Service-oriented logic programming

Ţuţu, Ionuţ; Fiadeiro, Jośe Luiz

Published in:
Logical Methods in Computer Science

DOI:
10.2168/LMCS-11(3:3)2015

Publication date:
2015

Licence:
CC BY

Document Version
Publisher's PDF, also known as Version of record

Logical Methods in Computer Science
Vol. 11(3:3)2015, pp. 1–37
www.lmcs-online.org

Submitted Feb. 27, 2014
Published Aug. 13, 2015

SERVICE-ORIENTED LOGIC PROGRAMMING *

IONUŢ ŢUŢU [a] AND JOSÉ LUIZ FIADEIRO [b]

[a] Department of Computer Science, Royal Holloway University of London
Institute of Mathematics of the Romanian Academy, Research group of the project ID-3-0439
e-mail address: ittutu@gmail.com

[b] Department of Computer Science, Royal Holloway University of London
e-mail address: jose.fiadeiro@rhul.ac.uk

Abstract. We develop formal foundations for notions and mechanisms needed to support service-oriented computing. Our work builds 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. We show how the denotational and the operational semantics specific to conventional logic programming can be generalized using the theory of institutions to address both static and dynamic aspects of service-oriented computing. 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.

1. Introduction

Service-Oriented Computing. 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 [ACKM04] or Grid computing [FK04], research into its formal foundations has lagged somewhat behind, partly because of our lack of understanding of (or agreement on) what is really new about the paradigm, especially in relation to distributed computing in general (see, for example, [Vog03]).

2012 ACM CCS: [**Theory of computation**]: Logic — Constraint and logic programming; Semantics and reasoning — Program reasoning — Program specifications; [**Information systems**]: World Wide Web — Web services — Service discovery and interfaces.

Key words and phrases: Logic programming, Institution theory, Service-oriented computing, Orchestration schemes, Service discovery and binding.

* A preliminary version of this work was presented at CALCO 2013 [ŢF13].

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-11(3:3)2015

It is fair to say that significant advances have been made towards formalizing new forms of distributed computation that have arisen around the notion of service (e.g. choreography [SBFZ07]), 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. 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, we have investigated 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 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, FL13b]). In the present paper, we take further this research to address the operational aspects behind *dynamic* discovery and binding, 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 challenge here 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 to be discovered, and (c) the way the discovery of services and the binding of their orchestrations to client applications can be expressed in logical/algebraic terms.

Logic Programming. The approach that we propose to develop to meet this challenge builds on the relational variant of (Horn-clause) logic programming – the paradigm that epitomizes the integration of declarative and operational aspects of logic. In conventional logic programming, clauses have a declarative semantics as universally quantified implications that express relationships over a domain (the Herbrand universe), and an operational semantics that derives from resolution and term unification: definite clauses (provided by a given logic program) are used to resolve logic-programming queries (expressed as existentially quantified conjunctions) by generating new queries and, through term unification, computing partial answers as substitutions for the variables of the original query.

In a nutshell, the analogy between service-oriented computing and conventional logic programming that we propose to systematically examine in this paper unfolds as follows:

- The Herbrand universe consists of those 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 that need to be discovered and to the actual services that are made available by orchestrations.
- Service clauses express conditional 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 orchestrations of services that an application requires in order to fulfil its goal – what we describe in [FLB11] as activity modules.
- Logic programs define service repositories as collections of service modules.
- Resolution and term unification account for service discovery by matching required properties with provided ones and the binding of required with provided services.

The structure of the paper. Our research into the logic-programming semantics of service-oriented computing is organized in two parts. In Section 2 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 [FL13a]. 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 3, 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 recall from [ŢF15] the algebraic structures that underlie institution-independent logic programming, in particular the substitution systems that are characteristic of relational logic programming, and prove that the resulting logic of orchestration schemes constitutes a *generalized substitution system*. This result is central to our work, not only because it provides the declarative semantics of our approach to service-oriented computing, but also because it gives a definite mathematical foundation to the analogy between service-oriented computing and conventional logic programming outlined above. Building on these results, we show how clauses, queries, unification and resolution can be defined over the generalized substitution system of orchestration schemes, providing in this way the corresponding operational semantics of service-oriented computing.

The work presented herein continues our investigation on logic-independent foundations of logic programming reported in [ŢF15]. As such, it is based on the theory of institutions of Goguen and Burstall [GB92]; although familiarity with the institution-independent presentation of logic programming is not essential, some knowledge of basic notions of institution theory such as institution, (co)morphism of institutions, and also of the description of institutions as functors into the category of rooms [Dia08, ST11] is presumed.

2. Orchestration Schemes

The first step in the development of the particular variant of logic programming that we consider in this paper 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 (such as 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 [FLB07, FLB11, FL13b]);

second, that the discovery of a service to be bound to a given client application can be formalized in terms of logical specifications of required and provided properties, ensuring that the specification of the properties offered by the service provider refines the specification of the properties requested by the client application. To this end, we explore the model-theoretic notion of refinement advanced in [ST88], except that, in the present setting, the structures over which specifications are evaluated are morphisms into ground orchestrations, i.e. 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 2.1 (Orchestration scheme)**.** An *orchestration scheme* is a quadruple $\langle \mathbb{O}\mathrm{rc}, \mathrm{Spec}, \mathbb{G}\mathrm{rc}, \mathrm{Prop}\rangle$ consisting of

- a category $\mathbb{O}\mathrm{rc}$ of *orchestrations* and *orchestration morphisms*,
- a functor $\mathrm{Spec}\colon \mathbb{O}\mathrm{rc} \to \mathbb{S}\mathrm{et}$ that defines a set $\mathrm{Spec}(\mathfrak{o})$ of *service specifications* over $\mathfrak{o}$ for every orchestration $\mathfrak{o}$,
- a full subcategory $\mathbb{G}\mathrm{rc} \subseteq \mathbb{O}\mathrm{rc}$ of *ground orchestrations*, and
- a functor $\mathrm{Prop}\colon \mathbb{G}\mathrm{rc} \to \mathbb{S}\mathrm{et}$ that defines a natural subset $\mathrm{Prop}(\mathfrak{g}) \subseteq \mathrm{Spec}(\mathfrak{g})$[1] of *properties* of $\mathfrak{g}$ (specifications that are guaranteed to hold when evaluated over $\mathfrak{g}$) for every ground orchestration $\mathfrak{g}$.

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 [FL13a], which emphasizes the role of services as an interface mechanism for software components that can be composed through run-time discovery and binding.

2.1. **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 [Hoa69] as defining 'clauses' in the sense of logic programming, where unification is controlled through the refinement of pre/post-conditions as specifications of provided/required services, and resolution binds program statements (terms) to variables in program expressions. In Figure 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 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 Subsection 3.3.

[1]By describing the set $\mathrm{Prop}(\mathfrak{g})$ as a *natural subset* of $\mathrm{Spec}(\mathfrak{g})$ we mean that the family of inclusions $(\mathrm{Prop}(\mathfrak{g}) \subseteq \mathrm{Spec}(\mathfrak{g}))_{\mathfrak{g}\in|\mathbb{G}\mathrm{rc}|}$ defines a natural transformation from Prop to $(\mathbb{G}\mathrm{rc} \subseteq \mathbb{O}\mathrm{rc})$;Spec.

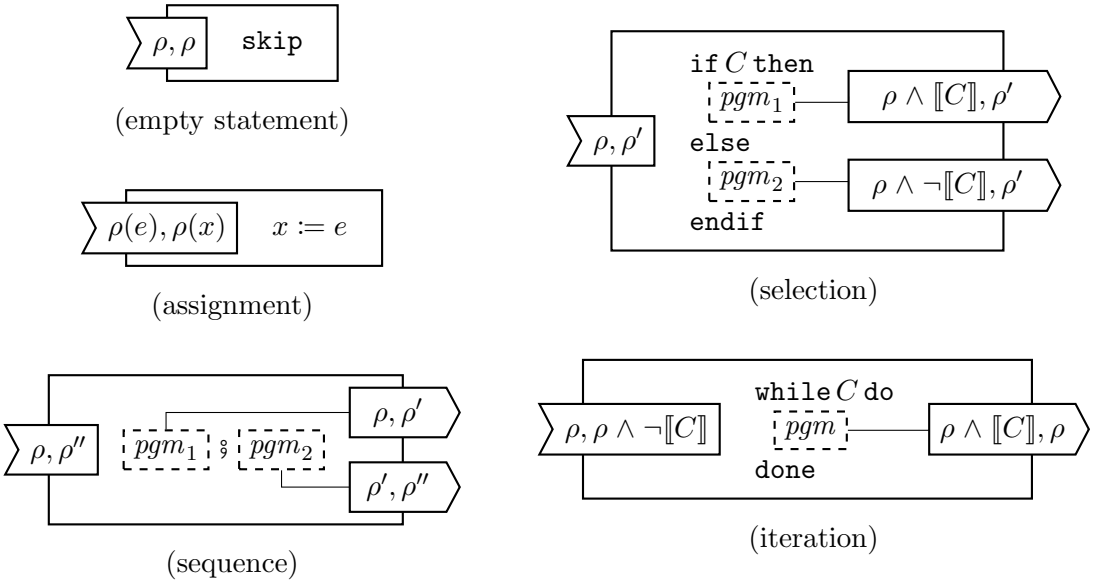


Figure 1: Program modules

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 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* $l \to r$, 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 $S^{\mathrm{pgm}} \subset S$ corresponding to the types of executable expressions supported by the language, and sorts *State*, *Config* $\in S \setminus S^{\mathrm{pgm}}$ capturing the states of the programs and the various configurations that may arise upon their execution, respectively;
- operation symbols $\langle \_ \rangle \colon \mathit{State} \to \mathit{Config}$ and $\langle \_, \_ \rangle \colon \mathit{eXp}\ \mathit{State} \to \mathit{Config}$ for sorts $\mathit{eXp} \in S^{\mathrm{pgm}}$, which we regard as constructor operators for the sort *Config*;
- a (sortwise infinite) S^{pgm}-indexed set *Var* of program variables, and state variables $st, st' \colon \mathit{State}$, 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 2.2. The premises that we consider within this subsection are weak enough to allow the proposed algebraic framework to accommodate a wide variety of programming

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

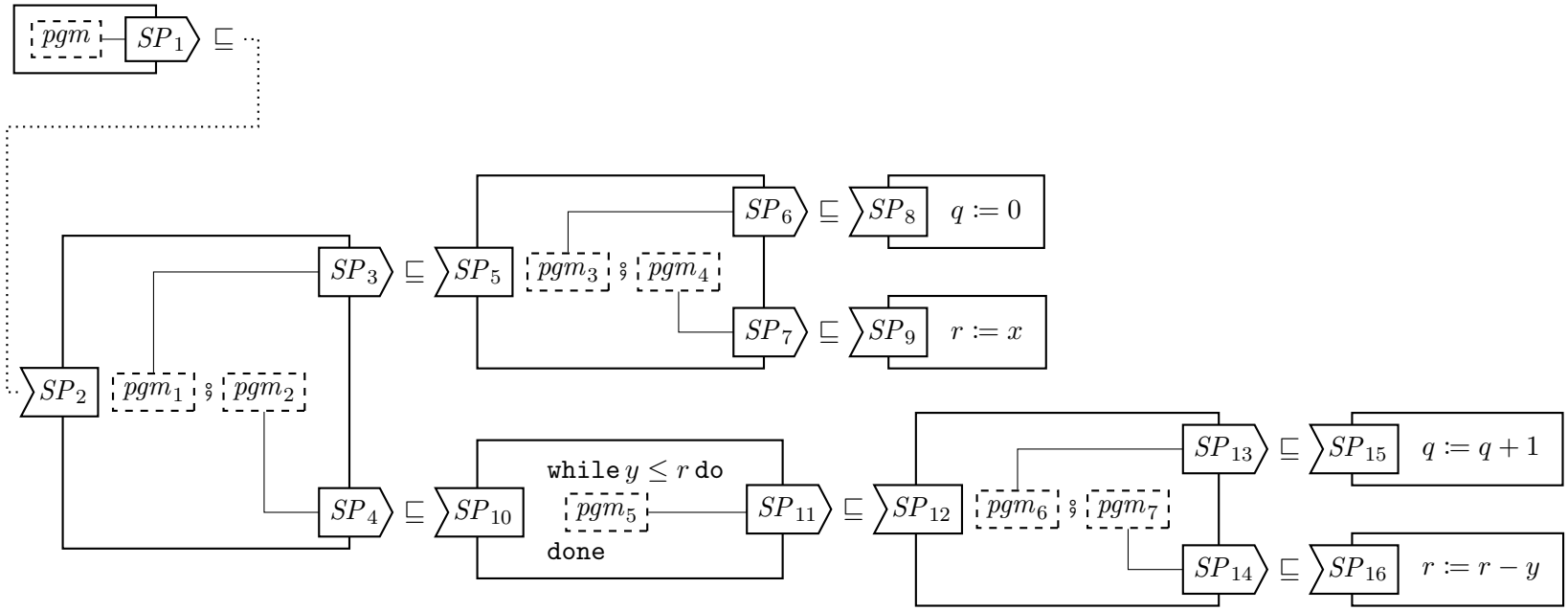


Figure 2: The derivation of a program that computes the quotient q and the remainder r obtained from the division of x by y

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

languages. For instance, the program expressions underlying the modules depicted in Figure 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) `skip`: $\to$ `Pgm`,

(assignment) _ := _: `Id AExp` $\to$ `Pgm`,

(sequence) _ ; _: `Pgm Pgm` $\to$ `Pgm`,

(selection) `if _then _else _endif`: `Cond Pgm Pgm` $\to$ `Pgm`,

(iteration) `while _do _done`: `Cond Pgm` $\to$ `Pgm`.

To simplify our presentation, we omit the details associated with the sorts `Id` of identifiers, `AExp` of arithmetic expressions and `Cond` of conditions; we also tacitly assume that the signature under consideration declares the usual operation symbols associated with the names of identifiers, the addition, subtraction and multiplication of arithmetic expressions, and with the atoms and Boolean connectives specific to conditions. Moreover, we assume the essential sorts `State` and `Config` to be defined, as well as the operation symbols $\langle \_ \rangle$ and $\langle \_, \_ \rangle$.

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

Orchestrations. The *orchestrations* are program expressions, that is $\langle S, F \cup Var \rangle$-terms $pgm \colon eXp$, 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 $pgm_1 \colon eXp_1$ and $pgm_2 \colon eXp_2$ consists of

- a substitution $\psi \colon \mathrm{var}(pgm_1) \to \mathrm{var}(pgm_2)$, mapping the variables that occur in pgm_1 to program expressions defined over the variables of pgm_2, together with
- a position π in pgm_2, i.e. a sequence of natural numbers that precisely identifies a particular occurrence of a subterm $pgm_2{\restriction_\pi}$ of pgm_2,

such that $\psi^{\mathrm{tm}}(pgm_1) = pgm_2{\restriction_\pi}$.[3] Their *composition* is defined componentwise, in a way that ensures the commutativity of the following diagram.

$$pgm_1 \colon eXp_1 \xrightarrow{\langle \psi_1, \pi_1 \rangle} pgm_2 \colon eXp_2 \xrightarrow{\langle \psi_2, \pi_2 \rangle} pgm_3 \colon eXp_3 \qquad \langle \psi_1 ; \psi_2, \pi_2 \cdot \pi_1 \rangle$$

Specifications. For each program expression $pgm \colon eXp$, a *(program) specification* is a triple of the form $\iota \colon [\rho, \rho']$, where ι is a position in pgm indicating the 'subprogram' of pgm whose behaviour is being analysed,[4] and ρ and ρ' are *pre-* and *post-conditions* associated with $pgm{\restriction_\iota}$, formalized as (quantifier-free) POA-sentences over the signature $\langle S, F \cup \{ st \colon State \} \rangle$. The intuitive interpretation is the usual one:

> Whenever the program $pgm{\restriction_\iota}$ 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'.

[3] Here, we let ψ^{tm} denote the canonical extension of the substitution ψ from variables to terms.

[4] The first component of specifications may be encountered in the literature (e.g. in [Mor94]) with a different meaning: the set of identifiers whose values may change during the execution of the program.

Note, however, that specifications cannot be evaluated over arbitrary program expressions because, due to the presence of program variables (from Var), some of the programs may not support a well-defined notion of execution. We will address this aspect in Section 3 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 $\iota\colon [\rho, \rho']$ of $pgm_1\colon eXp_1$ along a morphism $\langle \psi, \pi\rangle\colon (pgm_1\colon eXp_1) \to (pgm_2\colon eXp_2)$ is defined as the specification $(\pi \cdot \iota)\colon [\psi(\rho), \psi(\rho')]$ of $pgm_2\colon eXp_2$.

Ground orchestrations and properties. As expected, *ground program expressions* are just program expressions that do not contain variables: $\langle S, F\rangle$-terms $pgm\colon eXp$ 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 $\iota\colon [\rho, \rho']$ is a *property* of a ground program expression $pgm\colon eXp$ if and only if the following satisfaction condition holds for the preordered algebra A:

$$A \models^{\underline{\mathrm{POA}}} \forall \{st, st'\colon State\} \cdot (\rho(st) \wedge \langle pgm\restriction_\iota, st\rangle \to \langle st'\rangle) \Rightarrow \rho'(st').$$

To keep the notation simple and, at the same time, emphasize the roles of st and st', we used $\rho(st)$ in the above $\underline{\mathrm{POA}}$-sentence as another name for ρ, while $\rho'(st')$ is the sentence derived from ρ' by replacing the variable st with st'.[5] The same notational convention is used in Figure 1 to represent the specification attached to the assignment expression. In that case, ρ is assumed to be a sentence defined not only over $st\colon \texttt{State}$, but also over a variable $v\colon \texttt{AExp}$; 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 1 (and also in Figure 2) is $[\![C]\!]$, where C is a term of sort `Cond`; this follows Iverson's convention (see [Ive62], and also [GKP94]), and corresponds to an atomic $\underline{\mathrm{POA}}$-sentence that captures the semantics of the condition C.

We conclude the presentation of orchestrations as program expressions with Proposition 2.3 below, which guarantees that properties form natural subsets of the sets of specifications; in other words, the morphisms of ground programs preserve properties.

Proposition 2.3. *Let $\langle \psi, \pi\rangle\colon (pgm_1\colon eXp_1) \to (pgm_2\colon eXp_2)$ be a morphism of ground programs. For every property $\iota\colon [\rho, \rho']$ of $pgm_1\colon eXp_1$, the specification* $\mathrm{Spec}(\psi, \pi)(\iota\colon [\rho, \rho'])$ *is a property of $pgm_2\colon eXp_2$.*

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

$$A \models^{\underline{\mathrm{POA}}} \forall \{st, st'\colon State\} \cdot (\underbrace{\psi(\rho)(st)}_{\rho(st)} \wedge \langle \underbrace{pgm_2\restriction_{\pi\cdot\iota}}_{pgm_1\restriction_\iota}, st\rangle \to \langle st'\rangle) \Rightarrow \underbrace{\psi(\rho')(st')}_{\rho'(st')}.$$

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 $pgm_2\restriction_{\pi\cdot\iota} = pgm_2\restriction_\pi\restriction_\iota = \psi^{\mathrm{tm}}(pgm_1)\restriction_\iota = pgm_1\restriction_\iota$, from which we immediately deduce that both the evaluation of $\iota\colon [\rho, \rho']$ in $pgm_1\colon eXp_1$ and that of $\mathrm{Spec}(\psi, \pi)(\iota\colon [\rho, \rho'])$ in $pgm_2\colon eXp_2$ correspond to the satisfaction by A of the same $\underline{\mathrm{POA}}$-sentence. □

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

2.2. **Asynchronous Relational Networks.** Asynchronous relational networks as developed in [FL13a] 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 interactions that are involved in such systems. In this paper, 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 within this context depend upon elements of linear temporal logic, and are introduced through dedicated 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 in which 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;
4. all model homomorphisms reflect the satisfaction of sentences.

In addition to the above requirements, we implicitly assume, as is often done in institutions (see, for example, [Dia08] and [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[6] and under the temporal operators *next* ($\bigcirc\_$) and *until* ($\_\,\mathcal{U}\,\_$). As usual, the derived temporal sentences $\Diamond\rho$ and $\Box\rho$ stand for $\mathit{true}\,\mathcal{U}\,\rho$ and $\neg(\mathit{true}\,\mathcal{U}\,\neg\rho)$, respectively.

The semantics of aLTL is defined over (non-deterministic finite-state) 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 of a (finite) set Q of *states*, an *alphabet* $\mathcal{P}(A)$, a *transition relation* $\Delta \subseteq Q \times \mathcal{P}(A) \times Q$, a subset $I \subseteq Q$ of *initial states*, and a subset $\mathcal{F} \subseteq \mathcal{P}(Q)$ of (non-empty) *final-state sets*.

The satisfaction relation is based on that of LTL: an automaton Λ satisfies a sentence ρ if and only if every trace accepted by Λ satisfies ρ in the sense of LTL. 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, i.e. the set of states that occur infinitely often in ϱ, denoted $\mathrm{Inf}(\varrho)$, is a member of $\mathcal{F}$.

[6]For convenience, we assume that disjunctions, denoted $\bigvee E$, and conjunctions, denoted $\bigwedge E$, are defined over arbitrary finite sets of sentences E, and we abbreviate $\bigwedge\{\rho_1, \rho_2\}$ as $\rho_1 \wedge \rho_2$ and $\bigwedge\emptyset$ as *true*.

Then a trace λ is *accepted* by Λ if and only 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:

$\lambda \vDash a$ if and only if $a \in \lambda(0)$,

$\lambda \vDash \neg\rho$ if and only if $\lambda \nvDash \rho$,

$\lambda \vDash \bigvee E$ if and only if $\lambda \vDash \rho$ for some $\rho \in E$,

$\lambda \vDash \bigcirc\rho$ if and only if $\lambda(1..) \vDash \rho$, and

$\lambda \vDash \rho_1 \,\mathcal{U}\, \rho_2$ if and only if $\lambda(i..) \vDash \rho_2$ for some $i \in \omega$, and $\lambda(j..) \vDash \rho_1$ for all $j < i$,

where a is an action in A, ρ, ρ_1 and ρ_2 are A-sentences, and E is a 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 $\underline{\text{ALTL}}$, as it corresponds to a well-known result about the existence of small colimits in $\mathbb{S}$et. In order to check that the remaining three properties hold as well, let us first recall that a *homomorphism* $h\colon \Lambda_1 \to \Lambda_2$ 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 $h\colon Q_1 \to Q_2$ 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 $\sigma\colon A \to A'$, i.e. for any signature morphism, and any Muller automaton $\Lambda' = \langle Q', \mathcal{P}(A'), \Delta', I', \mathcal{F}'\rangle$, the *reduct* $\Lambda'{\restriction_\sigma}$ is the automaton $\langle Q', \mathcal{P}(A), \Delta'{\restriction_\sigma}, I', \mathcal{F}'\rangle$ with the same states, initial states and final-state sets as Λ', and with the transition relation given by $\Delta'{\restriction_\sigma} = \{(p', \sigma^{-1}(\alpha'), q') \mid (p', \alpha', q') \in \Delta'\}$.

The following results enable us to use the institution $\underline{\text{ALTL}}$ as a foundation for the subsequent development of asynchronous relational networks. In particular, Proposition 2.4 ensures the existence of cofree Muller automata along signature morphisms; Proposition 2.5 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, i.e. regular ω-languages, are closed under intersection; and finally, Proposition 2.6 guarantees that all model homomorphisms reflect the satisfaction of temporal sentences.

Proposition 2.4. *For every morphism of* $\underline{\text{ALTL}}$*-signatures* $\sigma\colon A \to A'$*, the reduct functor* $\_{\restriction_\sigma}\colon \text{Mod}^{\underline{\text{ALTL}}}(A') \to \text{Mod}^{\underline{\text{ALTL}}}(A)$ *admits a right adjoint, which we denote by* $(\_)^\sigma$*.*

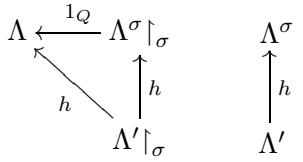
Proof. According to a general result about adjoints, it suffices to show that for any automaton Λ over the alphabet $\mathcal{P}(A)$ there exists a universal arrow from $\_{\restriction_\sigma}$ to Λ.

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 $\Lambda^\sigma{\restriction_\sigma} \to \Lambda$: for any transition $(p, \alpha, q) \in \Delta^\sigma{\restriction_\sigma}$, by the definition of the reduct functor $\_{\restriction_\sigma}$, there exists a set $\alpha' \subseteq A'$ 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 $(p, \alpha, q) \in \Delta$.



Let us now assume that $h\colon \Lambda'{\restriction_\sigma} \to \Lambda$ is another homomorphism of automata, with $\Lambda' = \langle Q', \mathcal{P}(A'), \Delta', I', \mathcal{F}' \rangle$. Then for any transition $(p', \alpha', q') \in \Delta'$, by the definition of the functor $\_{\restriction_\sigma}$, we have $(p', \sigma^{-1}(\alpha'), q') \in \Delta'{\restriction_\sigma}$. Based on the homomorphism property of h, it follows that $(h(p'), \sigma^{-1}(\alpha'), h(q')) \in \Delta$, which 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 $\Lambda' \to \Lambda^\sigma$. Even more, it is obviously the unique homomorphism $\Lambda' \to \Lambda^\sigma$ (in the category of automata over $\mathcal{P}(A')$) such that $h \fatsemi 1_Q = h$ in the category of automata over $\mathcal{P}(A)$. □

Proposition 2.5. *For any set of actions A, the category* $\mathrm{Mod}^{\underline{\text{ALTL}}}(A)$ *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 $\langle Q_i, \mathcal{P}(A), \Delta_i, I_i, \mathcal{F}_i \rangle$. We define the automaton $\Lambda = \langle Q, \mathcal{P}(A), \Delta, I, \mathcal{F} \rangle$ by

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

where the functions $\pi_i\colon Q \to Q_i$ 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 $\Lambda \to \Lambda_i$. Even more, one can easily see that for any other family of homomorphisms $(h_i\colon \Lambda' \to \Lambda_i)_{i \in J}$, with $\Lambda' = \langle Q', \mathcal{P}(A'), \Delta', I', \mathcal{F}' \rangle$, the unique map $h\colon Q' \to Q$ such that $h \fatsemi \pi_i = h_i$ for all $i \in J$ defines a homomorphism of automata as well. Therefore, the automaton Λ and the projections $(\pi_i)_{i \in J}$ form the product of $(\Lambda_i)_{i \in J}$. □

Proposition 2.6. *Let $h\colon \Lambda_1 \to \Lambda_2$ be a homomorphism between automata 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 $\Lambda_i = \langle Q_i, \mathcal{P}(A), \Delta_i, I_i, \mathcal{F}_i \rangle$, for $i \in \{1, 2\}$. Since the map $h\colon Q_1 \to Q_2$ defines a homomorphism of automata, for every successful run $\varrho \in Q_1^\omega$ of Λ_1 on a trace $\lambda \in \mathcal{P}(A)^\omega$, the composition $\varrho \fatsemi h$ yields a successful run of Λ_2 on λ. As a result, 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. □

Service Components. Following [FL13a], 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 2.7 (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* $A_M = A_{M^-} \cup A_{M^+}$, 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 2.8 (Process)**.** A *process* is a triple $\langle X, (M_x)_{x\in X}, \Lambda\rangle$ that 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 the alphabet $\mathcal{P}(A_M)$, where M is the port given by

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

Example 2.9. In Figure 3 we depict a process 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, named JP_1 and JP_2. The first port is used to communicate with potential client processes – 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.

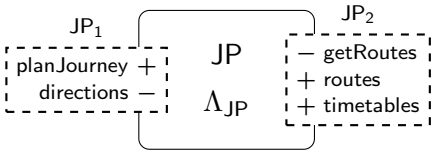


Figure 3: The process JP

The behaviour of JP is given by the Muller automaton depicted in Figure 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, and, once it receives both, it compiles the directions and replies to the client.

Remark 2.10. To generalize Definition 2.8 to an arbitrary institution (subject to the four technical assumptions listed at the beginning of the subsection), we first observe that every polarity-preserving map θ between ports M and M' defines a function $A_\theta \colon A_M \to A_{M'}$, i.e.

[7] 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.

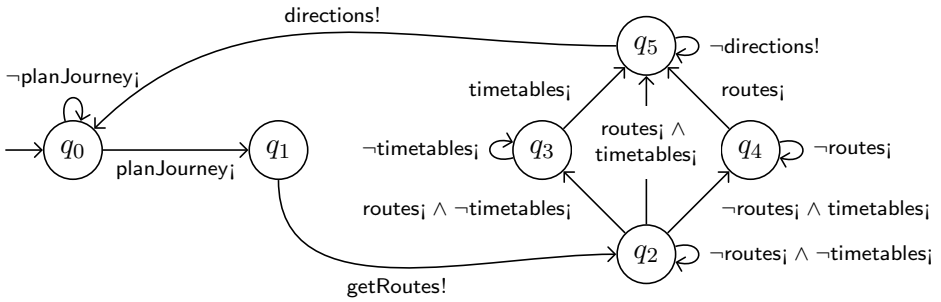


Figure 4: The automaton Λ_{JP}[7]

a morphism of ALTL-signatures, usually denoted simply by θ, that maps every publication action $m!$ to $\theta(m)!$ and every delivery action $m¡$ to $\theta(m)¡$. Moreover, for any process $\langle X, (M_x)_{x\in X}, \Lambda\rangle$, the injections $(x.\_\colon A_{M_x} \to A_M)_{x\in X}$ define a coproduct in the category of ALTL-signatures. This allows us to introduce an abstract notion of process as a triple $\langle X, (\iota_x\colon \Sigma_x \to \Sigma)_{x\in X}, \Lambda\rangle$ 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 $\iota_x\colon \Sigma_x \to \Sigma$ for $x \in X$ (usually defining a coproduct), and a model Λ of Σ.

Processes communicate by transmitting messages through channels. As in [BZ83, FL13a], channels are bidirectional: they may transmit both incoming and outgoing messages.

Definition 2.11 (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 $A_M^+ = \{m¡ \mid m \in M\}$.

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.

Definition 2.12 (Connection)**.** A *connection* $\langle M, \Lambda, (\mu_x\colon M \rightharpoonup M_x)_{x\in X}\rangle$ between the ports $(M_x)_{x\in X}$ consists of a channel $\langle M, \Lambda\rangle$ and a (finite) family of partial *attachment injections* $(\mu_x\colon M \rightharpoonup M_x)_{x\in X}$ such that $M = \bigcup_{x\in X} \mathrm{dom}(\mu_x)$ 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 [FL13a] 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 [FL13a].

Example 2.13. In order to illustrate how the process JP can send or receive messages, we consider the connection C depicted in Figure 5 that moderates the flow of messages between the port named JP_2 and two other ports, named R_1 and R_2.

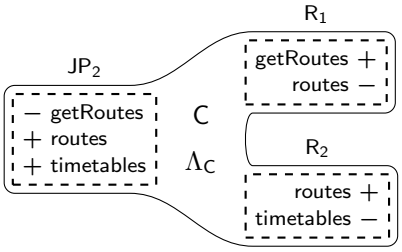


Figure 5: The Journey Planner's connection

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 6, and whose sets of states are all marked as final.

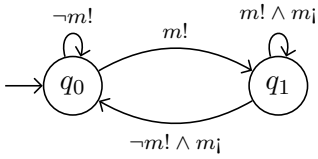


Figure 6: The automaton Λ_m

The channel is attached to the ports JP_2, R_1 and R_2 through the partial injections

- $\mu_{\mathsf{JP}_2} \colon M \to M_{\mathsf{JP}_2}$ given by $g \mapsto \mathsf{getRoutes}$, $r \mapsto \mathsf{routes}$ and $t \mapsto \mathsf{timetables}$,
- $\mu_{\mathsf{R}_1} \colon M \to M_{\mathsf{R}_1}$ given by $g \mapsto \mathsf{getRoutes}$ and $r \mapsto \mathsf{routes}$, and
- $\mu_{\mathsf{R}_2} \colon M \to M_{\mathsf{R}_2}$ given by $r \mapsto \mathsf{routes}$ and $t \mapsto \mathsf{timetables}$.

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 2.9 and 2.13, 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 $\langle M, \Lambda, (\mu_x \colon M \rightharpoonup M_x)_{x \in X} \rangle$, partial translations $(A_{\mu_x} \colon A_M \rightharpoonup A_{M_x})_{x \in X}$ of the actions defined by the channel into actions defined by the ports, as follows:

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

We usually designate the partial maps A_{μ_x} simply by μ_x if there is no danger of confusion.

Remark 2.14. 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, in

aLTL, every connection $\langle M, \Lambda, (\mu_x \colon M \rightharpoonup M_x)_{x \in X} \rangle$ determines a family of spans

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

indexed by points $x \in X$. Then we can consider connections more generally as triples $\langle \Sigma, \Lambda, (\iota_x \colon \Sigma'_x \to \Sigma, \mu_x \colon \Sigma'_x \to \Sigma_x)_{x \in X} \rangle$ in which the signature Σ and the model Λ of Σ abstract the *channel* component, and the spans of signature morphisms $(\iota_x, \mu_x)_{x \in X}$ provide the means of attaching 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 2.15 (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* $\gamma \colon E \to \mathcal{P}(X)$, 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, i.e. for every $e_1, e_2 \in E$ 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. [BGLL09, FHL+05]). We use them instead of graphs [FL13a] because they offer a more flexible mathematical framework for handling the notions of variable and variable binding required in Section 3.

Definition 2.16 (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* $c \in C$, and of

- 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 $\langle M_c, \Lambda_c, (\mu^c_x \colon M_c \rightharpoonup M_x)_{x \in \gamma_c} \rangle$ for every hyperedge $c \in C$.

Example 2.17. By putting together the process and the connection presented in Examples 2.9 and 2.13, we obtain the ARN JourneyPlanner depicted in Figure 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 $\gamma_{\mathsf{JP}} = \{\mathsf{JP}_1, \mathsf{JP}_2\}$ and $\gamma_{\mathsf{C}} = \{\mathsf{JP}_2, \mathsf{R}_1, \mathsf{R}_2\}$.

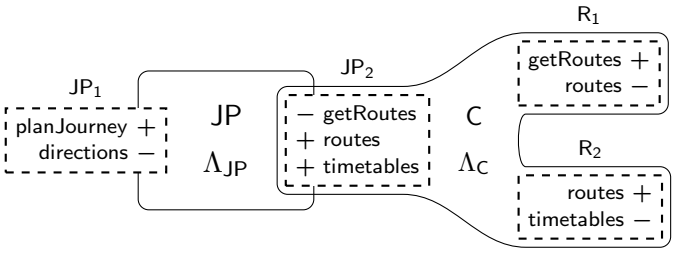


Figure 7: The ARN JourneyPlanner

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. 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 2.18 (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 a point incident only with a computation hyperedge.

For the ARN JourneyPlanner depicted in Figure 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 2.19 (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 functions $h\colon X_1 \to X_2$ and $h\colon E_1 \to E_2$[8] such that for any vertex $x \in X_1$ and hyperedge $e \in E_1$, $x \in \gamma^1_e$ if and only if $h(x) \in \gamma^2_{h(e)}$.

Definition 2.20 (Morphism of ARNs)**.** Given two ARNs $\mathfrak{N}_1 = \langle X_1, P_1, C_1, \gamma^1, M^1, \mu^1, \Lambda^1\rangle$ and $\mathfrak{N}_2 = \langle X_2, P_2, C_2, \gamma^2, M^2, \mu^2, \Lambda^2\rangle$, a *morphism* $\theta\colon \mathfrak{N}_1 \to \mathfrak{N}_2$ consists of

- an injective homomorphism $\theta\colon \langle X_1, P_1, C_1, \gamma^1\rangle \to \langle X_2, P_2, C_2, \gamma^2\rangle$ between the underlying hypergraphs of $\mathfrak{N}_1$ and $\mathfrak{N}_2$ such that $\theta(P_1) \subseteq P_2$ and $\theta(C_1) \subseteq C_2$, and
- a family θ^{pt} of polarity-preserving injections $\theta^{\mathrm{pt}}_x\colon M^1_x \to M^2_{\theta(x)}$, for $x \in X_1$,

such that

- for every point $x \in X_1$ incident with a computation hyperedge, $\theta^{\mathrm{pt}}_x = 1_{M^1_x}$,
- for every computation hyperedge $p \in P_1$, $\Lambda^1_p = \Lambda^2_{\theta(p)}$, and
- for every communication hyperedge $c \in C_1$, $M^1_c = M^2_{\theta(c)}$, $\Lambda^1_c = \Lambda^2_{\theta(c)}$ and the following diagram commutes, for every point $x \in \gamma^1_c$.

$$\begin{array}{ccc} M^1_c = M^2_{\theta(c)} & \xrightarrow{\mu^{1,c}_x} & M^1_x \\ & \searrow^{\mu^{2,\theta(c)}_{\theta(x)}} & \downarrow{\scriptstyle \theta^{\mathrm{pt}}_x} \\ & & M^2_{\theta(x)} \end{array}$$

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 consists solely of set-theoretic identities. We obtain in this way the first result supporting the construction of an orchestration scheme of ARNs.

Proposition 2.21. *The morphisms of* ARN*s form a category, denoted* $\mathbb{ARN}$. □

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

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 2.22 (Specification over an ARN)**.** For any ARN $\mathfrak{N}$, the set $\mathrm{Spec}(\mathfrak{N})$ 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}, i.e. 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 $\theta\colon \mathfrak{N} \to \mathfrak{N}'$, the map $\mathrm{Spec}(\theta)\colon \mathrm{Spec}(\mathfrak{N}) \to \mathrm{Spec}(\mathfrak{N}')$ is given by

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

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 sentences in $\underline{\text{ALTL}}$, thus giving rise to the functor $\mathrm{Spec}\colon \mathbb{ARN} \to \mathbb{S}\mathrm{et}$ that we are looking for.

Ground orchestrations. Morphisms of ARNs can also be regarded as refinements, as 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 2.23 (Dependency)**.** Let x and y be 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, i.e. if there exists an alternating sequence $x\,e_1\,x_1\,\ldots\,e_n\,y$ 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 2.24 (Network defined by a point)**.** The ARN *defined by a point* x of an ARN $\mathfrak{N}$ 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 $\theta\colon \mathfrak{N}_1 \to \mathfrak{N}_2$ assigns to each requires-point x of the source network $\mathfrak{N}_1$ the sub-ARN $\mathfrak{N}_{2,\theta(x)}$ of $\mathfrak{N}_2$ defined by $\theta(x)$.

Example 2.25. In Figure 8 we outline an extension of the ARN JourneyPlanner discussed in Example 2.17 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 $\theta\colon$ JourneyPlanner $\to$ 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, MS_1 only depends on itself, hence the sub-ARN of JourneyPlannerNet defined by MS_1, i.e. 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.

In view of the above observation, we may consider the requires-points of networks as counterparts of the variables used in program expressions, and their morphisms as substitutions. This leads us to the following definition of ground ARNs.

Definition 2.26 (Ground ARN)**.** An ARN is said to be *ground* if it has no requires-points.

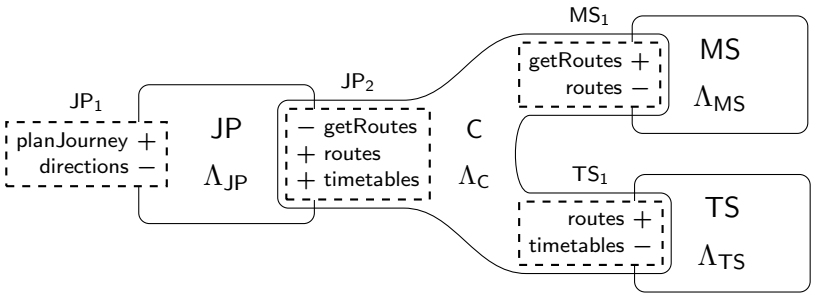


Figure 8: The ARN JourneyPlannerNet

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 2.10 and 2.14 to ARNs.

Fact 2.27 (Diagram of an ARN)**.** Every ARN $\mathfrak{N} = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$ defines a diagram $D_{\mathfrak{N}} \colon \mathbb{J}_{\mathfrak{N}} \to \mathbb{S}\mathrm{ig}^{\underline{\mathrm{aLTL}}}$ 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
  - $\{x \to p \mid p \in P, x \in \gamma_p\}$ for computation hyperedges, and
  - $\{c \leftarrow \langle c, x \rangle_{\mathfrak{N}} \to x \mid c \in C, x \in \gamma_c\}$ 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 $x \in \gamma_c$,
  - $D_{\mathfrak{N}}(c) = A_{M_c}$, $D_{\mathfrak{N}}(\langle c, x \rangle_{\mathfrak{N}}) = \mathrm{dom}(\mu^c_x)$, $D_{\mathfrak{N}}(x) = A_{M_x}$,
  - $D_{\mathfrak{N}}(\langle c, x \rangle_{\mathfrak{N}} \to c) = (\mathrm{dom}(\mu^c_x) \subseteq A_{M_c})$, and
  - $D_{\mathfrak{N}}(\langle c, x \rangle_{\mathfrak{N}} \to x) = \mu^c_x$.

We define the signature of an ARN by taking the colimit of its diagram, which is guaranteed to exist because the category $\mathbb{S}\mathrm{ig}^{\underline{\mathrm{aLTL}}}$, i.e. $\mathbb{S}\mathrm{et}$, is finitely cocomplete.

Definition 2.28 (Signature of an ARN)**.** The signature of an ARN $\mathfrak{N}$ is the colimiting cocone $\xi \colon D_{\mathfrak{N}} \Rightarrow A_{\mathfrak{N}}$ 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 2.29 (Automaton defined by a point)**.** Let x be a point of a ground ARN $\mathfrak{G}$. The *observed automaton* Λ_x at x is given by the reduct $\Lambda_{\mathfrak{G}_x} \restriction_{\xi_x}$, where

- $\mathfrak{G}_x = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$ is the sub-ARN of $\mathfrak{G}$ defined by x,
- $\xi \colon D_{\mathfrak{G}_x} \Rightarrow A_{\mathfrak{G}_x}$ 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 2.30. Consider once again the (ground) ARN represented in Figure 8. The automaton defined by the point MS_1 is just $\Lambda_{\mathsf{MS}} \restriction_{A_{\mathsf{MS}_1}}$; 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, in order 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 2.4 and 2.5, the resulting automaton has to accept precisely the projections to $A_{M_{\mathsf{JP}_1}}$ 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 ground ARNs.

Definition 2.31 (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 $\underline{\text{aLTL}}$) the temporal sentence ρ.

$$\Lambda_x \models^{\underline{\text{aLTL}}} \rho$$

Remark 2.32. It is important to notice that not only the signature of an ARN, but also the various cofree expansions and products considered in Definition 2.29 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 $\underline{\text{aLTL}}$ 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 ARNs 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 subsection: the reflection of the satisfaction of sentences by the model homomorphisms of the institution used as foundation for the construction of ARNs.

Proposition 2.33. *For every morphism of ground* ARN*s* $\theta\colon \mathfrak{G}_1 \to \mathfrak{G}_2$ *and every property* $@_x\,\rho$ *of* $\mathfrak{G}_1$*, the specification* $\mathrm{Spec}(\theta)(@_x\,\rho)$ *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 $\xi^i\colon D_{\mathfrak{G}_i^x} \Rightarrow A_{\mathfrak{G}_i^x}$ 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 the point x in $\mathfrak{G}_1$ satisfies ρ. We also know that $\theta\colon \mathfrak{G}_1 \to \mathfrak{G}_2$ defines the $\underline{\text{aLTL}}$-signature morphism $\theta_x^{\mathrm{pt}}\colon A_{M_{1,x}} \to A_{M_{2,\theta(x)}}$ as the identity of $A_{M_{1,x}}$ (because $\mathfrak{G}_1$ is ground); hence, the automaton $\Lambda^2_{\theta(x)}$ observed at $\theta(x)$ in $\mathfrak{G}_2$ is also a model of $A_{M_{1,x}}$.

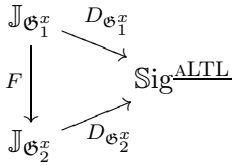
By Proposition 2.6, $\underline{\text{aLTL}}$-model homomorphisms reflect the satisfaction of sentences; therefore, in order to prove that $\Lambda^2_{\theta(x)}$ satisfies ρ – and in this way, that $\mathrm{Spec}(\theta)(@_x\,\rho)$ is a property of $\mathfrak{G}_2$ – it suffices to determine the existence of a homomorphism $\Lambda^2_{\theta(x)} \to \Lambda^1_x$.

Recall that Λ^1_x and $\Lambda^2_{\theta(x)}$ are the reducts $\Lambda_{\mathfrak{G}_1^x}{\restriction}_{\xi^1_x}$ and $\Lambda_{\mathfrak{G}_2^x}{\restriction}_{\xi^2_{\theta(x)}}$, 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 $\pi_e^i\colon \Lambda_{\mathfrak{G}_i^x} \to \Lambda_e^{\mathfrak{G}_i^x}$, and
- $\Lambda_e^{\mathfrak{G}_i^x}$, 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 $\_{\restriction}_{\xi_e^i}$ to Λ_e^i by $\varepsilon_e^i\colon \Lambda_e^{\mathfrak{G}_i^x}{\restriction}_{\xi_e^i} \to \Lambda_e^i$.

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 $F\colon \mathbb{J}_{\mathfrak{G}_1^x} \to \mathbb{J}_{\mathfrak{G}_2^x}$ that makes the following diagram commutative.

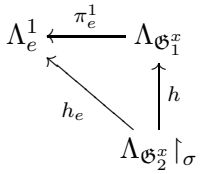


This allows us to define the derived cocone $F \cdot \xi^2\colon D_{\mathfrak{G}_1^x} \Rightarrow A_{\mathfrak{G}_2^x}$, whose components are given, for example, by $(F \cdot \xi^2)_x = \xi^2_{\theta(x)}$. Since ξ^1 is the colimit of $D_{\mathfrak{G}_1^x}$ it follows that there exists a (unique) morphism of cocones $\sigma\colon \xi^1 \to F \cdot \xi^2$, i.e. an $\underline{\mathrm{aLTL}}$-signature morphism $\sigma\colon A_{\mathfrak{G}_1^x} \to A_{\mathfrak{G}_2^x}$ that satisfies, in particular, $\xi^1_e \fatsemi \sigma = \xi^2_{\theta(e)}$ 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 $\pi^2_{\theta(e)}{\restriction_{\xi^2_{\theta(e)}}} \fatsemi \varepsilon^2_{\theta(e)}$ from $\Lambda_{\mathfrak{G}_2^x}{\restriction_{\xi^2_{\theta(e)}}} = \Lambda_{\mathfrak{G}_2^x}{\restriction_\sigma}{\restriction_{\xi^1_e}}$ to $\Lambda^1_e = \Lambda^2_{\theta(e)}$.

$$\begin{array}{ccccc}
\Lambda^1_e = \Lambda^2_{\theta(e)} & \xleftarrow{\ \varepsilon^1_e\ } & \Lambda_e^{\mathfrak{G}_1^x}{\restriction_{\xi^1_e}} & & \Lambda_e^{\mathfrak{G}_1^x} \\
\uparrow{\scriptstyle \varepsilon^2_{\theta(e)}} & & \uparrow{\scriptstyle h_e{\restriction_{\xi^1_e}}} & & \uparrow{\scriptstyle h_e} \\
\Lambda_{\theta(e)}^{\mathfrak{G}_2^x}{\restriction_{\xi^2_{\theta(e)}}} & \xleftarrow[\ \pi^2_{\theta(e)}{\restriction_{\xi^2_{\theta(e)}}}\ ]{} & \Lambda_{\mathfrak{G}_2^x}{\restriction_{\xi^2_{\theta(e)}}} = \Lambda_{\mathfrak{G}_2^x}{\restriction_\sigma}{\restriction_{\xi^1_e}} & & \Lambda_{\mathfrak{G}_2^x}{\restriction_\sigma}
\end{array}$$

Given that $\Lambda_e^{\mathfrak{G}_1^x}$ is the cofree expansion of Λ^1_e along ξ^1_e, we deduce that there exists a (unique) morphism $h_e\colon \Lambda_{\mathfrak{G}_2^x}{\restriction_\sigma} \to \Lambda_e^{\mathfrak{G}_1^x}$ such that the above diagram is commutative. This implies, by the universal property of the product $\Lambda_{\mathfrak{G}_1^x}$, the existence of a (unique) morphism $h\colon \Lambda_{\mathfrak{G}_2^x}{\restriction_\sigma} \to \Lambda_{\mathfrak{G}_1^x}$ such that $h \fatsemi \pi^1_e = h_e$ for every $e \in P_1 \cup C_1$.



It follows that the reduct $h{\restriction_{\xi^1_x}}$ is a morphism from $\Lambda_{\mathfrak{G}_2^x}{\restriction_\sigma}{\restriction_{\xi^1_x}}$ to $\Lambda_{\mathfrak{G}_1^x}{\restriction_{\xi^1_x}}$. Then, to complete the proof, we only need to notice that $\Lambda_{\mathfrak{G}_2^x}{\restriction_\sigma}{\restriction_{\xi^1_x}} = \Lambda_{\mathfrak{G}_2^x}{\restriction_{\xi^2_{\theta(x)}}} = \Lambda^2_{\theta(x)}$ and $\Lambda_{\mathfrak{G}_1^x}{\restriction_{\xi^1_x}} = \Lambda^1_x$. □

3. A Logical View on Service Discovery and Binding

Building on the results of Section 2, let us now investigate how the semantics of the service overlay can be characterized using fundamental computational aspects of the logic-programming paradigm 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 [FLB11], and concepts such as clause and query that are specific to (the relational variant of) logic programming [Llo87]. In order to clarify this analogy we rely on the institutional framework that we put forward in [ŢF15] to address the model-theoretic foundations of logic programming.

We begin by briefly recalling the most basic structure that underlies both the denotational and the operational semantics of relational logic programming: the substitution system of (sets of) variables and substitutions over a given (single-sorted) first-order signature. Its definition relies technically on the category $\mathbb{R}\mathrm{oom}$ of *institution rooms* and *corridors* (see e.g. [Mos02]). The objects of $\mathbb{R}\mathrm{oom}$ are triples $\langle S, \mathbb{M}, \vDash \rangle$ consisting of a set S of *sentences*, a category $\mathbb{M}$ of *models*, and a *satisfaction relation* $\vDash \subseteq |\mathbb{M}| \times S$. They are related through corridors $\langle \alpha, \beta \rangle \colon \langle S, \mathbb{M}, \vDash \rangle \to \langle S', \mathbb{M}', \vDash' \rangle$ that abstract the change of notation within or between logics by defining a *sentence-translation function* $\alpha \colon S \to S'$ and a *model-reduction functor* $\beta \colon \mathbb{M}' \to \mathbb{M}$ such that the following condition holds for all $M' \in |\mathbb{M}'|$ and $\rho \in S$:

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

Definition 3.1 (Substitution system)**.** A *substitution system* is a triple $\langle \mathbb{S}\mathrm{ubst}, G, \mathcal{S} \rangle$, often denoted simply by $\mathcal{S}$, that consists of

- a category $\mathbb{S}\mathrm{ubst}$ of *signatures of variables* and *substitutions*,
- a room G of *ground sentences* and *models*, and
- a functor $\mathcal{S} \colon \mathbb{S}\mathrm{ubst} \to G \,/\, \mathbb{R}\mathrm{oom}$, defining for every signature of variables X the corridor $\mathcal{S}(X) \colon G \to G(X)$ from G to the room $G(X)$ of *X-sentences* and *X-models*.

Example 3.2. In the case of conventional logic programming, every single-sorted first-order signature $\langle F, P \rangle$ determines a substitution system

$$\big(\mathrm{AFOL}^1_{\neq}\big)_{\langle F,P \rangle} \colon \mathbb{S}\mathrm{ubst}_{\langle F,P \rangle} \to \underline{\mathrm{AFOL}}^1_{\neq}(F,P) \,/\, \mathbb{R}\mathrm{oom}^9$$

where $\mathbb{S}\mathrm{ubst}_{\langle F,P \rangle}$ is simply the category whose objects are sets of variables (defined over the signature $\langle F, P \rangle$), and whose arrows are first-order substitutions. The room $\underline{\mathrm{AFOL}}^1_{\neq}(F,P)$ 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. And finally, the functor $\big(\mathrm{AFOL}^1_{\neq}\big)_{\langle F,P \rangle}$ maps every signature (i.e. set) of variables X to the corridor $\langle \alpha_{\langle F,P \rangle, X}, \beta_{\langle F,P \rangle, X} \rangle$,

$$\langle \mathrm{Sen}(F,P), \mathrm{Mod}(F,P), \vDash_{\langle F,P \rangle} \rangle \xrightarrow{\ \alpha_{\langle F,P \rangle, X}\ } \langle \mathrm{Sen}(F \cup X, P), \mathrm{Mod}(F \cup X, P), \vDash_{\langle F \cup X, P \rangle} \rangle$$
$$\langle \mathrm{Sen}(F,P), \mathrm{Mod}(F,P), \vDash_{\langle F,P \rangle} \rangle \xleftarrow{\ \beta_{\langle F,P \rangle, X}\ } \langle \mathrm{Sen}(F \cup X, P), \mathrm{Mod}(F \cup X, P), \vDash_{\langle F \cup X, P \rangle} \rangle$$

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 $\langle F, P \rangle \subseteq \langle F \cup X, P \rangle$.

Substitution systems are particularly useful when reasoning about the semantics of clauses and queries. For instance, the above substitution system can be used to define (definite) clauses over $\langle F, P \rangle$ as syntactic structures $\forall X \cdot C \leftarrow H$, also written

$$C \xleftarrow[X]{} H$$

such that X is a signature of variables, C is sentence over $\langle F \cup X, P \rangle$, and H is a (finite) set of sentences over $\langle F \cup X, P \rangle$.[10] The semantics of such a construction is given by the class

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

[10] Note that, in relational logic programming, the variables are often distinguished from other symbols through notational conventions; for this reason, the set X of variables is at times omitted.

of models of $\langle F, P\rangle$, i.e. of ground models of the substitution system, whose expansions to $\langle F \cup X, P\rangle$ satisfy C whenever they satisfy all sentences in H – this reflects the usual interpretation of logic-programming clauses as universally quantified sentences $\forall X \cdot \bigwedge H \Rightarrow C$.

Similarly to institutions, the axiomatic approach to logic programming on which we rely in this paper is parameterized by the signature used. In categorical terms, this means that the morphisms of signatures induce appropriate morphisms between their corresponding substitution systems, and moreover, that this mapping is functorial. As regards our inquiry on the semantics of the service overlay, it suffices to recall that the category $\mathbb{S}\mathrm{ubst}\mathbb{S}\mathrm{ys}$ of substitution systems results from the Grothendieck construction [TBG91] for the functor $[\_ \to \_ \,/\, \mathbb{R}\mathrm{oom}] \colon (\mathbb{C}\mathrm{at} \times \mathbb{R}\mathrm{oom})^{\mathrm{op}} \to \mathbb{C}\mathrm{at}$ that maps

- every category $\mathbb{S}\mathrm{ubst}$ and room G to the category of functors $[\mathbb{S}\mathrm{ubst} \to G \,/\, \mathbb{R}\mathrm{oom}]$,
- every functor $\Psi \colon \mathbb{S}\mathrm{ubst} \to \mathbb{S}\mathrm{ubst}'$ and corridor $\kappa \colon G \to G'$ to the canonical composition functor $\Psi\_(\kappa \,/\, \mathbb{R}\mathrm{oom}) \colon [\mathbb{S}\mathrm{ubst}' \to G' \,/\, \mathbb{R}\mathrm{oom}] \to [\mathbb{S}\mathrm{ubst} \to G \,/\, \mathbb{R}\mathrm{oom}]$.

This allows us to introduce the next notion of generalized substitution system.

Definition 3.3 (Generalized substitution system)**.** A *generalized substitution system* is a pair $\langle \mathbb{S}\mathrm{ig}, \mathcal{GS}\rangle$ given by a category $\mathbb{S}\mathrm{ig}$ of *signatures*, and a functor $\mathcal{GS} \colon \mathbb{S}\mathrm{ig} \to \mathbb{S}\mathrm{ubst}\mathbb{S}\mathrm{ys}$.

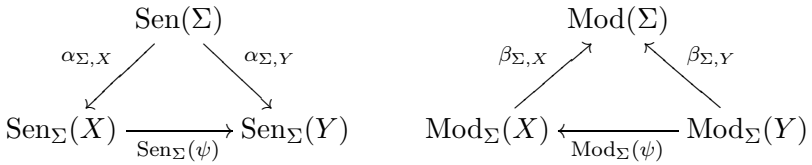
In order to provide a better understanding of the complex structure of generalized substitution systems, we consider the following notational conventions and terminology:

– For every signature Σ of a generalized substitution system $\mathcal{GS}$, we denote the (local) substitution system $\mathcal{GS}(\Sigma)$ by $\mathcal{GS}_\Sigma \colon \mathbb{S}\mathrm{ubst}_\Sigma \to G_\Sigma \,/\, \mathbb{R}\mathrm{oom}$, and we refer to the objects and morphisms of $\mathbb{S}\mathrm{ubst}_\Sigma$ as *signatures of* Σ*-variables* and Σ*-substitutions*. The room G_Σ is assumed to comprise the set $\mathrm{Sen}(\Sigma)$ of *ground* Σ*-sentences*, the category $\mathrm{Mod}(\Sigma)$ of Σ*-models*, and the Σ*-satisfaction relation* $\models_\Sigma \subseteq |\mathrm{Mod}(\Sigma)| \times \mathrm{Sen}(\Sigma)$.

– On objects, $\mathcal{GS}_\Sigma$ maps every signature of Σ-variables X to the corridor $\mathcal{GS}_\Sigma(X) = \langle \alpha_{\Sigma,X}, \beta_{\Sigma,X}\rangle$ from G_Σ to the room $G_\Sigma(X) = \langle \mathrm{Sen}_\Sigma(X), \mathrm{Mod}_\Sigma(X), \models_{\Sigma,X}\rangle$ of X*-sentences* and X*-models*.

$$\alpha_{\Sigma,X} \colon \mathrm{Sen}(\Sigma) \to \mathrm{Sen}_\Sigma(X) \qquad \beta_{\Sigma,X} \colon \mathrm{Mod}_\Sigma(X) \to \mathrm{Mod}(\Sigma)$$

– On arrows, $\mathcal{GS}_\Sigma$ maps every Σ-substitution $\psi \colon X \to Y$ to the corridor $\mathcal{GS}_\Sigma(\psi) = \langle \mathrm{Sen}_\Sigma(\psi), \mathrm{Mod}_\Sigma(\psi)\rangle$ from $G_\Sigma(X)$ to $G_\Sigma(Y)$, which satisfies, by definition, $\mathcal{GS}_\Sigma(X) \fatsemi \mathcal{GS}_\Sigma(\psi) = \mathcal{GS}_\Sigma(Y)$.



– With respect to signature morphisms, every $\varphi\colon \Sigma \to \Sigma'$ determines a morphism of substitution systems $\mathcal{GS}_\varphi\colon \mathcal{GS}_\Sigma \to \mathcal{GS}_{\Sigma'}$ in the form of a triple $\langle \Psi_\varphi, \kappa_\varphi, \tau_\varphi \rangle$, where Ψ_φ is a functor $\mathbb{S}\mathrm{ubst}_\Sigma \to \mathbb{S}\mathrm{ubst}_{\Sigma'}$, κ_φ is a corridor $\langle \mathrm{Sen}(\varphi), \mathrm{Mod}(\varphi) \rangle\colon G_\Sigma \to G_{\Sigma'}$, and for every signature of Σ-variables X, $\tau_{\varphi,X}$ is a (natural) corridor $\langle \alpha_{\varphi,X}, \beta_{\varphi,X} \rangle\colon G_\Sigma(X) \to G_{\Sigma'}(\Psi_\varphi(X))$.

$$\begin{array}{ccc} \mathrm{Sen}(\Sigma) & \xrightarrow{\mathrm{Sen}(\varphi)} & \mathrm{Sen}(\Sigma') \\ {\scriptstyle \alpha_{\Sigma,X}}\downarrow & & \downarrow{\scriptstyle \alpha_{\Sigma',\Psi_\varphi(X)}} \\ \mathrm{Sen}_\Sigma(X) & \xrightarrow[\alpha_{\varphi,X}]{} & \mathrm{Sen}_{\Sigma'}(\Psi_\varphi(X)) \end{array} \qquad \begin{array}{ccc} \mathrm{Mod}(\Sigma) & \xleftarrow{\mathrm{Mod}(\varphi)} & \mathrm{Mod}(\Sigma') \\ {\scriptstyle \beta_{\Sigma,X}}\uparrow & & \uparrow{\scriptstyle \beta_{\Sigma',\Psi_\varphi(X)}} \\ \mathrm{Mod}_\Sigma(X) & \xleftarrow[\beta_{\varphi,X}]{} & \mathrm{Mod}_{\Sigma'}(\Psi_\varphi(X)) \end{array}$$

In addition, we adopt notational conventions that are similar to those used for institutions. For example, we may use superscripts as in $\mathbb{S}\mathrm{ubst}^{\mathcal{GS}}_\Sigma$ is order to avoid potential ambiguities; or we may drop the subscripts of $\vDash_{\Sigma,X}$ when there is no danger of confusion. Also, we will often denote the functions $\mathrm{Sen}(\varphi)$, $\alpha_{\Sigma,X}$ and $\mathrm{Sen}_\Sigma(\psi)$ by $\varphi(\_)$, $X(\_)$ and $\psi(\_)$, respectively, and the functors $\mathrm{Mod}(\varphi)$, $\beta_{\Sigma,X}$ and $\mathrm{Mod}_\Sigma(\psi)$ by $\_\restriction_\varphi$, $\_\restriction_\Sigma$ and $\_\restriction_\psi$.

Example 3.4. Relational logic programming is based upon the generalized substitution system $\mathrm{AFOL}^1_{\neq}$ of the atomic fragment of single-sorted first-order logic without equality.

$$\mathrm{AFOL}^1_{\neq}\colon \mathbb{S}\mathrm{ig}^{\mathrm{AFOL}^1_{\neq}} \to \mathbb{S}\mathrm{ubst}\mathbb{S}\mathrm{ys}$$

In this case, the category $\mathbb{S}\mathrm{ig}^{\mathrm{AFOL}^1_{\neq}}$ is just the category of single-sorted first-order signatures. Every signature $\langle F, P \rangle$ is mapped to a substitution system $\big(\mathrm{AFOL}^1_{\neq}\big)_{\langle F,P \rangle}$ as described in Example 3.2, while every signature morphism $\varphi\colon \langle F, P \rangle \to \langle F', P' \rangle$ resolves to a morphism of substitution systems for which Ψ_φ is the obvious translation of $\langle F, P \rangle$-substitutions along φ, and κ_φ is the corridor $\underline{\mathrm{AFOL}}^1_{\neq}(\varphi)$. A more detailed presentation of first-order generalized substitution systems can be found in [Ţ F15].

3.1. **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 $\mathbb{OS}$ from which there exists a functor OrcScheme into $\mathbb{S}\mathrm{ubst}\mathbb{S}\mathrm{ys}$ 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

$$\mathrm{OrcScheme}\colon \mathbb{OS} \to \mathbb{S}\mathrm{ubst}\mathbb{S}\mathrm{ys}$$

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 $\mathrm{AFOL}^1_{\neq}$ 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 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 2.2, and $\varphi\colon \Sigma \to \Sigma'$ its extension with a new operation symbol `repeat _ until _`: `Pgm Cond` $\to$ `Pgm`. Then, it is easy to notice that the translation of Σ-terms (over a given set of program variables) along φ 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]) $\Sigma' \to \Sigma$ that encodes the `repeat_until_` operation as the term $\underline{1}$; `while not` $\underline{2}$ `do` $\underline{1}$ `done`.[11] 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 $\eta_{pgm} \colon pgm \to U(F(pgm))$ as a result of the potential renaming of program variables; thus, the morphism η_{pgm} accounts for translation of the program variables of pgm along $F \fatsemi U$. Furthermore, for every program expression pgm' over Σ', the translation of Σ-sentences determined by φ extends to a map between the specifications over $U(pgm')$ 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 Σ) to specifications over $F(pgm)$. With respect to the semantics, it is natural to expect that every program expression pgm over Σ has the same behaviour as $F(pgm)$ and, even more, that every program expression pgm' over Σ' (that may be built using `repeat_until_`), behaves in the same way as $U(pgm')$. These observations lead us to the following formalization of the notion of morphism of orchestration schemes.

Definition 3.5 (Morphism of orchestration schemes)**.** A *morphism* between orchestration schemes $\langle \mathbb{O}\mathrm{rc}, \mathrm{Spec}, \mathbb{G}\mathrm{rc}, \mathrm{Prop} \rangle$ and $\langle \mathbb{O}\mathrm{rc}', \mathrm{Spec}', \mathbb{G}\mathrm{rc}', \mathrm{Prop}' \rangle$ is a tuple $\langle F, U, \eta, \sigma \rangle$, where

$$\mathbb{O}\mathrm{rc} \underset{U}{\overset{F}{\rightleftarrows}} \mathbb{O}\mathrm{rc}'$$

- F and U are functors as depicted above such that $F(\mathbb{G}\mathrm{rc}) \subseteq \mathbb{G}\mathrm{rc}'$ and $U(\mathbb{G}\mathrm{rc}') \subseteq \mathbb{G}\mathrm{rc}$,
- η is a natural transformation $1_{\mathbb{O}\mathrm{rc}} \Rightarrow F \fatsemi U$ such that $\eta_{\mathfrak{g}} = 1_{\mathfrak{g}}$ for every $\mathfrak{g} \in |\mathbb{G}\mathrm{rc}|$, and
- σ is a natural transformation $U \fatsemi \mathrm{Spec} \Rightarrow \mathrm{Spec}'$ such that for every ground orchestration $\mathfrak{g}' \in |\mathbb{G}\mathrm{rc}'|$ and specification $\rho \in \mathrm{Spec}(U(\mathfrak{g}'))$,

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

Example 3.6. Let $\mathcal{I} = \langle \mathbb{S}\mathrm{ig}, \mathrm{Sen}, \mathrm{Mod}, \models \rangle$ and $\mathcal{I}' = \langle \mathbb{S}\mathrm{ig}', \mathrm{Sen}', \mathrm{Mod}', \models' \rangle$ be two institutions suitable for defining orchestration schemes of ARNs (according to the hypotheses introduced in Subsection 2.2), and let $\langle \Upsilon, \alpha, \beta \rangle$ be a morphism of institutions $\mathcal{I}' \to \mathcal{I}$ such that $\Upsilon \colon \mathbb{S}\mathrm{ig}' \to \mathbb{S}\mathrm{ig}$ is cocontinuous and $\beta \colon \mathrm{Mod}' \Rightarrow \Upsilon^{\mathrm{op}} \fatsemi \mathrm{Mod}$ preserves cofree expansions and products. If Υ and β admit sections, that is if there exist a functor $\Phi \colon \mathbb{S}\mathrm{ig} \to \mathbb{S}\mathrm{ig}'$ such that $\Phi \fatsemi \Upsilon = 1_{\mathbb{S}\mathrm{ig}}$ and a natural transformation $\tau \colon \mathrm{Mod} \Rightarrow \Phi^{\mathrm{op}} \fatsemi \mathrm{Mod}'$ such that $\tau \fatsemi (\Phi^{\mathrm{op}} \cdot \beta) = 1_{\mathrm{Mod}}$, 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 $\mathcal{I}$ and $\mathcal{I}'$. In particular, the functor F maps the diagram and the models that label an ARN defined over $\mathcal{I}$ to their images under Φ and τ; similarly, U maps ARNs defined over $\mathcal{I}'$ 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 $\underline{\text{ALTL}}$ (in the role of $\mathcal{I}$) with new temporal modalities such as *previous* and *since*, as in [KMWZ10]; 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

[11] In this context, $\underline{1}$: `Pgm` and $\underline{2}$: `Cond` are variables corresponding to the arguments of the derived operation.

deterministically recognizable – for which a minimal automaton can be shown to exist (see e.g. [MS97, Löd01]). In this case, in the roles of $\mathcal{I}$ and $\mathcal{I}'$ we could consider variants of $\underline{\text{ALTL}}$ with models given by sets of traces and deterministic weak automata, respectively;[12] 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 it 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.

Proposition 3.7. *The morphisms of orchestration schemes can be composed as follows:*

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

Under this composition, orchestration schemes and their morphisms form a category $\mathbb{OS}$. □

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

1. Orchestrations 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 of 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 [SML04].
2. There is a one-to-one correspondence between institutions and substitution systems defined over the initial room $\langle \emptyset, \ , \emptyset\rangle$ – the room given by the empty set of sentences, the terminal category $\ $, 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 $\langle \emptyset, \ , \emptyset\rangle$; therefore, we obtain precisely the notion of correctness of a service module [FLB11]: all models of the underlying signature of variables, i.e. of the orchestration, that satisfy the antecedent of the clause satisfy its consequent as well.

Formally, OrcScheme results from the composition of two functors, $\mathrm{Ins}\colon \mathbb{OS} \to \mathbb{coIns}$ and $\mathrm{SS}\colon \mathbb{coIns} \to \mathbb{SubstSys}$, that implement the general constructions outlined above.

$$\mathbb{OS} \xrightarrow{\ \mathrm{Ins}\ } \mathbb{coIns} \xrightarrow{\ \mathrm{SS}\ } \mathbb{SubstSys}, \qquad \mathbb{OS} \xrightarrow{\ \mathrm{OrcScheme}\ } \mathbb{SubstSys}$$

The functor Ins carries most of the complexity of OrcScheme, and is discussed in detail in Theorem 3.8. Concerning SS, we recall from [ŢF15] that the category $\mathbb{coIns}$ of institution comorphisms can also be described as the category $[\_ \to \mathbb{Room}]^\sharp$ of functors into $\mathbb{Room}$, and that any functor $G\colon \mathbb{K} \to \mathbb{K}'$ can be extended to a functor $[\_ \to \mathbb{K}]^\sharp \to [\_ \to \mathbb{K}']^\sharp$ that is given essentially by the right-composition with G. In particular, the isomorphism $\mathbb{Room} \to \langle \emptyset, \ , \emptyset\rangle \,/\, \mathbb{Room}$ that maps every room $\langle S, \mathbb{M}, \vDash\rangle$ to the unique corridor $\langle \emptyset, \ , \emptyset\rangle \to \langle S, \mathbb{M}, \vDash\rangle$ generates an isomorphism of categories between $[\_ \to \mathbb{Room}]^\sharp$, i.e. $\mathbb{coIns}$, and $[\_ \to \langle \emptyset, \ , \emptyset\rangle \,/\, \mathbb{Room}]^\sharp$. The latter is further embedded into $\mathbb{SubstSys}$, defining in this way, by composition, the required functor SS. To sum up, SS maps every institution

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

$\mathcal{I}\colon \mathbb{S}\mathrm{ig} \to \mathbb{R}\mathrm{oom}$ to the substitution system $\mathcal{S}\colon \mathbb{S}\mathrm{ig} \to \langle \emptyset, \ , \emptyset \rangle \,/\, \mathbb{R}\mathrm{oom}$ for which $\mathcal{S}(\Sigma)$, for every signature $\Sigma \in |\mathbb{S}\mathrm{ig}|$, is the unique corridor between $\langle \emptyset, \ , \emptyset \rangle$ and $\mathcal{I}(\Sigma)$.

Theorem 3.8. *The following map defines a functor* $\mathrm{Ins}\colon \mathbb{OS} \to \mathrm{co}\mathbb{I}\mathrm{ns}$.

- *For any orchestration scheme* $\mathcal{O} = \langle \mathbb{O}\mathrm{rc}, \mathrm{Spec}, \mathbb{G}\mathrm{rc}, \mathrm{Prop} \rangle$, $\mathrm{Ins}(\mathcal{O})$ *is the institution whose category of signatures is* $\mathbb{O}\mathrm{rc}$*, sentence functor is* Spec*, model functor is* $\_ / \mathbb{G}\mathrm{rc}$*, and whose family of satisfaction relations is given by*

$$(\delta\colon \mathfrak{o} \to \mathfrak{g}) \models_{\mathfrak{o}} SP \qquad \text{if and only if} \qquad \mathrm{Spec}(\delta)(SP) \in \mathrm{Prop}(\mathfrak{g})$$

  for every orchestration $\mathfrak{o}$*, every* $\mathfrak{o}$*-model* δ*, i.e. every morphism of orchestrations* $\delta\colon \mathfrak{o} \to \mathfrak{g}$ *such that* $\mathfrak{g}$ *is ground, and every specification* SP *over* $\mathfrak{o}$*.*[13]
- *For any morphism of orchestration schemes* $\langle F, U, \eta, \sigma \rangle\colon \mathcal{O} \to \mathcal{O}'$*, with* $\mathcal{O}$ *as above and* $\mathcal{O}'$ *given by* $\langle \mathbb{O}\mathrm{rc}', \mathrm{Spec}', \mathbb{G}\mathrm{rc}', \mathrm{Prop}' \rangle$*,* $\mathrm{Ins}(F, U, \eta, \sigma)$ *is the comorphism of institutions* $\langle F, \alpha, \beta \rangle\colon \mathrm{Ins}(\mathcal{O}) \to \mathrm{Ins}(\mathcal{O}')$ *defined by*

$$\alpha_{\mathfrak{o}} = \mathrm{Spec}(\eta_{\mathfrak{o}}) \fatsemi \sigma_{F(\mathfrak{o})}$$
$$\beta_{\mathfrak{o}} = \upsilon_{F(\mathfrak{o})} \fatsemi (\eta_{\mathfrak{o}} \,/\, \mathbb{G}\mathrm{rc})$$

  for every orchestration $\mathfrak{o} \in |\mathbb{O}\mathrm{rc}|$*, where* $\upsilon\colon (\_ \,/\, \mathbb{G}\mathrm{rc}') \Rightarrow U^{\mathrm{op}} \fatsemi (\_ \,/\, \mathbb{G}\mathrm{rc})$ *is the natural transformation given by* $\upsilon_{\mathfrak{o}'}(x) = U(x)$ *for every orchestration* $\mathfrak{o}' \in |\mathbb{O}\mathrm{rc}'|$ *and every object or arrow* x *of the comma category* $\mathfrak{o}' \,/\, \mathbb{G}\mathrm{rc}'$*.*

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 $\theta\colon \mathfrak{o}_1 \to \mathfrak{o}_2$, every $\mathfrak{o}_1$-specification SP and every $\mathfrak{o}_2$-model $\delta\colon \mathfrak{o}_2 \to \mathfrak{g}$,

$$\begin{aligned} \delta \models_{\mathfrak{o}_2} \mathrm{Spec}(\theta)(SP) \qquad & \text{if and only if} \qquad \mathrm{Spec}(\theta \fatsemi \delta)(SP) \in \mathrm{Prop}(\mathfrak{g}) \\ & \text{if and only if} \qquad (\theta \,/\, \mathbb{G}\mathrm{rc})(\delta) = \theta \fatsemi \delta \models_{\mathfrak{o}_2} SP. \end{aligned}$$

As regards the second part of the statement, let us begin by noticing that α and β are the natural transformations $(\eta \cdot \mathrm{Spec}) \fatsemi (F \cdot \sigma)$ and $(\eta^{\mathrm{op}} \cdot (\_ \,/\, \mathbb{G}\mathrm{rc})) \fatsemi (F^{\mathrm{op}} \cdot \upsilon)$, respectively. Then, in order to verify that $\langle F, \alpha, \beta \rangle$ is indeed a comorphism $\mathrm{Ins}(\mathcal{O}) \to \mathrm{Ins}(\mathcal{O}')$, consider an orchestration $\mathfrak{o}$ in $\mathbb{O}\mathrm{rc}$, a model $\delta'\colon F(\mathfrak{o}) \to \mathfrak{g}'$ of $F(\mathfrak{o})$, and a specification SP over $\mathfrak{o}$. Assuming that $\models'$ is the family of satisfaction relations of $\mathrm{Ins}(\mathcal{O}')$, we deduce that

$\delta' \models'_{F(\mathfrak{o})} \alpha_{\mathfrak{o}}(SP)$

iff	$\mathrm{Spec}'(\delta')(\alpha_{\mathfrak{o}}(SP)) \in \mathrm{Prop}'(\mathfrak{g}')$	by the definition of $\models'_{F(\mathfrak{o})}$
iff	$\mathrm{Spec}'(\delta')(\sigma_{F(\mathfrak{o})}(\mathrm{Spec}(\eta_{\mathfrak{o}})(SP))) \in \mathrm{Prop}'(\mathfrak{g}')$	by the definition of $\alpha_{\mathfrak{o}}$
iff	$\sigma_{\mathfrak{g}'}(\mathrm{Spec}(\eta_{\mathfrak{o}} \fatsemi U(\delta'))(SP)) \in \mathrm{Prop}'(\mathfrak{g}')$	by the naturality of σ
iff	$\mathrm{Spec}(\eta_{\mathfrak{o}} \fatsemi U(\delta'))(SP) \in \mathrm{Prop}(U(\mathfrak{g}'))$	since $\mathrm{Prop}(U(\mathfrak{g}')) = \sigma^{-1}_{\mathfrak{g}'}(\mathrm{Prop}'(\mathfrak{g}'))$
iff	$\eta_{\mathfrak{o}} \fatsemi U(\delta') \models_{\mathfrak{o}} SP$	by the definition of $\models_{\mathfrak{o}}$
iff	$\beta_{\mathfrak{o}}(\delta') \models_{\mathfrak{o}} SP$	by the definition of $\beta_{\mathfrak{o}}$.

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

[13] Moreover, $\mathrm{Ins}(\mathcal{O})$ is exact, because the functor $\_ \,/\, \mathbb{G}\mathrm{rc}\colon \mathbb{O}\mathrm{rc}^{\mathrm{op}} \to \mathbb{C}\mathrm{at}$ is continuous (see e.g. [Mes89]).

below, and suppose that $\mathrm{Ins}(F,U,\eta,\sigma) = \langle F,\alpha,\beta\rangle$ and $\mathrm{Ins}(F',U',\eta',\sigma') = \langle F',\alpha',\beta'\rangle$.

$$\langle \mathbb{O}\mathrm{rc}, \mathrm{Spec}, \mathbb{G}\mathrm{rc}, \mathrm{Prop}\rangle \xrightarrow{\langle F,U,\eta,\sigma\rangle} \langle \mathbb{O}\mathrm{rc}', \mathrm{Spec}', \mathbb{G}\mathrm{rc}', \mathrm{Prop}'\rangle \xrightarrow{\langle F',U',\eta',\sigma'\rangle} \langle \mathbb{O}\mathrm{rc}'', \mathrm{Spec}'', \mathbb{G}\mathrm{rc}'', \mathrm{Prop}''\rangle$$
$$\langle F \fatsemi F', U' \fatsemi U, \eta \fatsemi (F \cdot \eta' \cdot U), (U' \cdot \sigma) \fatsemi \sigma' \rangle$$

In addition, let $\upsilon\colon (\_ \,/\, \mathbb{G}\mathrm{rc}') \Rightarrow U^{\mathrm{op}} \fatsemi (\_ \,/\, \mathbb{G}\mathrm{rc})$ and $\upsilon'\colon (\_ \,/\, \mathbb{G}\mathrm{rc}'') \Rightarrow U'^{\mathrm{op}} \fatsemi (\_ \,/\, \mathbb{G}\mathrm{rc}')$ 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 $\mathrm{Ins}(\langle F,U,\eta,\sigma\rangle \fatsemi \langle F',U',\eta',\sigma'\rangle)$ is a comorphism of institutions of the form $\langle F \fatsemi F', \alpha'', \beta''\rangle$, where α'' and β'' are given by

$$\alpha''_{\mathfrak{o}} = \mathrm{Spec}((\eta \fatsemi (F \cdot \eta' \cdot U))_{\mathfrak{o}}) \fatsemi ((U' \cdot \sigma) \fatsemi \sigma')_{(F \fatsemi F')(\mathfrak{o})}$$
$$\beta''_{\mathfrak{o}} = (\upsilon' \fatsemi (U'^{\mathrm{op}} \cdot \upsilon))_{(F \fatsemi F')(\mathfrak{o})} \fatsemi ((\eta \fatsemi (F \cdot \eta' \cdot U))_{\mathfrak{o}} \,/\, \mathbb{G}\mathrm{rc}).$$

In order to complete the proof we need to show that $\alpha'' = \alpha \fatsemi (F \cdot \alpha')$ and $\beta'' = (F \cdot \beta') \fatsemi \beta$. 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 υ (in the case of β'').

$$\begin{aligned}
\alpha''_{\mathfrak{o}} &= \mathrm{Spec}(\eta_{\mathfrak{o}}) \fatsemi \underbrace{\mathrm{Spec}(U(\eta'_{F(\mathfrak{o})})) \fatsemi \sigma_{(F \fatsemi F' \fatsemi U')(\mathfrak{o})}} \fatsemi \sigma'_{(F \fatsemi F')(\mathfrak{o})} \\
&= \mathrm{Spec}(\eta_{\mathfrak{o}}) \fatsemi \sigma_{F(\mathfrak{o})} \fatsemi \mathrm{Spec}'(\eta'_{F(\mathfrak{o})}) \fatsemi \sigma'_{(F \fatsemi F')(\mathfrak{o})} \\
&= \alpha_{\mathfrak{o}} \fatsemi \alpha'_{F(\mathfrak{o})} \\
\beta''_{\mathfrak{o}} &= \upsilon'_{(F \fatsemi F')(\mathfrak{o})} \fatsemi \underbrace{\upsilon_{(F \fatsemi F' \fatsemi U')(\mathfrak{o})} \fatsemi (U(\eta'_{F(\mathfrak{o})}) \,/\, \mathbb{G}\mathrm{rc})} \fatsemi (\eta_{\mathfrak{o}} \,/\, \mathbb{G}\mathrm{rc}) \\
&= \upsilon'_{(F \fatsemi F')(\mathfrak{o})} \fatsemi (\eta'_{F(\mathfrak{o})} \,/\, \mathbb{G}\mathrm{rc}') \fatsemi \upsilon_{F(\mathfrak{o})} \fatsemi (\eta_{\mathfrak{o}} \,/\, \mathbb{G}\mathrm{rc}) \\
&= \beta'_{F(\mathfrak{o})} \fatsemi \beta_{\mathfrak{o}}
\end{aligned}$$

□

Corollary 3.9. *The pair* $\langle \mathbb{OS}, \mathrm{OrcScheme}\rangle$ *defines a generalized substitution system.* □

We recall from [ŢF15] 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, for example, that the correctness property of service modules does not depend on the actual orchestration scheme over which the modules are defined.

Definition 3.10 (Model amalgamation)**.** A generalized substitution system $\mathcal{GS}\colon \mathbb{S}\mathrm{ig} \to \mathbb{S}\mathrm{ubst}\mathbb{S}\mathrm{ys}$ has *weak model amalgamation* when for every signature morphism $\varphi\colon \Sigma \to \Sigma'$ and every signature of Σ-variables X, the diagram depicted below is a weak pullback.

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

This means that for every model Σ'-model M' and every X-model N such that $M'\restriction_{\varphi} = N\restriction_{\Sigma}$ there exists a $\Psi_{\varphi}(X)$-model N' that satisfies $N'\restriction_{\Sigma'} = M'$ and $\beta_{\varphi,X}(N') = N$.

Proposition 3.11. *The generalized substitution system* OrcScheme: $\mathbb{OS} \to \mathbb{S}\text{ubst}\mathbb{S}\text{ys}$ *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 3.5, and let $\mathfrak{o}$ be an orchestration of $\mathcal{O}$. Since orchestrations define substitution systems over the initial room $\langle \emptyset, \ , \emptyset \rangle$, we can redraw the diagram of interest as follows:

$$\begin{array}{ccc} |\ |\ & \xleftarrow{\text{-}\restriction_{\varphi}} & |\ |\ \\ \text{-}\restriction_{\mathcal{O}} \uparrow & & \uparrow \text{-}\restriction_{\mathcal{O}'} \\ |\mathfrak{o} \,/\, \mathbb{G}\text{rc}| & \xleftarrow[\beta_{\varphi,\mathfrak{o}}]{} & |F(\mathfrak{o}) \,/\, \mathbb{G}\text{rc}'| \end{array}$$

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 3.8, we know that $\beta_{\varphi,\mathfrak{o}}(\delta') = \eta_{\mathfrak{o}} \fatsemi U(\delta')$ for every object $\delta' \colon F(\mathfrak{o}) \to \mathfrak{g}'$ in $|F(\mathfrak{o}) \,/\, \mathbb{G}\text{rc}'|$. Therefore, for every $\delta \colon \mathfrak{o} \to \mathfrak{g}$ in $|\mathfrak{o} \,/\, \mathbb{G}\text{rc}|$ we obtain

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

□

Remark 3.12. In addition to model amalgamation, it is important to notice that, similarly to $\text{AFOL}^{1}_{\neq}$, 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 $\mathfrak{o}$, a specification SP over $\mathfrak{o}$, and a homomorphism ζ between $\mathfrak{o}$-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 $\delta_2 \models^{\text{OrcScheme}} SP$.

$$\begin{array}{ccc} & \mathfrak{o} & \\ {\scriptstyle \delta_1}\swarrow & & \searrow{\scriptstyle \delta_2} \\ \mathfrak{g}_1 & \xrightarrow[\zeta]{} & \mathfrak{g}_2 \end{array}$$

3.2. **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 $\text{AFOL}^{1}_{\neq}$. 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 concept of service module presented in [FLB11], and also to the concept of orchestrated interface discussed in [FL13a].

Definition 3.13 (Service clause)**.** A *(definite) service-oriented clause* over a given orchestration scheme $\mathcal{O} = \langle \mathbb{O}\mathrm{rc}, \mathrm{Spec}, \mathbb{G}\mathrm{rc}, \mathrm{Prop} \rangle$ is a structure $\forall \mathfrak{o} \cdot P \leftarrow R$, also denoted

$$P \underset{\mathfrak{o}}{\leftarrow} R$$

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

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

Definition 3.14 (Correct clause)**.** A service-oriented clause $\forall \mathfrak{o} \cdot P \leftarrow R$ is *correct* if for every morphism $\delta \colon \mathfrak{o} \to \mathfrak{g}$ such that $\mathfrak{g}$ is a ground orchestration and $\mathrm{Spec}(\delta)(R)$ consists only of properties of $\mathfrak{g}$, the specification $\mathrm{Spec}(\delta)(P)$ is also a property of $\mathfrak{g}$.

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

Example 3.15. We have already encountered several instances of service clauses in the form of the program modules depicted in Figure 1. Their provides- and requires-interfaces are placed on the left- 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 2.17 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:

$$@_{\mathsf{JP}_1}\, \rho^{\mathsf{JP}} \underset{\mathsf{JourneyPlanner}}{\longleftarrow} \left\{ @_{\mathsf{R}_1}\, \rho_1^{\mathsf{JP}}, @_{\mathsf{R}_2}\, \rho_2^{\mathsf{JP}} \right\}$$

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

Client applications are captured in the present setting by service-oriented queries. The way they are defined is similar to that of service clauses, but their semantics is based on an existential quantification, not on a universal one.

Definition 3.16 (Service query)**.** A *service-oriented query* over an orchestration scheme $\mathcal{O} = \langle \mathbb{O}\mathrm{rc}, \mathrm{Spec}, \mathbb{G}\mathrm{rc}, \mathrm{Prop} \rangle$ is a structure $\exists \mathfrak{o} \cdot Q$, also written

$$\underset{\mathfrak{o}}{\vdash} Q$$

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 3.17 (Satisfiable query)**.** A service-oriented query $\exists \mathfrak{o} \cdot Q$ is said to be *satisfiable* if there exists a morphism of orchestrations $\delta \colon \mathfrak{o} \to \mathfrak{g}$ such that $\mathfrak{g}$ is ground and all specifications in $\mathrm{Spec}(\delta)(Q)$ are properties of $\mathfrak{g}$.

Example 3.18. Figure 9 outlines the ARN of a possible client application for the service module Journey Planner discussed in Example 3.15. We specify the actual application, called Traveller, through the service query

$$\vdash_{\mathsf{Traveller}} \{@_{\mathsf{R}_1}\, \rho_1^{\mathsf{T}}\}$$

given by the aLTL-sentence $\Box(\mathsf{getRoute}_{\text{¡}} \Rightarrow \Diamond \mathsf{route}!)$.

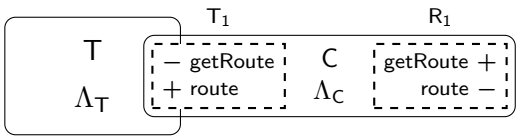


Figure 9: The ARN Traveller

3.3. **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, i.e. 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 $\iota_1 \colon [\rho_1, \rho_1']$ and $\iota_2 \colon [\rho_2, \rho_2']$ over programs $pgm_1 \colon eXp_1$ and $pgm_2 \colon eXp_2$, respectively, $\iota_2 \colon [\rho_2, \rho_2']$ refines $\iota_1 \colon [\rho_1, \rho_1']$ up to a cospan

$$pgm_1 \colon eXp_1 \xrightarrow{\langle \psi_1, \pi_1 \rangle} pgm \colon eXp \xleftarrow{\langle \psi_2, \pi_2 \rangle} pgm_2 \colon eXp_2$$

if by translation we obtain specifications that refer to the same position of $pgm \colon eXp$, i.e. $\pi_1 \cdot \iota_1 = \pi_2 \cdot \iota_2$, such that the pre-condition $\psi_2(\rho_2)$ is weaker that $\psi_1(\rho_1)$, and the post-condition $\psi_2(\rho_2')$ is stronger than $\psi_1(\rho_1')$, meaning that

$$\psi_1(\rho_1) \models^{\underline{\mathrm{POA}}} \psi_2(\rho_2) \qquad \text{and} \qquad \psi_2(\rho_2') \models^{\underline{\mathrm{POA}}} \psi_1(\rho_1').$$

This notion of refinement reflects the rules of consequence introduced in [Hoa69] (see also [Mor94], whence we also adopt the notation $\iota_1 \colon [\rho_1, \rho_1'] \sqsubseteq \iota_2 \colon [\rho_2, \rho_2']$ used in Figure 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 $\langle \theta_1 \colon \mathfrak{N}_1 \to \mathfrak{N}, \theta_2 \colon \mathfrak{N}_2 \to \mathfrak{N} \rangle$ when $\theta_1(x_1) = \theta_2(x_2)$ and $\theta^{\mathrm{pt}}_{2,x_2}(\rho_2) \models^{\underline{\mathrm{aLTL}}} \theta^{\mathrm{pt}}_{1,x_1}(\rho_1)$ [ŢF13].

Both of these notions of refinement generalize to the following concept of unification.

Definition 3.19 (Unification)**.** Let SP_1 and SP_2 be specifications defined over orchestrations $\mathfrak{o}_1$ and $\mathfrak{o}_2$, 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

$$\mathfrak{o}_1 \xrightarrow{\theta_1} \mathfrak{o} \xleftarrow{\theta_2} \mathfrak{o}_2$$

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

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 $\delta \colon \mathfrak{o} \to \mathfrak{g}$ such that $\mathfrak{g}$ is a ground orchestration, if $\mathrm{Spec}(\theta_2 \fatsemi \delta)(SP_2)$ is a property of g then so is $\mathrm{Spec}(\theta_1 \fatsemi \delta)(SP_1)$.

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*. However, unlike relational logic programming, in the case of services 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 3.20 (Resolution)**.** Let $\exists \mathfrak{o}_1 \cdot Q_1$ be a query and $\forall \mathfrak{o}_2 \cdot P_2 \leftarrow R_2$ a clause defined over an arbitrary but fixed orchestration scheme. A query $\exists \mathfrak{o} \cdot Q$ is said to be *derived by resolution* from $\exists \mathfrak{o}_1 \cdot Q_1$ and $\forall \mathfrak{o}_2 \cdot P_2 \leftarrow R_2$ using the *computed morphism* $\theta_1 \colon \mathfrak{o}_1 \to \mathfrak{o}$ when

$$\frac{\underset{\mathfrak{o}_1}{\longmapsto} Q_1 \qquad\qquad P_2 \underset{\mathfrak{o}_2}{\longleftarrow} R_2}{\underset{\mathfrak{o}}{\longmapsto} \theta_1(Q_1 \setminus \{SP_1\}) \cup \theta_2(R_2)}\,\theta_1$$

- 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, and
- 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 3.21. Consider the service query and the clause detailed in Examples 3.18 and 3.15. One can easily see that the single specification $@_{\mathsf{R}_1}\, \rho_1^{\mathsf{T}}$ of the requires-interface of the application Traveller and the provides-interface $@_{\mathsf{JP}_1}\, \rho^{\mathsf{JP}}$ of the module Journey Planner form a unifiable pair: they admit, for instance, the unifier $\langle \theta_1, \theta_2 \rangle$ given by

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

- the ARN JourneyPlannerApp depicted in Figure 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, respectively, while preserving all the remaining elements of Traveller, and
- the inclusion θ_2 of JourneyPanner into JourneyPlannerApp.

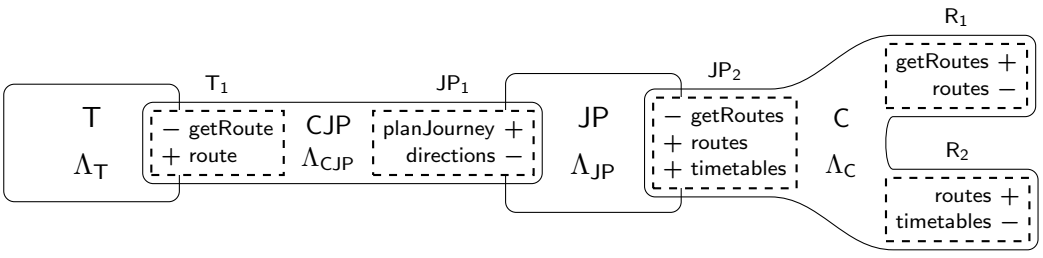


Figure 10: The ARN JourneyPlannerApp

It follows that we can derive by resolution a new service query defined by the network JourneyPlannerApp and the requires-specifications $@_{\mathsf{R}_1}\, \rho_1^{\mathsf{JP}}$ and $@_{\mathsf{R}_2}\, \rho_2^{\mathsf{JP}}$.

$$\dfrac{\vdash_{\mathsf{Traveller}} \{@_{\mathsf{R}_1}\, \rho_1^{\mathsf{T}}\} \qquad @_{\mathsf{JP}_1}\, \rho^{\mathsf{JP}} \leftarrow_{\mathsf{JourneyPlanner}} \{@_{\mathsf{R}_1}\, \rho_1^{\mathsf{JP}}, @_{\mathsf{R}_2}\, \rho_2^{\mathsf{JP}}\}}{\vdash_{\mathsf{JourneyPlannerApp}} \{@_{\mathsf{R}_1}\, \rho_1^{\mathsf{JP}}, @_{\mathsf{R}_2}\, \rho_2^{\mathsf{JP}}\}}\ \theta_1$$

The logic-programming framework of services. The crucial property of the above notions of service 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 [ŢF15]. 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 3.11), and also the fact that the satisfaction of specifications is preserved by model homomorphisms (detailed in Remark 3.12), are far from trivial, especially when taking into account particular orchestration schemes (see e.g. Proposition 2.33).

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 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 3.22 (Solution)**.** A *solution*, or *correct answer*, to a service-oriented query $\exists \mathfrak{o} \cdot Q$ consists of a morphism of orchestrations $\psi \colon \mathfrak{o} \to \mathfrak{o}'$ such that $\mathfrak{o}'$ has models, and every one of them satisfies the ψ-translations of the specifications in Q.

Proposition 3.23. *A service query is satisfiable if and only if it admits a solution.* □

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 $\exists \mathfrak{o} \cdot Q$ 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 2. The computed answer is given in this case by the sequence of substitutions

$$\begin{aligned} pgm &\mapsto pgm_1 \fatsemi pgm_2 \mapsto (pgm_3 \fatsemi pgm_4) \fatsemi pgm_2 \mapsto \cdots \\ &\mapsto (q \coloneqq 0 \fatsemi r \coloneqq x) \fatsemi \mathtt{while}\, y \leq r\, \mathtt{do} \\ &\qquad\qquad\qquad\qquad q \coloneqq q + 1 \fatsemi r \coloneqq r - y \\ &\qquad\qquad\qquad\qquad \mathtt{done}. \end{aligned}$$

In a similar manner, we can continue Example 3.21 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 2.25, and whose provides-interfaces meet the requires-specifications of the module Journey Planner. We obtain in this way the construction outlined in Figure 11.

The soundness of resolution, detailed in Proposition 3.24 below, entails 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, originally discussed in [ŢF15] in the context of abstract logic programming, ensures, in combination with Proposition 3.23, 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 3.24. *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 computed morphism $\theta_1 \colon \mathfrak{o}_1 \to \mathfrak{o}$. If $\forall \mathfrak{o}_2 \cdot P_2 \leftarrow R_2$ is correct then, for any solution ψ to $\exists \mathfrak{o} \cdot Q$, the composed morphism $\theta_1 \fatsemi \psi$ is a solution to $\exists \mathfrak{o}_1 \cdot Q_1$.* □

4. Conclusions

We have shown how the integration of the declarative and the operational semantics of conventional logic programming can be generalized to service-oriented computing, thus offering a unified semantics for the static and the dynamic aspects of this paradigm. That is, we have provided, for the first time, an algebraic framework that accounts for the mechanisms through which service interfaces can be orchestrated, as well as for those mechanisms that allow applications to discover and bind to services.

The analogy that we have established is summarized in Table 1. 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.

The theory of services that we have developed here 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 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

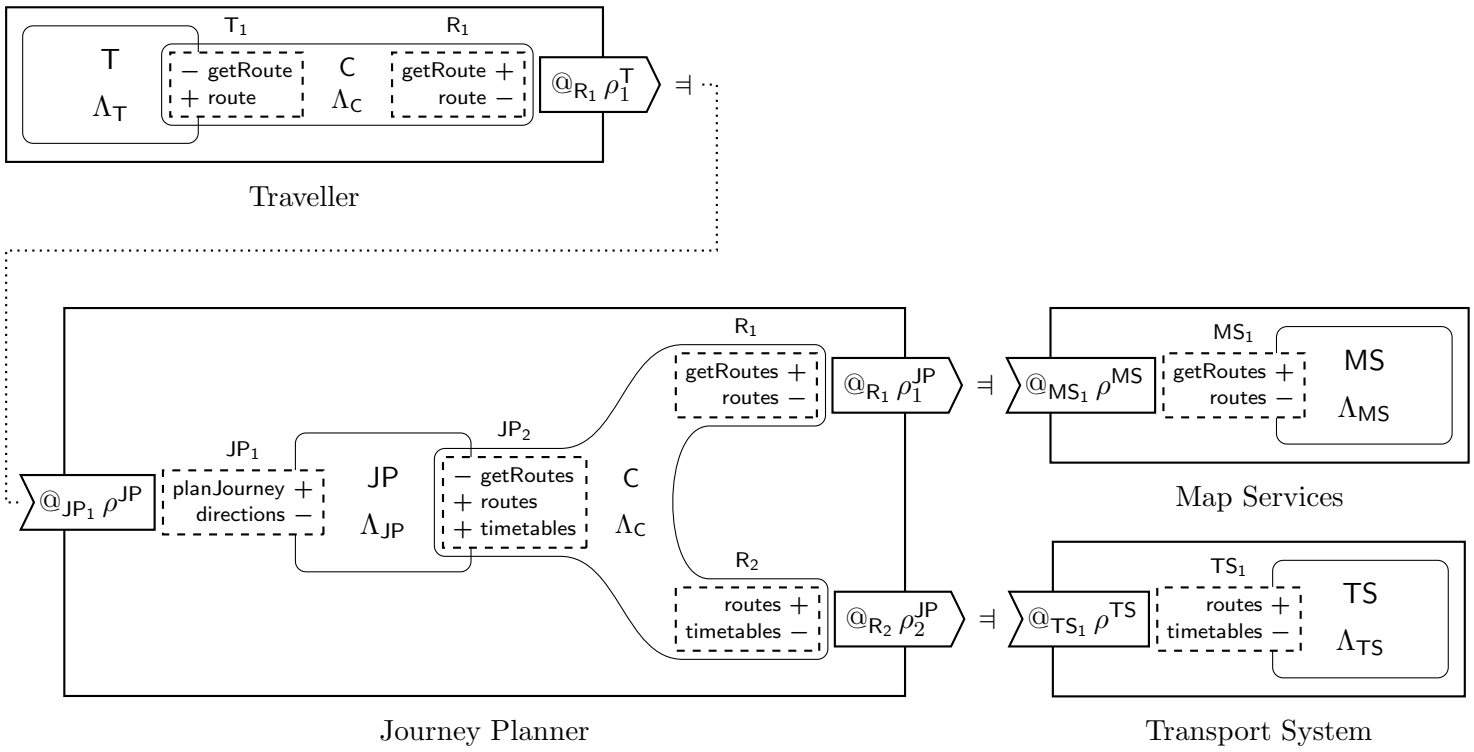
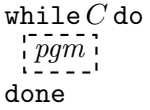
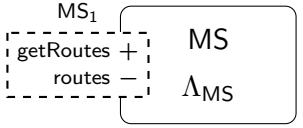
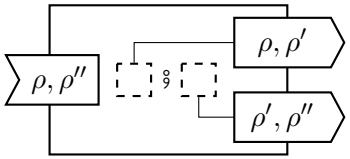
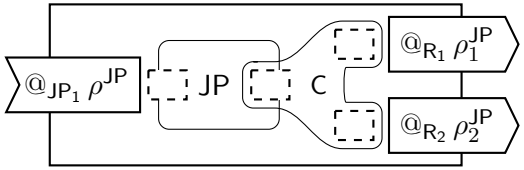
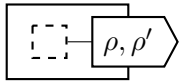
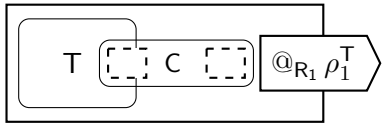


Figure 11: The derivation of an answer to the Traveller application

$\rho_1^{\mathsf{T}} : \Box(\mathsf{getRoute}_{\text{¡}} \Rightarrow \Diamond\mathsf{route}!)$
$\rho^{\mathsf{JP}} : \Box(\mathsf{planJourney}_{\text{¡}} \Rightarrow \Diamond\mathsf{directions}!)$
$\rho_1^{\mathsf{JP}} : \Box(\mathsf{getRoutes}_{\text{¡}} \Rightarrow \Diamond\mathsf{routes}!)$
$\rho_2^{\mathsf{JP}} : \Box(\mathsf{routes}_{\text{¡}} \Rightarrow \Diamond\mathsf{timetables}!)$
$\rho^{\mathsf{MS}} : \Box(\mathsf{getRoutes}_{\text{¡}} \Rightarrow \Diamond\mathsf{routes}!)$
$\rho^{\mathsf{TS}} : \Box(\mathsf{routes}_{\text{¡}} \Rightarrow \Diamond\mathsf{timetables}!)$

Table 1: Correspondence between concepts of relational and service-oriented logic programming

	Relational logic programming	Service-oriented logic programming	
Concept	over a signature $\langle F, P\rangle$	over program expressions	over asynchronous relational networks
Variable	pair (x, F_0)	program variable $pgm\colon eXp$	requires-point $x \in X$
Term	structure $\sigma(t_1, \ldots, t_n)$	program statement	subnetwork determined by a point
			
Clause	universally quantified implication $C \xleftarrow[X]{} H$	program module	service module
			
Query	existentially quantified conjunction $\xrightarrow[X]{} Q$	program query	client application
			
Unification and resolution	term unification and first-order resolution	program discovery and binding (see Figure 2)	service discovery and binding (see Figure 11)

consider, for instance, an encoding of the hypergraphs of processes and connections discussed in this paper into graph-based structures similar to those of [FL13b]; or we could change their underlying institution by adding new temporal modalities (along the lines of Example 3.6) or by considering other classes of automata, like the closed reduced Büchi automata used in [AS87, FL13a]. 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.

Acknowledgements

The work of the first author has been supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0439. The authors also wish to thank Fernando Orejas for suggesting the use of hypergraphs, Antónia Lopes for many useful discussions that led to the present form of this paper, and the anonymous referees for their careful study of the original manuscript.

References

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi A. Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer, 2004.

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2(3):117–126, 1987.

[BCT06] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Representing, analysing and managing Web service protocols. *Data & Knowledge Engineering*, 58(3):327–357, 2006.

[BGLL09] Roberto Bruni, Fabio Gadducci, and Alberto Lluch-Lafuente. A graph syntax for processes and services. In Cosimo Laneve and Jianwen Su, editors, *Web Services and Formal Methods*, volume 6194 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2009.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the* ACM, 30(2):323–342, 1983.

[DF98] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of AMAST *Series in Computing*. World Scientific, 1998.

[Dia08] Răzvan Diaconescu. *Institution-Independent Model Theory*. Studies in Universal Logic. Birkhäuser, 2008.

[FC96] José L. Fiadeiro and José F. Costa. Mirror, mirror in my hand: a duality between specifications and models of process behaviour. *Mathematical Structures in Computer Science*, 6(4):353–373, 1996.

[FHL+05] Gian Luigi Ferrari, Dan Hirsch, Ivan Lanese, Ugo Montanari, and Emilio Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects*, volume 4111 of *Lecture Notes in Computer Science*, pages 22–43. Springer, 2005.

[Fia12] José L. Fiadeiro. The many faces of complexity in software design. In Mike Hinchey and Lorcan Coyle, editors, *Conquering Complexity*, pages 3–47. Springer, 2012.

[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.

[FL13a] José L. Fiadeiro and Antónia Lopes. An interface theory for service-oriented design. *Theoretical Computer Science*, 503:1–30, 2013.

[FL13b] José L. Fiadeiro and Antónia Lopes. A model for dynamic reconfiguration in service-oriented architectures. *Software and Systems Modeling*, 12(2):349–367, 2013.

[FLB07] José L. Fiadeiro, Antónia Lopes, and Laura Bocchi. Algebraic semantics of service component modules. In José L. Fiadeiro and Pierre-Yves Schobbens, editors, *Recent Trends in Algebraic Development Techniques*, volume 4409 of *Lecture Notes in Computer Science*, pages 37–55. Springer, 2007.

[FLB11] José L. Fiadeiro, Antónia Lopes, and Laura Bocchi. An abstract model of service discovery and binding. *Formal Aspects of Computing*, 23(4):433–463, 2011.

[FS07] José L. Fiadeiro and Vincent Schmitt. Structured co-spans: an algebra of interaction protocols. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *Algebra and Coalgebra in Computer Science*, volume 4624 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2007.

[GB92] Joseph A. Goguen and Rod M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the* ACM, 39(1):95–146, 1992.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, second edition, 1994.

[GM96] Joseph A. Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs*. Foundations of computing. MIT Press, 1996.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the* ACM, 12(10):576–580, 1969.

[Ive62] Kenneth E. Iverson. *A Programming Language*. Wiley, 1962.

[KBG07] Srividya Kona, Ajay Bansal, and Gopal Gupta. Automatic composition of semantic Web services. In *2007* IEEE *International Conference on Web Services*, pages 150–158. IEEE Computer Society, 2007.

[KMWZ10] Alexander Knapp, Grzegorz Marczyński, Martin Wirsing, and Artur Zawłocki. A heterogeneous approach to service-oriented systems specification. In Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung, editors, ACM *Symposium on Applied Computing*, pages 2477–2484. ACM, 2010.

[Llo87] John W. Lloyd. *Foundations of Logic Programming*. Symbolic computation: Artificial intelligence. Springer, 1987.

[Löd01] Christof Löding. Efficient minimization of deterministic weak ω-automata. *Information Processing Letters*, 79(3):105–109, 2001.

[Mes89] José Meseguer. General logics. In Heinz-Dieter Ebbinghaus, José Fernández-Prida, Manuel Garrido, Daniel Lascar, and Mario Rodriquez-Artalejo, editors, *Logic Colloquium '87*, volume 129 of *Studies in Logic and the Foundations of Mathematics Series*, pages 275–329. Elsevier, 1989.

[Mor94] Carroll C. Morgan. *Programming from Specifications*. Prentice Hall International series in computer science. Prentice Hall, second edition, 1994.

[Mos02] Till Mossakowski. Comorphism-based Grothendieck logics. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002*, volume 2420 of *Lecture Notes in Computer Science*, pages 593–604. Springer, 2002.

[Mos04] Peter Mosses. CASL *Reference Manual: The Complete Documentation Of The Common Algebraic Specification Language*. Lecture Notes in Computer Science. Springer, 2004.

[MS97] Oded Maler and Ludwig Staiger. On syntactic congruences for ω-languages. *Theoretical Computer Science*, 183(1):93–112, 1997.

[Mul63] David E. Muller. Infinite sequences and finite machines. In 4^{th} *Annual Symposium on Switching Circuit Theory and Logical Design*, pages 3–16. IEEE Computer Society, 1963.

[PP04] Dominique Perrin and Jean-Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics. Elsevier Science, 2004.

[SBFZ07] Jianwen Su, Tevfik Bultan, Xiang Fu, and Xiangpeng Zhao. Towards a theory of web service choreographies. In Marlon Dumas and Reiko Heckel, editors, *Web Services and Formal Methods*, volume 4937 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.

[SML04] Lutz Schröder, Till Mossakowski, and Christoph Lüth. Type class polymorphism in an institutional framework. In José L. Fiadeiro, Peter D. Mosses, and Fernando Orejas, editors, *Recent Trends in Algebraic Development Techniques*, volume 3423 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2004.

[ST88] Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: implementations revisited. *Acta Informatica*, 25(3):233–281, 1988.

[ST11] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011.

[TBG91] Andrzej Tarlecki, Rod M. Burstall, and Joseph A. Goguen. Some fundamental algebraic tools for the semantics of computation. Part 3: Indexed categories. *Theoretical Computer Science*, 91(2):239–264, 1991.

[ŢF13] Ionuţ Ţuţu and José L. Fiadeiro. A logic-programming semantics of services. In Reiko Heckel and Stefan Milius, editors, *Algebra and Coalgebra in Computer Science*, volume 8089 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2013.

[ŢF15] Ionuţ Ţuţu and José L. Fiadeiro. From conventional to institution-independent logic programming. *Journal of Logic and Computation*, in press.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192. Elsevier and MIT Press, 1990.

[Vog03] Werner Vogels. Web services are not distributed objects. IEEE *Internet Computing*, 7(6):59–66, 2003.