvan Diaconescu. ‘Structural induction in institutions’. In: *Information and Computation* 209.9 (2011), pages 1197–1222 (cited on page 29).
- [Dia12a] Răzvan Diaconescu. ‘An axiomatic approach to structuring specifications’. In: *Theoretical Computer Science* 433 (2012), pages 20–42 (cited on pages 12, 38).
- [Dia12b] Răzvan Diaconescu. ‘Borrowing interpolation’. In: *Journal of Logic and Computation* 22.3 (2012), pages 561–586 (cited on page 54).
- [Dia13a] Răzvan Diaconescu. ‘Institutional semantics for many-valued logics’. In: *Fuzzy Sets and Systems* 218 (2013), pages 32–52 (cited on page 57).
- [Dia13b] Răzvan Diaconescu. ‘Quasi-varieties and initial semantics for hybridized institutions’. In: *Journal of Logic and Computation* (2013) (cited on page 57).
- [DF98] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. Volume 6. AMAST Series in Computing. World Scientific, 1998 (cited on pages 74, 76).
- [DGS93] Răzvan Diaconescu, Joseph A. Goguen and Petros Stefanescu. ‘Logical support for modularisation’. In: *Logical Environments*. Edited by Gérard Huet and Gordon Plotkin. Cambridge University Press, 1993, pages 83–130 (cited on pages 34, 39, 59).
- [DȚ11] Răzvan Diaconescu and Ionuț Țuțu. ‘On the algebra of structured specifications’. In: *Theoretical Computer Science* 412.28 (2011), pages 3145–3174 (cited on pages 12, 113).

- [Fer+05] Gian Luigi Ferrari et al. ‘Synchronised hyperedge replacement as a model for service oriented computing’. In: *Formal Methods for Components and Objects*. Edited by Frank S. de Boer et al. Volume 4111. Lecture Notes in Computer Science. Springer, 2005, pages 22–43 (cited on page 88).
- [Fia12] José L. Fiadeiro. ‘The many faces of complexity in software design’. In: *Conquering Complexity*. Edited by Mike Hinchey and Lorcan Coyle. Springer, 2012, pages 3–47 (cited on pages 13, 73).
- [FC96] José L. Fiadeiro and José F. Costa. ‘Mirror, mirror in my hand: a duality between specifications and models of process behaviour’. In: *Mathematical Structures in Computer Science* 6.4 (1996), pages 353–373 (cited on page 80).
- [FL13a] José L. Fiadeiro and Antónia Lopes. ‘A model for dynamic re-configuration in service-oriented architectures’. In: *Software and Systems Modeling* 12.2 (2013), pages 349–367 (cited on pages 10, 72, 122).
- [FL13b] José L. Fiadeiro and Antónia Lopes. ‘An interface theory for service-oriented design’. In: *Theoretical Computer Science* 503 (2013), pages 1–30 (cited on pages 13, 73, 79, 83, 85, 86, 88, 103, 122).
- [FLB07] José L. Fiadeiro, Antónia Lopes and Laura Bocchi. ‘Algebraic semantics of service component modules’. In: *Recent Trends in Algebraic Development Techniques*. Edited by José L. Fiadeiro and Pierre-Yves Schobbens. Volume 4409. Lecture Notes in Computer Science. Springer, 2007, pages 37–55 (cited on pages 10, 72).
- [FLB11] José L. Fiadeiro, Antónia Lopes and Laura Bocchi. ‘An abstract model of service discovery and binding’. In: *Formal Aspects of Computing* 23.4 (2011), pages 433–463 (cited on pages 8, 10, 11, 72, 95, 99, 103, 105).
- [FS07] José L. Fiadeiro and Vincent Schmitt. ‘Structured co-spans: an algebra of interaction protocols’. In: *Algebra and Coalgebra in Computer Science*. Edited by Till Mossakowski, Ugo Montanari and Magne Haveraaen. Volume 4624. Lecture Notes in Computer Science. Springer, 2007, pages 194–208 (cited on page 10).

- [FK04] Ian T. Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. The Morgan Kaufmann Series in Computer Architecture and Design Series. Morgan Kaufmann, 2004 (cited on page 10).
- [Găi14] Daniel Găină. ‘Forcing, Downward Löwenheim-Skolem and Omitting Types theorems, institutionally’. In: *Logica Universalis* 8.3-4 (2014), pages 469–498 (cited on pages 29, 54).
- [Găiip] Daniel Găină. ‘Foundations of logic programming in hybridised logics’. In: *Recent Trends in Algebraic Development Techniques*. Lecture Notes in Computer Science. Springer, in press (cited on page 9).
- [GFO12] Daniel Găină, Kokichi Futatsugi and Kazuhiro Ogata. ‘Constructor-based logics’. In: *Journal of Universal Computer Science* 18.16 (2012), pages 2204–2233 (cited on pages 29, 121).
- [GP10] Daniel Găină and Marius Petria. ‘Completeness by forcing’. In: *Journal of Logic and Computation* 20.6 (2010), pages 1165–1186 (cited on pages 29, 41).
- [GMR92] Laura Giordano, Alberto Martelli and Gianfranco Rossi. ‘Extending Horn clause logic with implication goals’. In: *Theoretical Computer Science* 95.1 (1992), pages 43–74 (cited on page 11).
- [Gog89] Joseph A. Goguen. ‘What is unification?’ In: *Resolution of Equations in Algebraic Structures* 1 (1989), pages 217–261 (cited on page 44).
- [GB84] Joseph A. Goguen and Rod M. Burstall. ‘Some fundamental algebraic tools for the semantics of computation. Part 1: Comma categories, colimits, signatures and theories’. In: *Theoretical Computer Science* 31 (1984), pages 175–209 (cited on page 16).
- [GB86] Joseph A. Goguen and Rod M. Burstall. ‘A study in the foundations of programming methodology: specifications, institutions, charters and parchments’. In: *Category Theory and Computer Programming*. Edited by David H. Pitt et al. Volume 240. Lecture Notes in Computer Science. Springer, 1986, pages 313–333 (cited on page 21).
- [GB92] Joseph A. Goguen and Rod M. Burstall. ‘Institutions: abstract model theory for specification and programming’. In: *Journal of the ACM* 39.1 (1992), pages 95–146 (cited on pages 8, 18, 20, 21, 39, 98).

- [GM96] Joseph A. Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs*. Foundations of computing. MIT Press, 1996 (cited on page 74).
- [GMK02] Joseph A. Goguen, Grant Malcolm and Tom Kemp. ‘A hidden Herbrand theorem: combining the object and logic paradigms’. In: *Journal of Logic and Algebraic Programming* 51.1 (2002), pages 1–41 (cited on pages 8, 9, 20, 40, 121).
- [GM86] Joseph A. Goguen and José Meseguer. ‘EQLOG: Equality, types, and generic modules for logic programming’. In: *Logic Programming: Functions, Relations, and Equations*. Prentice Hall, 1986, pages 295–363 (cited on pages 7, 11, 52).
- [GM87] Joseph A. Goguen and José Meseguer. ‘Models and equality for logical programming’. In: *Theory and Practice of Software Development*. Edited by Hartmut Ehrig et al. Volume 250. Lecture Notes in Computer Science. Springer, 1987, pages 1–22 (cited on pages 9, 20, 40, 53).
- [GR02] Joseph A. Goguen and Grigore Roşu. ‘Institution morphisms’. In: *Formal Aspects of Computing* 13.3–5 (2002), pages 274–307 (cited on pages 20, 98).
- [GKP94] Ronald L. Graham, Donald E. Knuth and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Second. Addison-Wesley, 1994 (cited on page 79).
- [HOD96] Magne Haveraaen, Olaf Owe and Ole-Johan Dahl, editors. *Specification of Abstract Data Types*. Volume 1130. Lecture Notes in Computer Science. Springer, 1996.
- [Hen50] Leon Henkin. ‘Completeness in the theory of types’. In: *The Journal of Symbolic Logic* 15.2 (1950), pages 81–91 (cited on page 54).
- [Her67] Jacques Herbrand. ‘Investigations in proof theory’. In: *From Frege to Gödel: A Source Book in Mathematical Logic*. Source books in the history of the sciences. Harvard University Press, 1967, pages 525–581 (cited on pages 7, 9).
- [Hoa69] C. A. R. Hoare. ‘An axiomatic basis for computer programming’. In: *Communications of the ACM* 12.10 (1969), pages 576–580 (cited on pages 73, 105).
- [Ive62] Kenneth E. Iverson. *A Programming Language*. Wiley, 1962 (cited on page 79).

- [JL87] Joxan Jaffar and Jean-Louis Lassez. ‘Constraint Logic Programming’. In: *The 14th Annual ACM Symposium on Principles of Programming Languages*. ACM Press, 1987, pages 111–119 (cited on page 8).
- [Kah87] Gilles Kahn. ‘Natural semantics’. In: *The 4th Annual Symposium on Theoretical Aspects of Computer Science*. Edited by Franz-Josef Brandenburg, Guy Vidal-Naquet and Martin Wirsing. Volume 247. Lecture Notes in Computer Science. Springer, 1987, pages 22–39 (cited on page 77).
- [Kna+10] Alexander Knapp et al. ‘A heterogeneous approach to service-oriented systems specification’. In: *ACM Symposium on Applied Computing*. Edited by Sung Y. Shin et al. ACM, 2010, pages 2477–2484 (cited on page 98).
- [KBG07] Srividya Kona, Ajay Bansal and Gopal Gupta. ‘Automatic composition of semantic Web services’. In: *2007 IEEE International Conference on Web Services*. IEEE Computer Society, 2007, pages 150–158 (cited on page 110).
- [Kow74] Robert A. Kowalski. ‘Predicate logic as programming language’. In: *Proceedings of IFIP Congress 74*. Edited by Jack L. Rosenfeld. North-Holland, 1974, pages 569–574 (cited on page 7).
- [KK71] Robert A. Kowalski and Donald Kuehner. ‘Linear resolution with selection function’. In: *Artificial Intelligence 2.3/4 (1971)*, pages 227–260 (cited on page 7).
- [Llo87] John W. Lloyd. *Foundations of Logic Programming*. Symbolic computation: Artificial intelligence. Springer, 1987 (cited on pages 7, 9, 12, 23, 40, 47, 50).
- [Lödo1] Christof Löding. ‘Efficient minimization of deterministic weak ω -automata’. In: *Information Processing Letters 79.3 (2001)*, pages 105–109 (cited on page 98).
- [Mac98] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate texts in mathematics. Springer, 1998 (cited on pages 15, 16).
- [MS97] Oded Maler and Ludwig Staiger. ‘On syntactic congruences for ω -languages’. In: *Theoretical Computer Science 183.1 (1997)*, pages 93–112 (cited on page 98).

- [Mar+11] Manuel A. Martins et al. ‘Hybridization of institutions’. In: *Algebra and Coalgebra in Computer Science*. Edited by Andrea Corradini, Bartek Klin and Corina Cîrstea. Volume 6859. Lecture Notes in Computer Science. Springer, 2011, pages 283–297 (cited on pages 29, 57).
- [Mes89] José Meseguer. ‘General logics’. In: *Logic Colloquium ’87*. Edited by Heinz-Dieter Ebbinghaus et al. Volume 129. Studies in Logic and the Foundations of Mathematics Series. Elsevier, 1989, pages 275–329 (cited on pages 8, 20, 59, 100).
- [Mes92] José Meseguer. ‘Multiparadigm logic programming’. In: *Algebraic and Logic Programming*. Edited by Hélène Kirchner and Giorgio Levi. Volume 632. Lecture Notes in Computer Science. Springer, 1992, pages 158–200 (cited on pages 8, 52).
- [Mil89] Dale Miller. ‘A logical analysis of modules in logic programming’. In: *Journal of Logic Programming* 6.1&2 (1989), pages 79–108 (cited on page 11).
- [MTW87] Bernhard Möller, Andrzej Tarlecki and Martin Wirsing. ‘Algebraic specifications of reachable higher-order algebras’. In: *Abstract Data Types*. Edited by Donald Sannella and Andrzej Tarlecki. Volume 332. Lecture Notes in Computer Science. Springer, 1987, pages 154–169 (cited on pages 20, 54).
- [Mor94] Carroll C. Morgan. *Programming from Specifications*. Second. Prentice Hall International series in computer science. Prentice Hall, 1994 (cited on pages 73, 78, 105).
- [Mos02] Till Mossakowski. ‘Comorphism-based Grothendieck logics’. In: *Mathematical Foundations of Computer Science 2002*. Edited by Krzysztof Diks and Wojciech Rytter. Volume 2420. Lecture Notes in Computer Science. Springer, 2002, pages 593–604 (cited on page 21).
- [Mos06] Till Mossakowski. ‘Institutional 2-cells and Grothendieck institutions’. In: *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*. Edited by Kokichi Futatsugi, Jean-Pierre Jouannaud and José Meseguer. Volume 4060. Lecture Notes in Computer Science. Springer, 2006, pages 124–149 (cited on page 21).

- [MAH01] Till Mossakowski, Serge Autexier and Dieter Hutter. ‘Extending development graphs with hiding’. In: *Fundamental Approaches to Software Engineering*. Edited by Heinrich Hußmann. Volume 2029. Lecture Notes in Computer Science. Springer, 2001, pages 269–283 (cited on page 115).
- [MDT09] Till Mossakowski, Răzvan Diaconescu and Andrzej Tarlecki. ‘What is a logic translation?’ In: *Logica Universalis* 3.1 (2009), pages 95–124 (cited on page 21).
- [MKMip] Till Mossakowski, Ulf Krumnack and Tom Maibaum. ‘What is a derived signature morphism?’ In: *Recent Trends in Algebraic Development Techniques*. Lecture Notes in Computer Science. Springer, in press (cited on page 121).
- [Mos04] Peter Mosses. *CASL Reference Manual: The Complete Documentation Of The Common Algebraic Specification Language*. Lecture Notes in Computer Science. Springer, 2004 (cited on page 76).
- [Mul63] David E. Muller. ‘Infinite sequences and finite machines’. In: *4th Annual Symposium on Switching Circuit Theory and Logical Design*. IEEE Computer Society, 1963, pages 3–16 (cited on page 80).
- [O’K85] Richard A. O’Keefe. ‘Towards an algebra for constructing logic programs’. In: *1985 Symposium on Logic Programming*. IEEE Computer Society, 1985, pages 152–160 (cited on page 11).
- [OPE97] Fernando Orejas, Elvira Pino and Hartmut Ehrig. ‘Institutions for logic programming’. In: *Theoretical Computer Science* 173.2 (1997), pages 485–511 (cited on page 52).
- [Paw95] Wieslaw Pawlowski. ‘Context Institutions’. In: *Specification of Abstract Data Types*. Edited by Magne Haveraaen, Olaf Owe and Ole-Johan Dahl. Volume 1130. Lecture Notes in Computer Science. Springer, 1995, pages 436–457 (cited on page 32).
- [PP04] Dominique Perrin and Jean-Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics. Elsevier Science, 2004 (cited on page 80).
- [RW83] George Robinson and Lawrence Wos. ‘Paramodulation and theorem-proving in first-order theories with equality’. In: *Machine intelligence*. Edited by Bernard Meltzer and Donald Michie. Volume 4. Edinburgh University Press, 1983, pages 135–150 (cited on pages 10, 67).

- [Rob65] John A. Robinson. ‘A machine-oriented logic based on the resolution principle’. In: *Journal of the ACM* 12.1 (1965), pages 23–41 (cited on pages 7, 9).
- [Ruso8] Bertrand Russell. ‘Mathematical logic as based on the theory of types’. In: *American Journal of Mathematics* 30.3 (1908), pages 222–262 (cited on page 54).
- [ST84] Donald Sannella and Andrzej Tarlecki. ‘Building specifications in an arbitrary institution’. In: *Semantics of Data Types*. Edited by Gilles Kahn, David B. MacQueen and Gordon D. Plotkin. Volume 173. Lecture Notes in Computer Science. Springer, 1984, pages 337–356 (cited on pages 29, 55).
- [ST88a] Donald Sannella and Andrzej Tarlecki. ‘Specifications in an arbitrary institution’. In: *Information and Computation* 76.2/3 (1988), pages 165–210 (cited on pages 12, 34, 57, 112).
- [ST88b] Donald Sannella and Andrzej Tarlecki. ‘Toward formal development of programs from algebraic specifications: implementations revisited’. In: *Acta Informatica* 25.3 (1988), pages 233–281 (cited on page 73).
- [ST11] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011 (cited on pages 20, 35, 53, 54, 67, 80, 97, 113, 115).
- [SW92] Donald Sannella and Lincoln A. Wallen. ‘A calculus for the construction of modular Prolog programs’. In: *Journal of Logic Programming* 12.1&2 (1992), pages 147–177 (cited on page 11).
- [SMLo4] Lutz Schröder, Till Mossakowski and Christoph Lüth. ‘Type class polymorphism in an institutional framework’. In: *Recent Trends in Algebraic Development Techniques*. Edited by José L. Fiadeiro, Peter D. Mosses and Fernando Orejas. Volume 3423. Lecture Notes in Computer Science. Springer, 2004, pages 234–251 (cited on page 99).
- [Şero4] Traian Florin Şerbănuță. ‘Institutional concepts in first-order logic, parameterized specification, and logic programming’. Master’s thesis. University of Bucharest, 2004 (cited on page 60).

- [Su+07] Jianwen Su et al. ‘Towards a theory of web service choreographies’. In: *Web Services and Formal Methods*. Edited by Marlon Dumas and Reiko Heckel. Volume 4937. Lecture Notes in Computer Science. Springer, 2007, pages 1–16 (cited on page 10).
- [Tar86] Andrzej Tarlecki. ‘Quasi-varieties in abstract algebraic institutions’. In: *Journal of Computer and System Sciences* 33.3 (1986), pages 333–360 (cited on pages 8, 29, 34, 57, 69, 95).
- [Tar95] Andrzej Tarlecki. ‘Moving between logical systems’. In: *Specification of Abstract Data Types*. Edited by Magne Haveraaen, Olaf Owe and Ole-Johan Dahl. Volume 1130. Lecture Notes in Computer Science. Springer, 1995, pages 478–502 (cited on page 20).
- [Tar00] Andrzej Tarlecki. ‘Towards heterogeneous specifications’. In: *Frontiers of Combining Systems*. Edited by Dov M. Gabbay and Maarten van Rijke. Volume 2. Research Studies Press, 2000, pages 337–360 (cited on page 34).
- [TBG91] Andrzej Tarlecki, Rod M. Burstall and Joseph A. Goguen. ‘Some fundamental algebraic tools for the semantics of computation. Part 3: Indexed categories’. In: *Theoretical Computer Science* 91.2 (1991), pages 239–264 (cited on page 16).
- [Tho90] Wolfgang Thomas. ‘Automata on infinite objects’. In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Edited by Jan van Leeuwen. Elsevier and MIT Press, 1990, pages 133–192 (cited on pages 80, 82).
- [Tu13] Ionuț Tuțu. ‘Comorphisms of structured institutions’. In: *Information Processing Letters* 113.22–24 (2013), pages 894–900 (cited on pages 14, 38, 54, 57).
- [TF13] Ionuț Tuțu and José L. Fiadeiro. ‘A logic-programming semantics of services’. In: *Algebra and Coalgebra in Computer Science*. Edited by Reiko Heckel and Stefan Milius. Volume 8089. Lecture Notes in Computer Science. Springer, 2013, pages 299–313 (cited on pages 8, 9, 14, 20, 29, 41, 105).
- [TFipa] Ionuț Tuțu and José L. Fiadeiro. ‘From conventional to institution-independent logic programming’. In: *Journal of Logic and Computation* (in press) (cited on page 14).

- [TFipb] Ionuț Țuțu and José L. Fiadeiro. ‘Revisiting the institutional approach to Herbrand’s theorem’. In: *Algebra and Coalgebra in Computer Science*. Leibniz International Proceedings in Informatics. Schloss Dagstuhl, in press (cited on page 14).
- [TFipc] Ionuț Țuțu and José L. Fiadeiro. ‘Service-oriented logic programming’. In: *Logical Methods in Computer Science* (in press) (cited on page 14).
- [Vog03] Werner Vogels. ‘Web services are not distributed objects’. In: *IEEE Internet Computing* 7.6 (2003), pages 59–66 (cited on page 10).

INDEX OF NOTATIONS

$ \mathbf{C} $	collection of objects of a category \mathbf{C} , 15
$\mathbf{C}(A, B)$	collection of arrows from A to B in a category \mathbf{C} , 15
$f \circ g$	diagrammatic composition of two arrows f and g , 15
1_A	identity arrow of an object A , 15
$\tau \circ \sigma$	‘vertical’ composition of natural transformations τ and σ , 15
$\tau \cdot \sigma$	‘horizontal’ composition of τ and σ , 15
Set	category of sets and functions, 15
Cat	(quasi-)category of categories and functors, 15
F_1 / F_2	comma category given by functors F_1 and F_2 , 16
A / \mathbb{K}	category of \mathbb{K} -objects under an object A of \mathbb{K} , 16
$\lfloor _ \rfloor_A$	forgetful functor $A / \mathbb{K} \rightarrow \mathbb{K}$, 16
h / \mathbb{K}	left-composition functor given by a morphism h in \mathbb{K} , 16
\mathbb{K}^{\rightarrow}	category of \mathbb{K} -arrows, 16
dom	domain functor, 17
cod	codomain functor, 17
$\mathbf{C}: \mathbb{I}^{\text{op}} \rightarrow \text{Cat}$	\mathbb{I} -indexed category, 17
$\mathbf{C}^{\#}$	Grothendieck category defined by $\mathbf{C}: \mathbb{I}^{\text{op}} \rightarrow \text{Cat}$, 17
$(_)^{\#}$	‘flattening’ functor, 17
$[_ \rightarrow \mathbb{K}]^{\#}$	category of functors into a category \mathbb{K} , 17
$[\mathbf{C} \rightarrow \mathbb{K}]$	functor category determined by \mathbf{C} and \mathbb{K} , 17
$U_$	left-composition functor given by a functor U , 17
$[G]$	right-composition indexed functor defined by G , 18
$[G]^{\#}$	functor between categories of functors determined by G , 18
$F \subseteq G$	(generalized) subfunctor relationship between F and G , 18
$\text{Sig}^{\mathcal{J}}$	category of signatures of an institution \mathcal{J} , 19
$\text{Sen}^{\mathcal{J}}$	sentence functor of an institution \mathcal{J} , 19
$\text{Mod}^{\mathcal{J}}$	model functor of an institution \mathcal{J} , 19
$\vDash_{\Sigma}^{\mathcal{J}}$	Σ -satisfaction relation of an institution \mathcal{J} , 19
$\varphi(_)$	sentence translation along a signature morphism φ , 19
$_ \dashv \varphi$	model reduction along a signature morphism φ , 19
$\vDash_{\Sigma}^{\mathcal{J}}$	semantic-entailment relation for an \mathcal{J} -signature Σ , 19
collns	category of institutions and institution comorphisms, 21
Room	category of rooms and corridors, 21

$\underline{\text{FOL}}_{\neq}^1$	single-sorted first-order logic without equality, 23
φ^{op}	translation of operation symbols along φ , 24
φ^{rel}	translation of relation symbols along φ , 24
T_F	set of F -terms (for $\underline{\text{FOL}}_{\neq}^1$), 24
φ^{tm}	translation of terms along φ , 24
$ M $	carrier set of a model M , 24
$\text{QF-}\underline{\text{FOL}}_{\neq}^1$	quantifier-free fragment of $\underline{\text{FOL}}_{\neq}^1$, 25
$\text{Subst}_{\langle F, P \rangle}$	category of first-order substitutions over $\langle F, P \rangle$, 26
$(\text{QF-}\underline{\text{FOL}}_{\neq}^1)_{\langle F, P \rangle}$	institution of $\langle F, P \rangle$ -substitutions, 27
$C \xleftarrow{X} H$	clause, 28
Subst	category of substitutions of a substitution system, 29
$(\text{QF-}\underline{\text{FOL}}_{\neq}^1)_{\langle F, P \rangle}$	substitution system of $\langle F, P \rangle$ -substitutions, 30
SubstSys	category of substitution systems, 32
Subst_{Σ}	category of Σ -substitutions, 32
G_{Σ}	room of ground Σ -sentences and models, 32
\mathcal{GS}_{Σ}	local substitution system determined by Σ , 32
$\text{Sen}_{\Sigma}(X)$	set of Σ -sentences over X , 33
$\text{Mod}_{\Sigma}(X)$	category of Σ -models over X , 33
$\vDash_{\Sigma, X}$	Σ -satisfaction between sentences and models over X , 33
$\alpha_{\Sigma, X}, X(_)$	extension of Σ -sentences over X , 33
$\beta_{\Sigma, X}, _ \uparrow_{\Sigma}$	reduction of Σ -models over X , 33
$\text{Sen}_{\Sigma}(\psi), \psi(_)$	translation of sentences along a Σ -substitution ψ , 33
$\text{Mod}_{\Sigma}(\psi), _ \uparrow_{\psi}$	reduction of models along a Σ -substitution ψ , 33
Ψ_{φ}	translation of variables along a signature morphism φ , 33
κ_{φ}	the corridor $\langle \text{Sen}(\varphi), \text{Mod}(\varphi) \rangle$, 33
$\alpha_{\varphi, X}$	translation of X -sentences along φ , 33
$\beta_{\varphi, X}$	reduction of X -models along φ , 33
$\tau_{\varphi, X}$	the corridor $\langle \alpha_{\varphi, X}, \beta_{\varphi, X} \rangle$, 33
$\text{QF-}\underline{\text{FOL}}_{\neq}^1$	generalized substitution system of the quantifier-free, single-sorted fragment of first-order logic without equality, 33
$\forall X \cdot \rho$	universal quantification of an X -sentence ρ , 34
$\exists X \cdot \rho$	existential quantification of an X -sentence ρ , 34
$\text{QSen}^{\mathcal{GS}}$	quantified-sentence functor associated with a generalized substitution system \mathcal{GS} , 34
\mathcal{GS}^{qs}	institution of quantified sentences over \mathcal{GS} , 36
$\vDash_{\Sigma}^{\text{qs}}$	the Σ -satisfaction relation of \mathcal{GS}^{qs} , 36
$\text{LSen}^{\mathcal{GS}}$	local-sentence functor associated with \mathcal{GS} , 37
$C_{\Sigma}(X)$	set of local X -clauses over a signature Σ , 37
$Q_{\Sigma}(X)$	set of local X -queries over Σ , 37

$\Vdash_{\Sigma, X}$	collection of goal-directed rules over Σ , 37
\mathcal{FOL}_{\neq}^1	relational first-order logic-programming framework, 38
$\forall C$	clause functor of a logic-programming framework, 38
$\exists Q$	query functor of a logic-programming framework, 38
\mathbb{LIP}	category of logic programs, 39
Sign	signature functor of a logic-programming language, 39
PMod	model functor of a logic-programming language, 39
Ax	clause functor of a logic-programming language, 39
$\langle\langle \Sigma, \Gamma \rangle\rangle$	logic program with signature Σ and set of clauses Γ , 39
$\vDash_{\Sigma}^{\text{lp}}$	Σ -entailment relation of a logic-programming language, 39
$\mathcal{F}^{\text{pres}}$	language of theory presentations over a framework \mathcal{F} , 39
$\overline{\vdash}_X \rho$	query, 43
$\longrightarrow_{\Gamma, \theta}$	one-step θ -derivation relation generated by Γ , 45
$\longrightarrow_{\Gamma, \theta}^*$	general θ -derivation relation generated by Γ , 46
$\exists Y \cdot \top$	trivial query, 46
$X(\Gamma)$	X -instance of a set of clauses Γ , 48
$\Vdash_{\Sigma, X}^*$	collection of extended goal-directed rules over Σ , 48
$\underline{\text{FOL}}_{=}$	many-sorted first-order equational logic, 53
$\text{QF-}\underline{\text{FOL}}_{=}$	quantifier-free fragment of $\underline{\text{FOL}}_{=}$, 53
φ^{st}	translation of sorts along φ , 53
$T_{F, s}$	set of F -terms of sort (or type) s , 53
M_s	carrier set of a sort s in a model M , 54
$\underline{\text{HNK}}$	higher-order logic with Henkin semantics, 54
$\text{QF-}\underline{\text{HNK}}$	quantifier-free fragment of $\underline{\text{HNK}}$, 54
\vec{S}	set of S -types, 54
φ^{type}	translation of types along φ , 54
$\text{Subst}_{\Sigma}^{\text{Q}}$	category of institution-independent Σ -substitutions, 56
$\mathcal{S}_{\Sigma}^{\text{Q}}$	local Σ -substitution system given by an institution \mathcal{J} , 56
$\Sigma(\chi)$	extension of a signature Σ along a morphism φ , 56
ι_{Q}	'domain' projection given by a category Q of arrows, 57
χ^{φ}	translation of a signature extension χ along φ , 58
φ^{χ}	extension of a signature morphism φ along χ , 58
M_{χ}	representation of a signature extension χ , 60
$U_{\varphi, \chi}$	reduction of homomorphisms given by a signature morphism φ and an extension χ of the domain signature of φ , 61
U_{ψ}	reduction of homomorphisms given by a substitution ψ , 61
R_{Σ}^{Q}	representation functor $\text{Subst}_{\Sigma}^{\text{Q}} \rightarrow \text{Mod}(\Sigma)$ corresponding to a quantification space Q and a signature Σ , 62
h_{ψ}	representation of a substitution ψ , 62

ψ^φ	translation of a substitution ψ along φ , 64
$\mathcal{S}\mathcal{J}$	generalized substitution system derived from \mathcal{J} , 67
$\text{QF-FOL}_=$	generalized substitution system of the quantifier-free fragment of many-sorted first-order equational logic, 67
QF-HNK	generalized substitution system of the quantifier-free fragment of higher-order logic with Henkin semantics, 67
$c[t]$	substitution of a term t in a context c , 67
$\text{FOL}_=$	first-order equational logic-programming framework, 68
HNK	higher-order equational logic-programming framework, 68
Orc	category of orchestrations of an orchestration scheme, 73
Spec	specification functor of an orchestration scheme, 73
Grc	category of ground orchestrations, 73
Prop	property functor of an orchestration scheme, 73
$\underline{\text{POA}}$	institution of preorder algebra, 74
SP^{pgm}	set of types (sorts) of executable expressions, 76
State	sort of program states, 76
Config	sort of program configurations, 76
$\langle _ \rangle, \langle _ , _ \rangle$	constructor operators for the sort Config , 76
Var	set of program variables, 76
$\text{pgm} : eXp$	program expression, 77
$\iota : [\rho, \rho']$	program specification, 78
$\underline{\text{LTL}}$	institution of linear temporal logic, 80
$\underline{\text{ALTL}}$	automata-based variant of the institution $\underline{\text{LTL}}$, 80
$\text{Inf}(\varrho)$	infinity set of a run ϱ , 81
$\lambda(i..)$	the suffix of a trace λ that starts at $\lambda(i)$, 81
M^-	set of published messages of a port M , 84
M^+	set of delivered messages of a port M , 84
A_{M^-}	set of publication actions of a port M , 84
A_{M^+}	set of delivery actions of a port M , 84
A_M	set of actions defined by a port or by a channel, 84
M_x	port associated with a point x , 84
A_M^-	set of publication actions of a channel $\langle M, \Lambda \rangle$, 85
A_M^+	set of delivery actions of a channel $\langle M, \Lambda \rangle$, 85
μ_x	partial attachment injection of a point x to a channel, 86
γ_e	set of vertices incident with a hyperedge e , 88
θ_x^{pt}	polarity-preserving injection given by a morphism of ARNs θ and a point x of its domain network, 89
\mathfrak{N}_x	sub- ARN of \mathfrak{N} defined by a point x , 91
$D_{\mathfrak{N}}$	diagram of an ARN \mathfrak{N} , 92

$\xi: D_{\mathfrak{N}} \Rightarrow A_{\mathfrak{N}}$	signature of an ARN \mathfrak{N} , 92
$\Lambda_x, \Lambda_{\mathfrak{G}_x} \uparrow_{\xi_x}$	automaton observed at a point x of a network \mathfrak{G} , 92
$\Lambda_e^{\mathfrak{G}_x}$	cofree expansion of Λ_e (the automaton associated with a hyperedge e of an ARN \mathfrak{G}_x) along ξ_e , 93
$\forall X \cdot C \leftarrow H$	definite clause, 95
AFOL_{\neq}^1	atomic fragment of relational first-order logic, 95
OS	category of orchestration schemes, 96
OrcScheme	logical system of orchestration schemes, 96
$\langle \emptyset, \mathbb{1}, \emptyset \rangle$	initial room, 99
$\text{SS}(\mathcal{J})$	substitution system over $\langle \emptyset, \mathbb{1}, \emptyset \rangle$ generated by \mathcal{J} , 100
$\text{Ins}(\mathcal{O})$	institution generated by an orchestration scheme \mathcal{O} , 100
$P \xleftarrow{\circ} R$	(definite) service-oriented clause, 103
$\vdash_{\circ} Q$	service-oriented query, 104
$SP_1 \sqsubseteq SP_2$	refinement of program specifications, 105
$\mathcal{F}^{\text{struc}}$	language of program modules over a framework \mathcal{F} , 113
$P_1 \cup P_2$	union of program modules P_1 and P_2 , 113
translate	translation operator (for program modules), 113
free	free-semantics operator, 113

INDEX OF CONCEPTS

- action, temporal, 80
 - delivery action, 84
 - publication action, 84
- answer to a query
 - computed answer, 47
 - correct answer, *see* solution
- asynchronous relational network
 - (ARN), 88
 - defined by a point, 91
- category
 - arrow, 16
 - comma, 16
 - functor, 17
 - of functors, 17
 - indexed, 17
 - of institutions, 21
 - of objects under A , 16
 - of representable substitutions, 63
- channel, 85
- clause, 38
 - definite, 95
 - identity, 47
 - instance, *see* instance of a set of clauses
 - local clause, 37
 - equational, 67
 - relational, 38
 - of a logic program, 39
 - service-oriented, definite, 103
 - correct, 103
 - provides-interface, 103
 - requires-interface, 103
- comorphism of institutions, 20
- connection, 86
 - attachment injection, 86
- conservative
 - morphism of programs, 115
 - signature of variables, 40
- context, 67
- corridor, 21
- dependency, of a point, 91
- diagram, of an ARN, 92
- expansion of a model, 19
- functor
 - codomain functor, 17
 - domain functor, 17
 - indexed, 17
 - left-composition, 16, 17
 - local-sentence functor, 37
 - model functor, 19
 - quantified-sentence functor, 34
 - reduct functor, 19
 - sentence functor, 19
- generalized substitution system, 32
- goal-directed derivation, 46
- goal-directed rule, 37
 - equational, *see* paramodulation extended, 48
 - relational, 38
- Grothendieck

- category, 17
- construction, 17
- homomorphism, *see* morphism
 - of models
 - of algebras, 54
 - of first-order models, 25
 - of Henkin models, 55
 - of hypergraphs, 89
 - of Muller automata, 81
- hyperedge, 87
 - communication, 88
 - computation, 88
- hypergraph, labelled, 87
 - edge-bipartite, 88
- instance of a set of clauses, 48
- institution, 19
 - of equational logic, 53
 - of extended models, 99
 - of first-order logic, 23
 - of first-order substitutions, 27
 - of higher-order logic, 54
 - of linear temporal logic, 80
 - of preorder algebra, 74
 - of quantified sentences, 36
- institution representation, *see*
 - comorphism of institutions
- lifting substitutions, 115
- logic programming
 - framework, 37
 - language, 38
 - query-complete, *see* query-completeness
- logic programs
 - query-complete, *see* query-completeness
 - structured, 113
 - as theory presentations, 39
- message, 84, 85
 - delivered, 84
 - published, 84
- model, 19
 - algebra, 54
 - amalgamation, *see* model-amalgamation property
 - first-order, 24
 - ground model, 29
 - Henkin, 55
 - of a logic program, 39
 - Muller automaton, 80
 - observed at a point, 92
 - preorder algebra, 76
 - projective, 69
 - reachable, 41
 - of a signature of variables, 29
 - trace, 81
 - accepted, 81
- model-amalgamation property, 35
- model-amalgamation square, 35
- morphism
 - of ARNS, 89
 - of first-order signatures, 24
 - of institutions, 20, 98
 - of logic programs, 39
 - simple, 117
 - of many-sorted signatures, 53
 - of models, 19
 - of signatures
 - representable, 60
 - of orchestration schemes, 97
 - of orchestrations, 73
 - of program expressions, 77
 - signature extension, 56
 - of signatures, 19
 - derived, 97
 - of structured programs, 113

- of substitution systems, 31
 - of theory presentations, 39
- orchestration, 73
 - ARN, 90
 - ground, 92
 - ground, 73
 - program expression, 77
 - ground, 78
- orchestration scheme, 73
- paramodulation, 68
- plain map, *see* comorphism of institutions
- point, 88
 - interaction point, 84, 89
 - provides-point, 89
 - requires-point, 89
- port, 84
- preservation
 - of computed answers, 118
 - of queries, 114
 - of satisfaction, 42
 - of solutions, 114
- process, 84
- quantification space, 58
 - adequate, 58
 - first-order, 58
 - higher-order, 58
- query, 38
 - derived, *see* resolution
 - local query, 37
 - equational, 68
 - relational, 38
 - service-oriented, 104
 - requires-interface, 104
 - satisfiable, 104
 - trivial, 46
- query-completeness, 50
- reduct of a model, 19
- reduction
 - of first-order models, 25
 - of ground models, 31
 - of models, *see* model functor
 - of Muller automata, 81
- reflection
 - of computed answers, 119
 - of final (goal-directed) rules, 118
 - of satisfaction, 80
 - of solutions, 116
- representation
 - of a signature morphism, 60
- resolution
 - service-oriented, 106
- resolution, 44
 - completeness, 50
 - soundness, 47
- room, 21
- run, on a trace, 81
 - successful, 81
- satisfaction condition
 - for an institution, 19
 - for an institution comorphism, 21
- satisfaction relation, 19
 - equational, 54
 - first-order, 25
 - higher-order, 55
 - linear temporal, 81
 - for preorder algebra, 76
 - of a signature of variables, 30
- semantic consequence, 19
- sentence, 19
 - basic, 70
 - equational atom, 53, 55, 76
 - existentially quantified, 34

- first-order, 24
- first-order equational, 53
- ground sentence, 29
- higher-order, 55
- linear temporal, 80
- local sentence, 37
- preorder atom, 76
- relational atom, 24
- of a signature of variables, 29
- unification, 44
- universally quantified, 34
- service
 - binding, 106
 - discovery, 105
 - property, 73
 - of an ARN, 93
 - of a program expression, 78
 - specification, 73
 - of an ARN, 90
 - of a program expression, 78
 - refinement of, 105
- signature, 19
 - of an ARN, 92
 - first-order, 23
 - of first-order variables, 25, 56
 - higher-order, 54
 - of higher-order variables, 57
 - linear temporal, 80
 - many-sorted, algebraic, 53
 - of variables, 29
- signature functor
 - of a comorphism, 20
 - of a logic-programming language, 39
- solution to a query, 40
- structuring operator, 112
- subfunctor, 18
 - generalized, 18
- substitution, 29
 - compatible, 66
 - computed, 44
 - first-order, 26, 56
 - higher-order, 57
 - institution-independent, 55
 - representable, 63
 - of a term in a context, 67
- substitution system, 29
 - generalized, 32
- term
 - first-order, 24
 - higher-order, 55
 - many-sorted, 53
 - over a set of variables, 25
- translation
 - of ground sentences, 31
 - of local sentences, 37
 - of relational first-order sentences, 24
 - of sentences, *see* sentence functor
 - of signatures of variables, 31
 - of terms, 24
- twisted relation, *see* room
- unification, *see* sentence unification
 - service-oriented, 106
- unifier, 44
 - instance, 44
 - most general, 44
- variable
 - first-order, 24, 56
 - higher-order, 57
 - requires-point, 89