



Comparing logics for rewriting: rewriting logic, action calculi and tile logic [☆]

Fabio Gadducci*, Ugo Montanari

Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy

Abstract

The large diffusion of concurrent and distributed systems has spawned in recent years a variety of new formalisms, equipped with features for supporting an easy specification of such systems. The aim of our paper is to analyze three proposals, namely *rewriting logic*, *action calculi* and *tile logic*, chosen among those formalisms designed for the description of rule-based systems. For each of these logics we first try to understand their foundations, then we briefly sketch some applications. The overall goal of our work is to find out a common layout where these logics can be recast, thus allowing for a comparison and an evaluation of their specific features. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Action calculi; Rewriting logic; Tile logic; Term graphs; Algebraic theories; gs-monoidal theories; π -calculus; Concurrent systems; Interactive systems; Mobile systems

1. Introduction

Recent years have seen a large diffusion of rule-based computational systems, in particular supporting concurrent and distributed features. Their operational behaviour could be summarized this way: There exists a world of possible *states*, or *global configurations*, each of them representing an intermediate step in the evaluation of a program. In practice, a configuration should take into account the memory contents,

[☆] Research partly supported by the TMR Network GETGRATS, by the Esprit Working Groups *APPLI-GRAPH*, *CONFER2* and *COORDINA*, and by the MURST projects “Tecniche Formali per Sistemi Software” and “Tipi, Ordine Superiore e Concorrenza”. Research partly carried out during the stay of the first author at the Laboratory for Foundations of Computer Science, Division of Informatics, University of Edinburgh, supported by the British EPSRC Grant R29375.

* Corresponding author.

E-mail addresses: gadducci@di.unipi.it (F. Gadducci), ugo@di.unipi.it (U. Montanari).

the data structures and the flow diagram of an abstract machine implementing the system. Rules denote possible, *local* evolutions of the system, and in many cases some of them may possibly take place simultaneously, either independently or interacting. A suitable notion of *matching* is then (often implicitly) required, for identifying the occurrence of a certain local configuration inside a global one, in order to know when a certain rule can be applied. This is the case of the *firing semantics* for *Petri nets* [73], where a state is a multi-set of atomic components, and a transition σ is allowed to be executed from a *marking* M if the preconditions of σ appear in M ; or the tightly related case of the reaction analogy for the *chemical abstract machine* [6]; or the case of the *equational deduction* view of *term rewriting systems* [46], where states are terms of an algebra, and a rewriting step $l \rightarrow r$ can be performed from a term t if one of its subterms can be equated to an instance of the term l ; or the case of the implementations of λ -calculus based on *permutation equivalence* [51]; or the case of *operational semantics* for *process algebras* [61]; or ...

Such a large diffusion of rule-based systems has spawned a number of proposals, aiming at methodologies that could offer a flexible framework (intended as a metaformalism) for specifying their operational behaviours. Even in this metatheoretical case we may try to identify a pattern: The idea is to abstract as much as possible from the details of the system at hand, offering general guidelines for characterizing inductively the class of all possible evolution steps performed by the system. Such a procedure usually involves some kind of deduction system, usually built by lifting the structure on states to the level of *computations*. The aim of our paper is to analyze three formalisms, namely (*unconditional*) *rewriting logic*, *action calculi* and *tile logic*, representing three of the most complex and articulated proposals available at the moment. They all share a common assumption, namely, that the rules should *carry no condition* restricting their applicability.¹ In the paper we describe the above formalisms, and we try to find a common ground to compare them. In particular, we will take care of showing how both tile logic and action calculi can be encoded into rewriting logic, and in providing benchmarks, via suitable case studies, in order to underline the specific features of each of those approaches.

Rewriting logic [54] has been introduced by José Meseguer as a foundational tool for the specification of concurrent systems with state changes. Since it is also adopted as the semantic basis of several implementation efforts [7,17,27,55], which support either executable specifications or parallel programming in the formalism, it can be considered as a suitable framework in which many other logics can be implemented [53]. In Section 3 we first sketch an introductory description of rewriting logic, restricting our attention to the *unconditional* case. Then, we review briefly two relevant applications of the logic, in view of our specific interest toward the modeling of systems with distributed or higher-order features. Thus, we first present one of the best-known formalisms for distributed systems, namely *Petri nets* [73], and we recall the descrip-

¹ More explicitly, we consider that formalisms such as *conditional rewriting systems*, or languages like *Prolog*, do not fall naturally into the scope of the assumption. In the chosen case studies, instead, the basic rules interact *via* local information, explicitly carried over by the evolution steps, without resorting to any condition on the structure of the deduction.

tion of its concurrent behaviour proposed in [54]. Afterwards, we present an encoding of the *untyped λ -calculus* [3] into rewriting logic; it is a simplified version of the translation described in [49]. We close the section showing how the rewriting logic paradigm can be applied to data structures different from terms.

Action structures were originally proposed by Milner [64] as a foundational model for different kinds of process algebras, especially for mobile and higher-order systems. The focus is on the extension of the algebraic description with functional higher-order features, in the line of the (typed) lambda calculus. In particular, *action calculi* [63] identify a class of such structures, which can be used as a (non-standard) syntax for a variety of models of interactive behaviour. In Section 4 we first present the basic action calculus, and we then review two calculi recently proposed. The first is an example of a higher-order action calculus, called LAMC, introduced in [63], and which is able to encode a variant of the simply-typed, call-by-value λ -calculus. The second is instead a simple action calculus presenting the basic features for the mobility of processes, called PIC, also introduced in [63]; its extension with a primitive for *boxing* — roughly corresponding to *prefixing* — models the asynchronous π -calculus [38]. We close the section presenting an original contribution of the paper, namely, an encoding of (the closed variant of) the generic action calculus into rewriting logic, and proving its soundness.

Tile logic [30] has been proposed by the authors as a general framework for the specification of rule-based systems, whose actual behaviour relies on the notions of *synchronization* and *side-effects*. The main idea is to enrich each rewrite rule with an observation, carrying information on the possible behaviour of its subcomponents, that is, imposing a dynamic constraint to the terms to which it can be applied. Thus, the resulting formalism extends rewriting logic *via* a suitable format for representing generic *open configurations of reactive systems with coordination*. Section 5 has a three-part structure. We start reviewing the basic definitions of tile logic, then presenting two case studies that involve languages for process description. The first example is the encoding of CCS proposed in [30]. The second is the encoding of the asynchronous π -calculus; it is original to this paper, even if it is loosely based on [28]. We close the section sketching the encoding of tile logic into (unconditional) rewriting logic [11,58], and briefly surveying *term tile logic* [10], an extension of the formalism with a rich structure for observations, based on the notion of *hyper-transformation* [12]. We refer instead to [13] for the presentation of a higher-order extension of tile logic.

The connection between the structure of states and that of computations suggests the search for additional algebraic structures besides terms. We give in Section 2.1 some intuition about the notion of *name sharing*. On the various case studies tackled in the paper, names will be considered either as links to communication channels, or to objects, or to locations, or to remote shared resources, or also to some *cause* in the event history of the system. Often, these names are freely α -converted, because the only important information they offer is *sharing*. In Section 2.2 we then introduce *gs-monoidal theories*, which represent the algebraic counterpart of such notions; they give an axiomatic description to various graph-like structures, introducing formalisms that are different from the ordinary tree-like presentation of terms. We often use such

theories in our paper, but we do not aim at any completeness, and we do not give any account of similar structures that have been presented in the literature (see e.g. [23,34,42,43,72]); we refer the interested reader to [9,20].

2. On the structure of terms

2.1. Some thoughts on name sharing

We suggest an informal ‘wire-and-box’ notation for giving an intuitive understanding of the name sharing mechanism, and more generally of the rôle played by the *auxiliary* structure in the ordinary representation of terms. In this notation, variables are represented by wires and the operators of the signature are denoted by boxes labeled with the corresponding operation symbols. For instance, the term $h(x_1, f(x_2), g(x_1, a))$ over the signature $\Sigma_e = \bigcup_{i=0}^3 \Sigma_i$ — where $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{f\}$, $\Sigma_2 = \{g\}$ and $\Sigma_3 = \{h\}$ — and variables x_1, x_2 can be graphically represented as in Fig. 1. Notice that wire duplications (e.g., of x_1) and wire swappings (e.g., of x_2 and a copy of x_1) are auxiliary, in the sense that they belong to *any* wire-and-box model, independently from the underlying signature. The properties of the auxiliary structure are far from trivial and could lead to misleading system representations if their interpretation were not well-formalized.

For example, let us consider the wire-and-box diagrams c_1 and c_2 in Fig. 2. In a *value-oriented* interpretation, both c_1 and c_2 yield the same term $g(a, a)$. Instead, in a *reference-oriented* interpretation, c_1 and c_2 define different situations: In the former case, the two arguments of g are not related, while in the latter case they point to the same *shared* location. The difference becomes more evident if we add the rewriting rules $\{a \Rightarrow b, a \Rightarrow c\}$ for modeling the dynamics of our diagrams. Indeed, while we need two (concurrent) rewrites to transform c_1 into the representation of $g(b, b)$, only one rewrite is needed to apply a similar transformation to c_2 . At the same time, it is impossible to obtain (any representation of) the term $g(b, c)$ from c_2 .

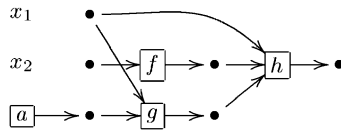


Fig. 1. Example of wire-and-box notation.

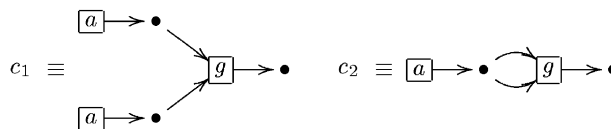


Fig. 2. Example of explicit sharing.

Many mathematical structures have been proposed in the literature to express formalisms different from the ordinary, tree-like presentation of terms. Early examples can be found in the works of German algebraists, such as the *x-categories* of Hotz [39], the *s-monoidal categories* of Pfender [70], and the *dht-categories* of Hönke [36,37]. A common element to all these structures is the fact that they can be thought of as suitable enrichments of *monoidal categories* [52], which give the basis for the description of a distributed environment in terms of a wire-and-box diagram; the enrichment usually relies on the addition of categorical *transformations* with or without the naturality requirement. Mid-1980s studies along this line include the *flownomial calculus* of Ștefănescu [23,24], and the *Petri nets are monoids* approach proposed by Meseguer and Montanari [26,57].

In the next section we introduce an incremental description for *algebraic theories*. Such a presentation allows to capture also other important theories, and we focus our attention on *gs-monoidal theories*. We observe that algebraic and gs-monoidal theories differ only for two axioms, which represent the difference between implicit (as in the ordinary description of terms) and explicit (as it is the case instead in formalisms like *term graphs*,² usually presented with a set-theoretical flavour) sharing of subterms. For more details, we refer to [20] and to a recent joint work of the authors with Bruni [9]; in the latter we propose a schema for describing normal forms for this kind of structures, obtaining a framework where each structure finds a standard representation.

2.2. On theories

We recall here some basic definitions, which are used to recast the usual notion of term over a signature in a more general setting.

Definition 2.1 (*Hyper-signature*). A *many-sorted hyper-signature* Σ over a set S of sorts is a family $\{\Sigma_{w,w'}\}_{w,w' \in S^*}$ of sets of operators, where each operator $f \in \Sigma_{w,w'}$ takes $|w|$ arguments typed according to w , and returns a tuple of $|w'|$ values, typed according to w' .

If S is a singleton, the hyper-signature Σ is called *one-sorted* and it is simply denoted by the family $\{\Sigma_{n,m}\}_{n,m \in \mathbb{N}}$.

Ordinary, many-sorted signatures Σ can be obtained as an instance of the previous definition, by requiring that $\Sigma_{w,w'} \neq \emptyset \Rightarrow w' \in S$. Moreover, if S is a singleton, the signature is *one-sorted*, and it is denoted by the family $\{\Sigma_n\}_{n \in \mathbb{N}}$.

We can think of a many-sorted hyper-signature Σ of operators as some sort of graph structure: Its nodes are the elements of the free monoid S^* , while its edges are the operators, such that $f : w \rightarrow w'$ if and only if $f \in \Sigma_{w,w'}$. Exploiting this intuition, we can give an inductive, step-by-step account of the usual algebraic notion of term, by means

² Roughly, term graphs are graphs whose nodes are labeled by operators of a signature, as defined e.g. in the introductory chapter of [75]. They are a reference-oriented generalization of the ordinary (value-oriented) notion of term, where the sharing of subterms can be specified also for closed (i.e., without variables) terms. Indeed, terms can share variables, but shared subterms of a closed term can be freely copied, always yielding an equivalent term.

of a chain of structures of increasing complexity. For the sake of readability, in this section we stick to one-sorted hyper-signatures, where arities are elements of the set \mathbb{N}_c of (underlined) natural numbers, the monoid operation is defined as $\underline{n} \otimes \underline{m} = \underline{n + m}$, and $\underline{0}$ is the neutral element. Given a hyper-signature, we can inductively characterize suitable monoids of *arrows*. Each arrow is equipped with *source* and *target* functions, with value in \mathbb{N}_c , and it is usually written as $t : \underline{n} \rightarrow \underline{m}$. Roughly, such an arrow denotes a structure t (a diagram in the wire-and-box model), with \underline{n} distinct variables occurring in it, and with \underline{m} ports from which it can be accessed.

Definition 2.2 (*Graph theories*). Given a hyper-signature Σ , the set of arrows of the *graph theory* $G(\Sigma)$ is generated by the following inference rules:

$$\begin{array}{l} \text{(generators)} \frac{f \in \Sigma_{n,m}}{f : \underline{n} \rightarrow \underline{m} \in G(\Sigma)} \quad \text{(pairing)} \frac{s : \underline{n} \rightarrow \underline{m}, t : \underline{n}' \rightarrow \underline{m}'}{s \otimes t : \underline{n} \otimes \underline{n}' \rightarrow \underline{m} \otimes \underline{m}'} \\ \text{(identities)} \frac{\underline{n} \in \mathbb{N}_c}{id_{\underline{n}} : \underline{n} \rightarrow \underline{n}} \end{array}$$

Moreover, the pairing operator is associative, $id_{\underline{0}}$ is the neutral element of the resulting monoid of arrows, and the *monoidality* axiom $id_{\underline{n} \otimes \underline{m}} = id_{\underline{n}} \otimes id_{\underline{m}}$ holds for all $\underline{n}, \underline{m} \in \mathbb{N}_c$.

Given a hyper-signature Σ , a *graph Σ -equation* is a sentence of the form $s = t$ with $s, t : \underline{n} \rightarrow \underline{m}$ arrows in $G(\Sigma)$. Given a set E of such equations, we denote by $G_E(\Sigma)$ the graph theory for the pair (Σ, E) : Its objects are the elements of \mathbb{N}_c ; and its arrows are the equivalence classes of those in $G(\Sigma)$, modulo the congruence generated by the set E of equations.

Given $G_E(\Sigma), G_{E'}(\Sigma')$ graph theories, a *graph theory morphism* $F : G_E(\Sigma) \rightarrow G_{E'}(\Sigma')$ is a monoid morphism, preserving source and target.

We usually denote the E -equivalence class of an arrow t by $[t]_E$, or just $[t]$. Graph theories simply equip a signature with an explicit notion of *pairing* (or *tupling*), modulo equations: Each arrow denotes an array of operators and identities. This theory is rather inexpressive, per se; we will use it only in conjunction with the rewriting mechanism to be introduced later on.

Definition 2.3 (*Monoidal theories*). Given a hyper-signature Σ , the set of arrows of the *monoidal theory* $\mathbf{M}(\Sigma)$ is generated by the following inference rules:

$$\begin{array}{l} \text{(generators)} \frac{f \in \Sigma_{n,m}}{f : \underline{n} \rightarrow \underline{m} \in \mathbf{M}(\Sigma)} \quad \text{(pairing)} \frac{s : \underline{n} \rightarrow \underline{m}, t : \underline{n}' \rightarrow \underline{m}'}{s \otimes t : \underline{n} \otimes \underline{n}' \rightarrow \underline{m} \otimes \underline{m}'} \\ \text{(identities)} \frac{\underline{n} \in \mathbb{N}_c}{id_{\underline{n}} : \underline{n} \rightarrow \underline{n}} \quad \text{(composition)} \frac{s : \underline{n} \rightarrow \underline{m}, t : \underline{m} \rightarrow \underline{k}}{s; t : \underline{n} \rightarrow \underline{k}} \end{array}$$

Moreover, the composition operator is associative (whenever the corresponding compositions are defined), the pairing operator is associative with $id_{\underline{0}}$ the neutral element of the resulting monoid of arrows, and the monoidality axiom $id_{\underline{n} \otimes \underline{m}} = id_{\underline{n}} \otimes id_{\underline{m}}$ holds

for all $\underline{n}, \underline{m} \in \mathbb{N}_c$. In addition, the monoid of arrows satisfies the *functoriality* axiom

$$(s \otimes t); (s' \otimes t') = (s; s') \otimes (t; t')$$

(whenever both sides are defined) and the *identity* axiom $id_{\underline{n}}; s = s = s; id_{\underline{m}}$ for all $s : \underline{n} \rightarrow \underline{m}$.

Given a hyper-signature Σ , a *monoidal Σ -equation* is a sentence of the form $s = t$ with $s, t : \underline{n} \rightarrow \underline{m}$ arrows in $\mathbf{M}(\Sigma)$. Given a set E of such equations, we denote by $\mathbf{M}_E(\Sigma)$ the monoidal theory for the pair (Σ, E) : Its objects are the elements of \mathbb{N}_c ; and its arrows are the equivalence classes of those in $\mathbf{M}(\Sigma)$, modulo the congruence generated by the set E of equations.

Given $\mathbf{M}_E(\Sigma), \mathbf{M}_{E'}(\Sigma')$ monoidal theories, a *monoidal theory morphism* $F : \mathbf{M}_E(\Sigma) \rightarrow \mathbf{M}_{E'}(\Sigma')$ is a graph theory morphism, preserving composition.

Monoidal theories consider a monoid of arrows equipped with an explicit operation of composition. Thanks to the functoriality axiom, any arrow in $\mathbf{M}(\Sigma)$ can indeed be written as a sequential composition of concrete arrows already ‘appearing’ in the underlying graph theory. A monoidal theory is just an example of a (strict) monoidal category (it is, in fact, the free strict monoidal category generated by Σ), and for these categories many representation results are well-known. Now we introduce the more expressive kind of theories we deal with in our paper, *gs-monoidal theories*.

Definition 2.4 (*GS-monoidal theories*). Given a hyper-signature Σ , the set of arrows of the *gs-monoidal theory* $\mathbf{GS}(\Sigma)$ is generated by the following inference rules:

$$\begin{array}{ll} \text{(generators)} \quad \frac{f \in \Sigma_{n,m}}{f : \underline{n} \rightarrow \underline{m} \in \mathbf{GS}(\Sigma)} & \text{(pairing)} \quad \frac{s : \underline{n} \rightarrow \underline{m}, t : \underline{n}' \rightarrow \underline{m}'}{s \otimes t : \underline{n} \otimes \underline{n}' \rightarrow \underline{m} \otimes \underline{m}'} \\ \text{(identities)} \quad \frac{\underline{n} \in \mathbb{N}_c}{id_{\underline{n}} : \underline{n} \rightarrow \underline{n}} & \text{(composition)} \quad \frac{s : \underline{n} \rightarrow \underline{m}, t : \underline{m} \rightarrow \underline{k}}{s; t : \underline{n} \rightarrow \underline{k}} \\ \text{(duplicators)} \quad \frac{\underline{n} \in \mathbb{N}_c}{\nabla_{\underline{n}} : \underline{n} \rightarrow \underline{n} \otimes \underline{n}} & \text{(dischargers)} \quad \frac{\underline{n} \in \mathbb{N}_c}{!_{\underline{n}} : \underline{n} \rightarrow \underline{0}} \\ \text{(permutations)} \quad \frac{\underline{n}, \underline{m} \in \mathbb{N}_c}{\rho_{\underline{n}, \underline{m}} : \underline{n} \otimes \underline{m} \rightarrow \underline{m} \otimes \underline{n}} & \end{array}$$

Moreover, the above operations satisfy the axioms stated in Definition 2.3 for monoidal theories, plus the additional *monoidality* axioms

$$\begin{aligned} \rho_{\underline{n} \otimes \underline{m}, \underline{l}} &= (id_{\underline{n}} \otimes \rho_{\underline{m}, \underline{l}}); (\rho_{\underline{n}, \underline{l}} \otimes id_{\underline{m}}) \\ !_{\underline{n} \otimes \underline{m}} &= !_{\underline{n}} \otimes !_{\underline{m}} \\ \nabla_{\underline{n} \otimes \underline{m}} &= (\nabla_{\underline{n}} \otimes \nabla_{\underline{m}}); (id_{\underline{n}} \otimes \rho_{\underline{n}, \underline{m}} \otimes id_{\underline{m}}) \\ !_{\underline{0}} &= \nabla_{\underline{0}} = \rho_{\underline{0}, \underline{0}} = id_{\underline{0}} \end{aligned}$$

for all $\underline{n}, \underline{m}, \underline{l} \in \mathbb{N}_c$; the *coherence* axioms

$$\nabla_{\underline{n}}; (id_{\underline{n}} \otimes \nabla_{\underline{n}}) = \nabla_{\underline{n}}; (\nabla_{\underline{n}} \otimes id_{\underline{n}}) \quad \nabla_{\underline{n}}; \rho_{\underline{n}, \underline{n}} = \nabla_{\underline{n}}$$

$$\nabla_{\underline{n}}; (id_{\underline{n}} \otimes !_{\underline{n}}) = id_{\underline{n}} \quad \rho_{\underline{n}, \underline{m}}; \rho_{\underline{m}, \underline{n}} = id_{\underline{n}} \otimes id_{\underline{m}}$$

for all $\underline{n}, \underline{m} \in \mathbb{N}_c$; and the *naturality* axiom

$$(s \otimes t); \rho_{\underline{m}, \underline{k}} = \rho_{\underline{n}, \underline{l}}; (t \otimes s)$$

for all $s : \underline{n} \rightarrow \underline{m}, t : \underline{l} \rightarrow \underline{k}$.

Given a hyper-signature Σ , a *gs-monoidal Σ -equation* is a sentence of the form $s = t$ with $s, t : \underline{n} \rightarrow \underline{m}$ arrows in $\mathbf{GS}(\Sigma)$. Given a set E of such equations, we denote by $\mathbf{GS}_E(\Sigma)$ the monoidal theory for the pair (Σ, E) : Its objects are the elements of \mathbb{N}_c ; and its arrows are the equivalence classes of those in $\mathbf{GS}(\Sigma)$, modulo the congruence generated by the set E of equations.

Given $\mathbf{GS}_E(\Sigma), \mathbf{GS}_{E'}(\Sigma')$ gs-monoidal theories, a *gs-monoidal theory morphism* $F : \mathbf{GS}_E(\Sigma) \rightarrow \mathbf{GS}_{E'}(\Sigma')$ is a monoidal theory morphism, preserving also permutations, duplicators and dischargers.

The gs-monoidal theory $\mathbf{GS}(\Sigma)$ is an example of a symmetric strict monoidal category. The additional structure, namely the operators ∇ and $!$, allows for a (controlled) form of *duplication* and *discharge* of data. The enriched structure falls short of the definition of a *cartesian category* only because of two missing axioms, imposing the satisfaction of naturality also for these operators.

Definition 2.5 (*Algebraic theories*). Given a hyper-signature Σ , the *algebraic theory* $\mathbf{A}(\Sigma)$ is the gs-monoidal theory $\mathbf{GS}_{E_n}(\Sigma)$, where E_n is the set of gs-monoidal equations expressing the naturality of duplicators and dischargers, namely

$$s; \nabla_{\underline{m}} = \nabla_{\underline{n}}; (s \otimes s), \quad s; !_{\underline{m}} = !_{\underline{n}}$$

for all $s : \underline{n} \rightarrow \underline{m}$.

It can be considered part of the categorical folklore the fact that the cartesian product canonically induces a monoidal product, together with a family of suitable natural transformations, usually denoted as *diagonals* and *projections* (related papers range from [37,70] to the more recent [41,48]). Then, our definition of algebraic theory can be proved equivalent to the classical one, dating back to the early work of Lawvere [47,50]. The following, classical result states the equivalence between these theories and the usual term algebra construction for ordinary signatures.

Proposition 2.1 (*Algebraic theories and term algebras*). *Given an ordinary signature Σ , for all $\underline{n}, \underline{m} \in \mathbb{N}_c$ there exists a one-to-one correspondence between the set of arrows with source \underline{n} and target \underline{m} of $\mathbf{A}(\Sigma)$ and the m -tuples of elements of the term algebra—over a set of n variables—associated to Σ .*

The previous result states that each arrow $t : \underline{n} \rightarrow \underline{1}$ uniquely identifies an element of the term algebra over the set $\{x_1, \dots, x_n\}$. An arrow $t : \underline{n} \rightarrow \underline{m}$ is an m -tuple of such elements, and composition is term substitution. Note that this correspondence *requires* that both ∇ and $!$ are natural; that is, gs-monoidal theories are in fact too concrete, distinguishing elements that intuitively represent the same term. In fact, a fundamental property of correspondence can be shown between gs-monoidal theories (over ordinary signatures) and term graphs: Each arrow $t : \underline{n} \rightarrow \underline{m}$ identifies a term graph over Σ with a specified m -tuple of roots and a specified n -tuple of variable nodes, and composition is graph replacement [19]. The acronym *gs* stands indeed for *graph substitution*.

Example 2.1 (Terms and theories). We consider again the signature $\Sigma_e = \bigcup_{i=0}^3 \Sigma_i$, where $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{f\}$, $\Sigma_2 = \{g\}$ and $\Sigma_3 = \{h\}$. Some of the arrows in $\mathbf{GS}(\Sigma_e)$ are $a; f : \underline{0} \rightarrow \underline{1}$, $(a \otimes f); g : \underline{1} \rightarrow \underline{1}$ and $a; \nabla_1; (f \otimes \nabla_1); h : \underline{0} \rightarrow \underline{1}$. They correspond to terms $f(a)$, $g(a, f(x))$ and $h(f(a), a, a)$, respectively, for a given variable x . Such a correspondence cannot be pushed too far. For example, both $c_1 = (a \otimes a); g$ and $c_2 = a; \nabla_1; g$ are arrows of $\mathbf{GS}(\Sigma_e)$; they correspond to the same term $g(a, a)$, but they are different as elements of $\mathbf{GS}(\Sigma_e)$, while they are identified by the naturality axiom for ∇ in $\mathbf{A}(\Sigma_e)$. Fig. 2 offers an intuitive pictorial description of such a difference.

The incremental description of these theories, and in particular the relevance of the computational interpretation of *gs*-monoidal theories, plays an important rôle in the description of the various encodings presented in the following sections. In fact, the main point of our discussion is that algebraic (and a fortiori *gs*-monoidal) theories allow for a description of terms which is far more general, and at the same time more concrete, than their ordinary description as elements of a term algebra, separating in a better way the ‘ Σ -structure’ from the additional algebraic structure that the *meta-operators* used in the set-theoretic presentation of term algebras (like *substitution*) implicitly enjoy [19,21].

3. Rewriting logic

3.1. The unconditional, one-sorted rewriting logic

We assume the reader to be familiar with the usual, set-theoretic presentation of algebraic specifications. In particular, given a signature Σ and a set of variables X , then $T_\Sigma(X)$ denotes the free algebra over X . Given an ordering $\{x_1, x_2, \dots\}$ over variables, the set of arrows $t : \underline{n} \rightarrow \underline{1}$ of the algebraic theory $\mathbf{A}(\Sigma)$ coincides with $T_\Sigma(\{x_1, \dots, x_n\})$ for all $n \in \mathbb{N}$, as suggested by Proposition 2.1. If E is a set of Σ -equations, that is, sentences of the form $s = t$ with $s, t \in T_\Sigma(X)$, then $T_{\Sigma, E}(X)$ denotes the free algebra over X of the equational variety induced by E .³

³ A result analogous to Proposition 2.1 holds, but some care is needed in the translation of the Σ -equations. If m is the number of distinct variables in X occurring in either s or t , then such an equation identifies a *gs*-monoidal equation $s' = t'$, with $s', t' : \underline{m} \rightarrow \underline{1}$, such that $[s']_{E_n} = s$ and $[t']_{E_n} = t$.

Table 1
Inference rules for sequents

<i>Reflexivity:</i>	$\frac{[t] \in T_{\Sigma, E}(X)}{[t] \Rightarrow [t]}$
<i>Congruence:</i>	$\frac{[s_1] \Rightarrow [t_1], \dots, [s_n] \Rightarrow [t_n], f \in \Sigma_n}{[f(s_1, \dots, s_n)] \Rightarrow [f(t_1, \dots, t_n)]}$
<i>Replacement:</i>	$\frac{[v_1] \Rightarrow [w_1], \dots, [v_n] \Rightarrow [w_n], r : [s(x_1, \dots, x_n)] \Rightarrow [t(x_1, \dots, x_n)] \in R}{[s(\vec{v}/\vec{x})] \Rightarrow [t(\vec{w}/\vec{x})]}$
<i>Transitivity:</i>	$\frac{[t_1] \Rightarrow [t_2], [t_2] \Rightarrow [t_3]}{[t_1] \Rightarrow [t_3]}$

Definition 3.1 (*Rewrite theories*). A *labeled rewrite theory* \mathcal{R} is a 4-tuple $\langle \Sigma, E, L, R \rangle$ where Σ is a signature, E is a set of Σ -equations, L is a set of *labels*, and $R \subseteq L \times T_{\Sigma, E}(X) \times T_{\Sigma, E}(X)$ is a set of *labeled rewrite rules*. For $(r, [s], [t]) \in R$ we use the notation $r : [s] \Rightarrow [t]$.

Rewrite rules in \mathcal{R} may be understood as the basic *sequents* of a theory, that is, the building blocks of the actual rewrite relation. More complex deductions can be obtained by a finite application of four simple rules.

Definition 3.2 (*Sequents*). Let $\mathcal{R} = \langle \Sigma, E, L, R \rangle$ be a rewrite theory. We say that \mathcal{R} *entails* a *sequent* $[s] \Rightarrow [t]$, written $\mathcal{R} \vdash [s] \Rightarrow [t]$, if and only if $[s] \Rightarrow [t]$ can be obtained by a finite number of applications of the inference rules in Table 1, where $t(\vec{w}/\vec{x})$ denotes the *simultaneous* substitution of w_i for x_i in t .

A rewrite theory is just a *static description* of ‘what a system can do’; the *behaviour* of the theory is instead given by the rewrite relation induced by the rules of deduction. The deduction system in Table 1 was introduced in [54], and it is only one of the possible, equivalent ways to entail the same class of sequents. It has, however, the advantage of being rather intuitive. Maybe the most interesting rule is *replacement*. First, it implies that the transition relation is *stable*, that is, it is closed under substitution. Moreover, the associated sequent describes the *simultaneous* execution of *nested* rewrites: Two subterms matching the left-hand sides of two rules can be rewritten in parallel even if they are not disjoint, provided that they do not overlap.

Indeed, in [54] Meseguer takes advantage of the correspondence between *deductions* in rewriting logic and (*concurrent*) *computations*, defining a *model* for a rewrite theory as a system whose states are E -equivalence classes of Σ -terms, and whose transitions are equivalence classes of terms representing *proofs* in rewriting deduction, that is, concurrent rewrites using the rules in R . The rules for generating such *proof terms* are obtained from the inference rules of Definition 3.2 by decorating the sequents, thus encoding a *justification* of the rewrite. The topic is out of the scope of the present

discussion; we refer to [54] for the definition of the original semantics, and to [21] for a discussion of the subtleties connected to the presence of *non-linear* rules.

3.2. Review of applications

3.2.1. At the basis of concurrency: on Petri nets

Introduced by Carl Adam Petri in his dissertation [69], *Petri nets* are a foundational formalism for the specification of distributed systems. They assume the existence of a set of atomic resources, the *places*, with the intended meaning that a collection of places represents a state of the system. The evolution is described by (the closure of) a set of *transitions*, stating possible local changes in the distribution of resources.

Given a set S , we denote as S^\oplus the free commutative monoid over S and, accordingly to the literature on nets, we call *markings* its elements, that is, the (finite) multisets over S . We can now recall the description of the behaviour for so-called *place/transition* nets.

Definition 3.3 (*Place/Transition nets*). A *place/transition net* N is a 4-tuple $\langle S_N, T_N, \delta_0, \delta_1 \rangle$ such that S_N and T_N are sets of *places* and *transitions*, respectively, while $\delta_0, \delta_1 : T_N \rightarrow S_N^\oplus$ are functions, denoting the source and target of each transition.

Given a *transition* t and a marking M for S_N , a *firing step* is a triple $M[t]M'$ such that M *enables* t (that is, $\delta_0(t) \subseteq M$), and $M' = (M \ominus \delta_0(t)) \oplus \delta_1(t)$. A *firing sequence* is a tuple $M[\sigma]M'$, for $\sigma = \langle t_1, \dots, t_n \rangle$, such that $M_{i-1}[t_i]M_i$ and $M_0 = M$, $M_n = M'$. We usually denote by $M \Rightarrow M'$ the presence of a firing sequence $M[\sigma]M'$.

The multiset union \oplus and subtraction \ominus operators are defined pointwise (note that \ominus is a partial operation), and the partial order \subseteq over markings is just multiset inclusion.

We already remarked that rewriting logic offers a flexible framework where other formalisms for the specification of concurrent systems can be expressed and implemented, thus exploiting its tools for executability. In the case at hand, we just need to state the associativity, commutativity and existence of an identity \hat{e} for the binary operator $\hat{\oplus}$, representing multiset union.

Definition 3.4 (*One-sorted signature for a Petri net*). The one-sorted signature Σ_N associated to a Petri net N contains the constants $\{a : \underline{0} \rightarrow \underline{1} \mid a \in S_N\}$, denoting the places of the net; the constant $\hat{e} : \underline{0} \rightarrow \underline{1}$, denoting the empty marking; and the sum $\hat{\oplus} : \underline{2} \rightarrow \underline{1}$, denoting multiset union. Moreover, given a set of variables X , the sum operator satisfies the set E_{ACI} of axioms, namely

$$x \hat{\oplus} (y \hat{\oplus} z) = (x \hat{\oplus} y) \hat{\oplus} z \quad x \hat{\oplus} y = y \hat{\oplus} x \quad x \hat{\oplus} \hat{e} = x$$

(with $\hat{\oplus}$ used in infix notation) for $x, y, z \in X$.

The order of application of the operator $\hat{\oplus}$ is immaterial, since it satisfies the associativity axiom. Thus, there is an obvious bijection between multisets of places and

arrows with source $\underline{0}$ and target $\underline{1}$. It associates to each multiset $M = \{a_1, \dots, a_n\}$ an arrow $[s_M]: \underline{0} \rightarrow \underline{1}$, defined informally as $a_1 \hat{\oplus} \dots \hat{\oplus} a_n$.

Definition 3.5 (*Rewrite theory for a Petri net*). The rewrite theory \mathcal{R}_N for a Petri net $N = \langle S_N, T_N, \delta_0, \delta_1 \rangle$ is the 4-tuple $\langle \Sigma_N, E_{ACI}, L_N, R_N \rangle$, for $L_N = T$ and $R_N = \{t: [s_{\delta_0(t)}] \rightarrow [s_{\delta_1(t)}] \mid t \in T_N\}$.

We can now state a simple correspondence result between the firing semantics of a net, and the sequents entailed by the associated theory. We do not discuss here the algebraic description of the *process semantics* for nets, and we refer the reader to the seminal [57].⁴

Proposition 3.1 (*Correspondence for firing semantics*). *Given a Petri net N and a marking M for S_N , $M \Rightarrow M'$ holds if and only if the sequent $[s_M] \Rightarrow [s_{M'}]: \underline{0} \rightarrow \underline{1}$ is entailed by \mathcal{R}_N .*

Alternatively, the multiset structure of markings could be recovered by considering each place as an operation with source and target $\underline{0}$, and then analyzing its associated monoidal theory; this is reflected in the presentation of *zero-safe nets* [14] given by Bruni and Montanari. Likewise, also the firing step semantics could be recovered extending the rewriting logic paradigm to monoidal theories; we defer such an extension to Section 3.3.

3.2.2. Encoding substitution: on the λ -calculus

As Petri nets represent a foundational paradigm for distributed computations, the *untyped λ -calculus* is universally acknowledged as a canonical representative for functional languages. Roughly, each element t of the language represents a function, and the *binding operator* $\lambda x.t$ informally denotes the same function of t , abstracting on the formal parameter x . The presentation of (untyped) λ -terms is usually given with a standard algebraic construction, while the reduction mechanism relies on the meta-operation of *substitution*, replacing each occurrence of a formal parameter x in a term t with another term s . To fit the whole presentation in a purely algebraic style, we must therefore recast substitution as just another operator, and write equational axioms which reduce every term to an equivalent one which is substitution-free.

Definition 3.6 (*Many-sorted signature for the λ -calculus*). The signature $\Sigma_{\lambda e}$ associated to the λ -calculus contains the sorts \mathcal{N} and \mathcal{A} , corresponding to names and terms, respectively; the constants $x, y, \dots : \varepsilon \rightarrow \mathcal{N}$, denoting a denumerable set of names; the coercion operator $c : \mathcal{N} \rightarrow \mathcal{A}$, stating that a name is also a term; the *abstraction*

⁴Note that a Petri net closely resembles a hyper-signature, modulo the commutativity of the multiset operator. In fact, considering *net processes* as terms of the (symmetric or strictly symmetric) monoidal theory associated to a net, lies at the heart of the *Petri nets are monoids* approach [26,59]. See also Section 3.3.

operator $\lambda - . - : \mathcal{N} \times \mathcal{A} \rightarrow \mathcal{A}$; the (function) application operator $- \cdot - : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$; and finally, (the encoding of) the substitution operator $-[-/_-] : \mathcal{A} \times \mathcal{A} \times \mathcal{N} \rightarrow \mathcal{A}$.

The standard syntax of λ -calculus is obtained via the restriction of the signature $\Sigma_{\lambda e}$ to the subsignature Σ_λ , obtained by throwing away the substitution operator, and quotienting terms with respect to α -conversion, stating that bound names are immaterial in determining the semantics. The unique reduction rule is the β -reduction rule $(\lambda x.t)s \rightarrow t\{s/x\}$, defined over terms of Σ_λ and where $\{s/x\}$ indicates the (meta-operation of) substitution of the occurrences of the name x in t for the term s . In order to find a rewrite theory for terms in $\Sigma_{\lambda e}$, we need first to encode α -convertibility as a suitable axiom, and to ensure that the substitution operator behaves in the intended way.

Let $X = X_A$ be a set of variables of sort A , ranged over by M, N, O, \dots , and let fn be the usual function computing the free names of a ground term of sort A . Given a ground term t , the axiom defining α -conversion is

$$\lambda x.t = \lambda y.(t[y/x]) \quad \text{for } y \notin fn(t).$$

The remaining axioms allow for reducing every ground term to one built up without using substitution. As for the one encoding α -conversion, they are just schemata, i.e., they represent a denumerable set of axioms. We refer to [49] for a conditional, finitary presentation inducing the same equivalence relation.

$$\begin{aligned} c(x)[^M/x] &= M & c(x)[^M/y] &= c(x) \quad \text{for } x \neq y \\ (M \cdot N)[^O/x] &= (M[^O/x]) \cdot (N[^O/x]) \\ (\lambda x.t_1)[^t_2/y] &= \lambda x.(t_1[^t_2/y]) \quad \text{for } x \neq y \quad \text{and } x \notin fn(t_2) \end{aligned}$$

We denote by $=_\lambda$ the congruence generated by the set E_λ of axioms above, where t_1, t_2 are ground terms of type A . A term $t \in T_{\Sigma_{\lambda e}}(X)$ is in *standard form* if it is built without any operation of substitution, that is, if $t \in T_{\Sigma_\lambda}(X)$.

Proposition 3.2. *Given a ground term t of sort A , there exists a ground term \tilde{t} in standard form such that $[t]_{E_\lambda} = [\tilde{t}]_{E_\lambda}$.*

β -reduction can now be simulated over the terms in $\Sigma_{\lambda e}$ via a denumerable set of rules, using the substitution operator.

Definition 3.7 (Rewrite theory for the λ -calculus). The rewrite theory \mathcal{R}_λ for λ -calculus is the 4-tuple $\langle \Sigma_{\lambda e}, E_\lambda, L_\lambda, R_\lambda \rangle$, such that $L_\lambda = \{\beta_x \mid x \in \mathcal{N}\}$ and $R_\lambda = \{\beta_x : (\lambda x.M) \cdot N \rightarrow M[^N/x] \mid x \in \mathcal{N}\}$.

The precise correspondence statement between the semantics can be found in [49]. We just point out that the main result of that paper is not the presentation of the β -reduction via an ordinary calculus (a topic well-explored in the so-called *explicit substitution* framework [1]), but the analysis of the concurrent semantics induced over the reduction steps by the axioms of rewriting logic, *via* a comparison with the *permutation equivalence* proposed by Lévy [40,51]. Roughly, the main point is that the

rewriting steps are too concrete. For example, forgetting for the sake of readability the coercion operator and the subscript E_λ , let us consider the terms $(\lambda x.x) \cdot z$ and $(\lambda y.y) \cdot z$, belonging to the same equivalence class. The sequent $[(\lambda x.x) \cdot z] \rightarrow [z]$ can then be obtained via two different choices of the formal parameter for the binding operator, and the associated proof terms are therefore different.⁵ Since the two reductions act on the same term, there is no reason for the two proofs not being equated. However, the expected equivalence cannot be derived by the standard axiomatization proposed for rewriting logic, and an additional set of equations has to be added, simulating α -conversion for proof terms; we refer again the reader to [49].

3.3. Extending the paradigm to non-cartesian structures

The deduction rules presented in Table 1 make clear that the underlying idea of the rewriting logic paradigm is that the rewrite relation has to be built inductively, lifting to computations the structure of terms. Such an intuition can be exploited to describe suitable notions of computation also over structures other than terms: In particular, over elements of gs-monoidal theories, as for the deduction system presented in this section.

Definition 3.8 (*GS-monoidal rewrite theories*). A labeled, gs-monoidal rewrite theory \mathcal{R} is a 4-tuple $\langle \Sigma, E, L, R \rangle$ where Σ is a hyper-signature, E is a set of gs-monoidal Σ -equations, L is a set of labels, and $R \subseteq L \times \mathbf{GS}_E(\Sigma) \times \mathbf{GS}_E(\Sigma)$ is a set of labeled, gs-monoidal rewrite rules. For each $\langle r, [s], [t] \rangle \in R$, we assume that $[s], [t] : \underline{n} \rightarrow \underline{m}$, and we use the notation $r : [s] \Rightarrow [t] : \underline{n} \rightarrow \underline{m}$.

The explicit mention in gs-monoidal rules of the source and target of the arrows is the counterpart of the implicit assumption on the number of variables occurring in the rules of Definition 3.1. To further stress this intuition, gs-monoidal rewrite rules in \mathcal{R} may also be understood as basic sequents, while more complex deductions can be obtained by a finite application of five rules.

Definition 3.9 (*GS-monoidal sequents*). Let $\mathcal{R} = \langle \Sigma, E, L, R \rangle$ be a gs-monoidal rewrite theory. We say that \mathcal{R} entails a gs-monoidal sequent $[s] \Rightarrow [t] : \underline{n} \rightarrow \underline{m}$, written $\mathcal{R} \vdash_{\text{gs}} [s] \Rightarrow [t] : \underline{n} \rightarrow \underline{m}$, if and only if $[s] \Rightarrow [t] : \underline{n} \rightarrow \underline{m}$ can be obtained by a finite number of applications of the inference rules in Table 2.

Of course, the deduction system we just presented is also valid for rewriting over monoidal theories: Since we are not interested in the eventual structure of proof terms, we just need to change the premise of the *reflexivity* rule, restricting the attention to terms in $\mathbf{M}_E(\Sigma)$. The system in Table 2 induces over terms the same rewrite relation as the one defined in Table 1 for algebraic sequents, since algebraic theories are just

⁵ Informally, the proof terms are given by $\beta_x(c(id_x), c(id_z))$ and $\beta_y(c(id_y), c(id_z))$. They are syntactically different, albeit describing the same reduction up to α -conversion in the (equivalence class of the) λ -term $(\lambda x.x) \cdot z$.

Table 2
Inference rules for gs-monoidal sequents

Reflexivity:

$$\frac{[t] : \underline{n} \rightarrow \underline{m} \in \mathbf{GS}_E(\Sigma)}{[t] \Rightarrow [t] : \underline{n} \rightarrow \underline{m}}$$

Generators:

$$\frac{r : [s] \Rightarrow [t] : \underline{n} \rightarrow \underline{m} \in R}{[s] \Rightarrow [t] : \underline{n} \rightarrow \underline{m}}$$

Pairing:

$$\frac{[s_1] \Rightarrow [t_1] : \underline{n}_1 \rightarrow \underline{m}_1, [s_2] \Rightarrow [t_2] : \underline{n}_2 \rightarrow \underline{m}_2}{[s_1 \otimes s_2] \Rightarrow [t_1 \otimes t_2] : \underline{n}_1 \otimes \underline{n}_2 \rightarrow \underline{m}_1 \otimes \underline{m}_2}$$

Composition:

$$\frac{[s_1] \Rightarrow [t_1] : \underline{n}_1 \rightarrow \underline{m}_1, [s_2] \Rightarrow [t_2] : \underline{n}_2 \rightarrow \underline{m}_2, \underline{m}_1 = \underline{n}_2}{[s_1; s_2] \Rightarrow [t_1; t_2] : \underline{n}_1 \rightarrow \underline{m}_2}$$

Transitivity:

$$\frac{[t_1] \Rightarrow [t_2] : \underline{n} \rightarrow \underline{m}, [t_2] \Rightarrow [t_3] : \underline{n} \rightarrow \underline{m}}{[t_1] \Rightarrow [t_3] : \underline{n} \rightarrow \underline{m}}$$

gs-monoidal theories plus the naturality axioms E_n , that is, $\mathbf{A}_E(\Sigma) = \mathbf{GS}_{E \cup E_n}(\Sigma)$. The correspondence result between the two deduction systems is explicitly given by the following proposition, stated here only for rewrite theories with an empty set of axioms.

Proposition 3.3 (From algebraic to gs-monoidal sequents). *Let $\mathcal{R} = \langle \Sigma, \emptyset, L, R \rangle$ be a rewrite theory, and let $\mathcal{R}_{gs} = \langle \Sigma, E_n, L, R \rangle$ be the associated gs-monoidal rewrite theory. Furthermore, let s, t be terms over n variables, and let $[s], [t] : \underline{n} \rightarrow \underline{1}$ be the corresponding elements of $\mathbf{GS}_{E_n}(\Sigma)$ (whose existence is assured by Proposition 2.1).*

- *If $\mathcal{R} \vdash s \Rightarrow t$, then $\mathcal{R}_{gs} \vdash_{gs} [s] \Rightarrow [t] : \underline{n} \rightarrow \underline{1}$;*
- *if $\mathcal{R}_{gs} \vdash_{gs} [s] \Rightarrow u : \underline{n} \rightarrow \underline{1}$, then there exists $t \in T_{\Sigma}(\{x_1, \dots, x_n\})$ such that $\mathcal{R} \vdash s \Rightarrow t$ and $u = [t]$.*

It is relevant the fact that also a converse translation exists. It is enough to introduce explicit operators and rules for encoding the gs-monoidal structure into an algebraic theory, as it is the case for the monoidal structure of Petri nets. The explicit construction is similar to the encoding of controls for action calculi *via* control signatures, presented in Definition 4.9, and to the encoding of tile logic into rewriting logic, recalled on Section 5.3. They all use a formalism for algebraic specification called (*partial*) membership equational logic [56], which is able to specify partial algebras with overloading and operations with equationally specified domains of definition.

4. Action calculi

4.1. The first-order calculus

As for gs-monoidal rewriting logic, also action calculi consider as primitive the combination of rules with respect to three basic operators. With respect to such a logic, though, the emphasis is on higher-order features: The associated notion is that of *controls*, that is, of generalized parametric operators. Controls allow for modeling some primitive operators of process calculi, such as *prefixing*.

We recall here the presentation of action calculi given in [2,32]. It is slightly different from the original presentation in [64] regarding the choice of basic operators and axioms, but it is by now considered standard, and it is better suited for our purposes. In addition, we freely use the notation introduced in Section 2 for monoidal and gs-monoidal theories.

Definition 4.1 (*Controls*). A set of *controls* \mathcal{K} over a monoid \mathbb{Z} of *arities* is a pair (K, R) , where K is the set of control names, and $R \subseteq K \times \mathbb{Q}^* \times \mathbb{Q}$ is the set of associated control *arities*, for $\mathbb{Q} = \mathbb{Z} \times \mathbb{Z}$. For each $(k, \langle p_1, q_1 \rangle, \dots, \langle p_r, q_r \rangle, \langle p, q \rangle) \in R$ we use the notation $k : \langle p_1, q_1 \rangle, \dots, \langle p_r, q_r \rangle \rightarrow \langle p, q \rangle \in \mathcal{K}$.

Intuitively, a set of controls generalizes the standard notion of hyper-signature: Each control name $k \in K$ is an operator, and it is implicitly *polymorphic*, since there exists in general an infinite number of tuples $(\langle p_1, q_1 \rangle, \dots, \langle p_r, q_r \rangle, \langle p, q \rangle)$ such that $k : \langle p_1, q_1 \rangle, \dots, \langle p_r, q_r \rangle \rightarrow \langle p, q \rangle \in \mathcal{K}$. We say that $k \in K$ has *rank* r (in general, that is has *fixed rank*), if $R \cap (\{k\} \times \mathbb{Q}^r \times \mathbb{Q}) \neq \emptyset$ for at most one r . A hyper-signature is then a set of controls with fixed rank 0, such that \mathbb{Z} is the free monoid S^* over a set S of arities, and $R \cap (\{k\} \times \varepsilon \times \mathbb{Q})$ contains at most one element for each k (thus R becomes a function from K to \mathbb{Q}).

Definition 4.2 (*Action calculi statics*). Given a set \mathcal{K} of controls over a monoid \mathbb{Z} of arities, and a set of names $X = \bigsqcup_{z \in \mathbb{Z}} X^z$, the set $T_X(\mathcal{K})$ of terms over (\mathcal{K}, X) is the set of arrows generated by the following inference rules:

$$\begin{array}{ll}
 \text{(pairing)} \quad \frac{s : \underline{p} \rightarrow \underline{q}, \quad t : \underline{p}' \rightarrow \underline{q}'}{s \otimes t : \underline{p} \otimes \underline{p}' \rightarrow \underline{q} \otimes \underline{q}'} & \text{(identities)} \quad \frac{z \in \mathbb{Z}_c}{id_z : z \rightarrow z} \\
 \text{(composition)} \quad \frac{s : \underline{p} \rightarrow \underline{q}, \quad t : \underline{q} \rightarrow \underline{k}}{s; t : \underline{p} \rightarrow \underline{k}} & \text{(abstraction)} \quad \frac{t : p \rightarrow q, \quad x \in X_z}{(x)t : z \otimes p \rightarrow q} \\
 \text{(names)} \quad \frac{x \in X_z}{\langle x \rangle : \varepsilon \rightarrow z} & \text{(controls)} \quad \frac{t_1 : p_1 \rightarrow q_1, \dots, t_r : p_r \rightarrow q_r}{k(t_1, \dots, t_r) : p \rightarrow q}
 \end{array}$$

for $k : \langle p_1, q_1 \rangle, \dots, \langle p_r, q_r \rangle \rightarrow \langle p, q \rangle \in \mathcal{K}$, and by satisfying in addition to the axioms of a monoidal theory (see Definition 2.3) also the concrete axioms

$$(\langle y \rangle \otimes id_p); (x)t = t\{^y/x\} \quad (x)((\langle x \rangle \otimes id_p); s) = s \quad \text{for } x \notin fn(s)$$

for all $x, y \in X_z$, $t : p \rightarrow q$, $s : z \otimes p \rightarrow q$.

The notions of *free* and *bound* names are standard: (x) binds the name x , $\langle x \rangle$ represents a free occurrence of x and $t\{s/x\}$ denotes the application of a capture-avoiding substitution. Thus, intuitively, the first *concrete* axiom simply describes the application of an abstraction to a name; the second deletes an abstraction over a name not occurring inside the term.

We have then all the basic features of a higher-order calculus, and more structure (in fact, a gs-monoidal one) is actually hidden in the interaction between basic operators and abstraction. Until now, however, we have just defined the *statics* of an action calculus, that is, the data structure (in our own terminology, the theory) over which rewriting is executed. The *dynamics* of a calculus — a topic less studied in the literature — is described as a preorder over the arrows of each calculus, where $s \searrow t$ means that s can be rewritten to t .

Definition 4.3 (*Action calculi dynamics*). Given a set \mathcal{K} of controls, and a set of names X , a *reaction relation* for $T_X(\mathcal{K})$ is given by a preorder \searrow over the terms in $T_X(\mathcal{K})$, which preserves the arities and is closed under abstraction, composition and pairing.

Equivalently, we will sometimes say that \searrow is generated by the closure of a basic reaction \searrow_R , that is, $\searrow = (\searrow_R)^{\{(x), \cdot, \otimes\}}$. Thus, the reaction is *not* closed under controls; as we will see, this is the pivotal point for action calculi, enhancing their expressive power and allowing for their modeling of mobility.

Definition 4.4 (*Action calculus*). An action calculus AC is a triple $\langle \mathcal{K}, X, \searrow \rangle$, for \mathcal{K} a set of controls, X a set of names, and \searrow a reaction relation over $T_X(\mathcal{K})$. By abuse of notation, we usually denote a calculus by $AC_X(\mathcal{K})$, that is, characterizing only its statics.

4.2. Review of applications

4.2.1. At the basis of higher-order: on the λ -calculus

As well-argued in [68], the relevant aspect of abstraction in action calculi is its use for the communication of names, instead of (the body of) processes. The feature is typical of the formalism at the roots of action calculi, the π -calculus, and it is in contrast with the usual higher-order languages. In order to faithfully encode (a variant of) the *simply typed* λ -calculus, explicit controls for λ -abstraction and function application are thus needed.

The action calculus LAMC lies at the basis of the presentation of *higher-order* action calculi [62]. The main idea is to define the set of generators as the free closure of the set of (underlined) natural numbers with respect to a ‘functional map’ $-\Rightarrow-$. The monoid of arities \mathbb{N}_f is then obtained by closing the set $\mathbb{N}_c \uplus \{\underline{n} \Rightarrow \underline{m} \mid \underline{n}, \underline{m} \in \mathbb{N}_c\}$ with respect to a monoid operator, which still coincides with the monoid operator of \mathbb{N}_c on natural numbers. Thus, the interaction of abstraction with the controls λ of rank 1 and ap of rank 0 recovers both λ -abstraction and function application.

Definition 4.5 (*Action calculus for λ -calculus*). The action calculus LAMC (over the monoid of arities \mathbb{N}_f) contains a set of names $X^1 \uplus X^{n \Rightarrow m}$ and the set of controls $\lambda : \langle \underline{n}, \underline{m} \rangle \rightarrow \langle \underline{0}, \underline{n} \Rightarrow \underline{m} \rangle$ of rank 1 and $ap : \varepsilon \rightarrow \langle \langle \underline{n} \Rightarrow \underline{m} \rangle \otimes \underline{n}, \underline{m} \rangle$ of rank 0.⁶ The reaction relation is obtained as the free closure of the following rules of deduction:

$$\begin{aligned} (\sigma) \quad & (\lambda(t) \otimes id_{\underline{l}}); (x)s \searrow s\{\lambda(t)/x\} \quad \text{for } t : \underline{n} \rightarrow \underline{m}, s : \underline{l} \rightarrow \underline{k}, x \in X^{n \Rightarrow m}, \\ (\beta) \quad & (\lambda(t) \otimes id_{\underline{n}}); ap \searrow t \quad \text{for } t : \underline{n} \rightarrow \underline{m}. \end{aligned}$$

The combination of the two rules implements β -reduction: The first rule allows the substitution of higher-order variables with terms; the second simulates the η -reduction over a λ -abstraction.

Note that σ is a schema, since it relies on the operation of substitution. As argued in [63], it ‘can be replaced by a set of rules which perform the substitution incrementally’, thus falling into the range of calculi for explicit substitutions, recalled in Section 3.2.2. We refer to [33] for a correspondence result between a variant of the simply-typed, call-by-value λ -calculus arising from the computational λ -calculus of Moggi [66] and the action calculus LAMB , obtained extending LAMC with a reaction rule for simulating the η -reduction over a higher-order variable, namely

$$(\eta) \quad \lambda(\langle x \rangle \otimes id_{\underline{n}}); ap \searrow \langle x \rangle \quad \text{for } x \in X^{n \Rightarrow m}.$$

4.2.2. Prefixes as boxes: on the asynchronous π -calculus

The π -calculus [65] is one of the best studied examples of *mobile* process calculi, namely calculi in which the communication topology among processes can evolve dynamically when computation progresses. We recall here the action calculus PIC , extended with the *box* control, which is able to recast the reduction semantics for a suitable sublanguage of the *asynchronous* π -calculus [38], a variant of the π -calculus where emission of messages is non-blocking, described in the appendix.

Definition 4.6 (*Action calculus for π -calculus*). The action calculus PIC (over the monoid of arities \mathbb{N}_c) contains the set of names $X^1 = \text{Names}$ (that is, those names used for the construction of the basic π -processes in Definition A.1), and the set of controls $box : \langle \underline{n}, \underline{m} \rangle \rightarrow \langle \underline{1}, \underline{m} \rangle$ of rank 1, $v : \varepsilon \rightarrow \langle \underline{0}, \underline{1} \rangle$ and $out : \varepsilon \rightarrow \langle \underline{1} \otimes \underline{n}, \underline{0} \rangle$ of rank 0. The reaction relation is obtained via the free closure of the following rule of deduction (which is a schema)

$$out_x \otimes box_x(t) \searrow_p t \quad \text{for } t : \underline{n} \rightarrow \underline{m}, x \in X^1$$

for derived operators $out_x = (\langle x \rangle \otimes id_{\underline{n}}); out$ and $box_x(t) = \langle x \rangle; box(t)$.

Similarly to what happened for the encoding of the λ -calculus, the interaction of the controls with abstraction is able to recover prefix, output and restriction. Now we address the question of mapping each asynchronous basic process into a term of

⁶ The rank 0 does not mean that ap is just a standard operator for a signature, since it is parametric with respect to $\underline{n}, \underline{m}$.

the statics of PIC . We consider the mapping $\hat{\cdot} : \text{BPProc} \rightarrow T_X(\text{box}, \nu, \text{out})$ defined as follows:

$$\begin{aligned} \hat{0} &= id_{\underline{0}}, & \widehat{P|Q} &= \hat{P} \otimes \hat{Q}, & \widehat{(\nu x)P} &= \nu; (x)\hat{P} \\ \widehat{\bar{x}y} &= (\langle x \rangle \otimes \langle y \rangle); \text{out}, & \widehat{x(y).P} &= \langle x \rangle; \text{box}(\langle y \rangle \hat{P}) \end{aligned}$$

Note that each basic process P is mapped into a term $\hat{P} : \underline{0} \rightarrow \underline{0}$, even if there are terms of arity $\langle \underline{0}, \underline{0} \rangle$ that do not lie in the image of the translation. Nevertheless, the following theorem, appeared in [63], states that the translation represents faithfully the reduction relation for the fragment of the π -calculus.

Theorem 4.1 (Reduction as reaction). *Given P, Q basic processes, the following properties hold:*

- $P \cong Q$ if and only if $\hat{P} = \hat{Q}$;
- if $P \rightarrow_b Q$, then $\hat{P} \searrow_p \hat{Q}$;
- if $\hat{P} \searrow_p t$, then there exists R such that $P \rightarrow_b R$ and $t = \hat{R}$.

4.3. Flattening controls

The main problem in the encoding of an action calculus into a rewrite theory is linked to the elimination of names. More properly, it is linked to the elimination of abstraction, and of the axioms involving the metacondition of *freeness*; without those axioms, names could instead be thought of just as additional constants, as for λ -calculus in Section 3.2.2. When introduced by Gardner in [31], *closed action calculi* were designed with the intention of offering a *name-free* account of action calculi. More general results have been proved recently by Pavlović [68], who showed that abstraction is a derived operator in a large class of closed calculi containing names. We recall now the basic definitions of *closed calculi with names*, restricting ourselves, for the sake of readability, to the one-sorted case (hence, considering \mathbb{N}_c as the monoid of arities), and to controls with fixed rank.

Definition 4.7 (*Closed controls*). Given a set of controls \mathcal{K} , the corresponding set of *closed controls* \mathcal{K}_c is defined as the set of controls $\{k_l : \langle \underline{l} \otimes \underline{n}_1, \underline{m}_1 \rangle, \dots, \langle \underline{l} \otimes \underline{n}_r, \underline{m}_r \rangle \rightarrow \langle \underline{l} \otimes \underline{n}, \underline{m} \rangle\}$ for all $\underline{l} \in \mathbb{N}_c$ and $k : \langle \underline{n}_1, \underline{m}_1 \rangle, \dots, \langle \underline{n}_r, \underline{m}_r \rangle \rightarrow \langle \underline{n}, \underline{m} \rangle \in \mathcal{K}$.

Roughly, an extended control k_l mimics the application of a control k to a set of terms that have been ‘abstracted’ with respect to a sequence of names of length l . Intuitively, an action calculus $AC_X(\mathcal{K})$ can be encoded into a closed calculus with names via a mapping F that associates to a term of the form $(x)[k(t)]$ its ‘simulation’ $k_{\underline{l}}(F(t))$.

Definition 4.8 (*Closed action calculi with names*). Given a set \mathcal{K} of controls, and a set of names X , the set $CT_X(\mathcal{K})$ of *extended terms* over (\mathcal{K}, X) is the set of arrows

generated by the following inference rules:

$$\begin{array}{ll}
\text{(pairing)} \frac{s : \underline{n} \rightarrow \underline{m}, t : \underline{n}' \rightarrow \underline{m}'}{s \otimes t : \underline{n} \otimes \underline{n}' \rightarrow \underline{m} \otimes \underline{m}'} & \text{(identities)} \frac{\underline{n} \in \mathbb{N}_c}{id_{\underline{n}} : \underline{n} \rightarrow \underline{n}} \\
\text{(composition)} \frac{s : \underline{n} \rightarrow \underline{m}, t : \underline{m} \rightarrow \underline{k}}{s; t : \underline{n} \rightarrow \underline{k}} & \text{(duplicators)} \frac{\underline{n} \in \mathbb{N}_c}{\nabla_{\underline{n}} : \underline{n} \rightarrow \underline{n} \otimes \underline{n}} \\
\text{(dischargers)} \frac{\underline{n} \in \mathbb{N}_c}{!_{\underline{n}} : \underline{n} \rightarrow \underline{0}} & \text{(permutations)} \frac{\underline{n}, \underline{m} \in \mathbb{N}_c}{\rho_{\underline{n}, \underline{m}} : \underline{n} \otimes \underline{m} \rightarrow \underline{m} \otimes \underline{n}} \\
\text{(names)} \frac{x \in X}{\langle x \rangle : \underline{0} \rightarrow \underline{1}} & \text{(controls)} \frac{t_1 : \underline{n}_1 \rightarrow \underline{m}_1, \dots, t_r : \underline{n}_r \rightarrow \underline{m}_r}{k(t_1, \dots, t_r) : \underline{n} \rightarrow \underline{m}}
\end{array}$$

for $k : \langle \underline{n}_1, \underline{m}_1 \rangle, \dots, \langle \underline{n}_r, \underline{m}_r \rangle \rightarrow \langle \underline{n}, \underline{m} \rangle \in \mathcal{K}_c$, and by satisfying in addition to the axioms of a gs-monoidal theory (see Definition 2.4) also the *control* axioms

$$\begin{aligned}
(\rho_{\underline{l}, \underline{l}'} \otimes id_{\underline{n}}); k_{\underline{l}' \otimes \underline{l}}(t_1, \dots, t_r) &= k_{\underline{l} \otimes \underline{l}'}((\rho_{\underline{l}, \underline{l}'} \otimes id_{\underline{n}_1}); t_1, \dots, (\rho_{\underline{l}, \underline{l}'} \otimes id_{\underline{n}_r}); t_r) \\
(\nabla_{\underline{l}} \otimes id_{\underline{n}}); k_{\underline{l} \otimes \underline{l}}(t_1, \dots, t_r) &= k_{\underline{l}}((\nabla_{\underline{l}} \otimes id_{\underline{n}_1}); t_1, \dots, (\nabla_{\underline{l}} \otimes id_{\underline{n}_r}); t_r) \\
(!_{\underline{l}} \otimes id_{\underline{n}}); k_{\underline{0}}(t_1, \dots, t_r) &= k_{\underline{l}}(!_{\underline{l}} \otimes id_{\underline{n}_1}); t_1, \dots, (!_{\underline{l}} \otimes id_{\underline{n}_r}); t_r)
\end{aligned}$$

for all $k \in \mathcal{K}$, $\underline{l}, \underline{l}' \in \mathbb{N}_c$, and the *naming* axioms

$$\begin{aligned}
\langle x \rangle; \nabla_{\underline{1}} &= \langle x \rangle \otimes \langle x \rangle, \quad \langle x \rangle; !_{\underline{1}} = id_{\underline{0}} \\
\langle \langle x \rangle \otimes id_{\underline{l} \otimes \underline{n}} \rangle; k_{\underline{l} \otimes \underline{l}}(t_1, \dots, t_r) &= k_{\underline{l}}(\langle \langle x \rangle \otimes id_{\underline{l} \otimes \underline{n}_1} \rangle; t_1, \dots, \langle \langle x \rangle \otimes id_{\underline{l} \otimes \underline{n}_r} \rangle; t_r)
\end{aligned}$$

for all $k \in \mathcal{K}$, $x \in X$, $\underline{l} \in \mathbb{N}_c$.

Given a set \mathcal{K} of controls, and a set of names X , a *reaction relation* for $CT_X(\mathcal{K})$ is given by a preorder \searrow over the terms in $CT_X(\mathcal{K})$, which preserves the arities and is closed under composition and pairing.

An *extended* closed action calculus CAC is a triple $\langle \mathcal{K}, X, \searrow \rangle$, for \mathcal{K} a set of controls, X a set of names, and \searrow a reaction relation over $CT_X(\mathcal{K})$. By abuse of notation, we usually denote a closed calculus by $CAC_X(\mathcal{K})$.

Intuitively, the control and naming axioms simulate the interaction between the basic operators of the gs-monoidal theory on one side, and the controls and names, respectively, on the other. For example, the final naming axiom states that the term obtained by composing the name $\langle x \rangle$ and the control $k_{\underline{l} \otimes \underline{l}}$, applied to a set of subterms abstracted with respect to $1 + \underline{l}$ names, is equivalent to the control $k_{\underline{l}}$, applied to a set of subterms abstracted with respect to \underline{l} names.

We refer the reader to [32] for an intuitive, syntax-driven presentation of the encoding of the generic action calculus $AC_X(\mathcal{K})$ into the closed calculus $CAC_X(\mathcal{K})$. In this

section we consider such an encoding as given, and we prove instead the correspondence between closed calculi with names and gs-monoidal theories. Our presentation suggests that extended terms are just arrows of suitable (gs-)monoidal theories, where the *generator* rule is subsumed by the more general *control* rule. Thus, a set of controls can indeed be considered as a hyper-signature, just by encoding each term of the form $k(t_1, \dots, t_n)$ as a new operator.

Definition 4.9 (*Control signatures*). Given a set \mathcal{K} of controls, the associated *control signature* $\Sigma^{\mathcal{K}}$ is a hyper-signature such that $\sigma_{k(t_1, \dots, t_n)} \in \Sigma_{\underline{n}, \underline{m}}^{\mathcal{K}}$ if and only if $k(t_1, \dots, t_n) : \underline{n} \rightarrow \underline{m} \in CT_{\emptyset}(\mathcal{K})$.

We agree that the previous translation is rather simplistic. More abstract presentations could be obtained by enriching the kind of equational logic at hand, using e.g. membership equational logic [56], but we consider the proposed definition of control signatures adequate for our purposes.

Definition 4.10 (*Name signatures*). Given a set of names X , the associated *name signature* Σ^X is a signature such that $\sigma_x \in \Sigma_{0,1}^X$ if and only if $x \in X$.

Definition 4.11 (*Calculus signatures*). Given a set \mathcal{K} of controls, and a set of names X , the associated *calculus signature* $\Sigma^{\mathcal{K}, X}$ is the disjoint union $\Sigma^{\mathcal{K}} \uplus \Sigma^X$; the *calculus axioms* $E^{\mathcal{K}, X}$ are those equations over terms of $\mathbf{GS}(\Sigma^{\mathcal{K}, X})$ that correspond to the control and naming axioms of Definition 4.8.

The following proposition shows that it is not necessary to generalize the control signature, including a new operator for each term containing both controls *and* names. Instead, names can be freely added *after* the construction of the signature $\Sigma^{\mathcal{K}}$.

Proposition 4.2 (Normal forms). *Given a set \mathcal{K} of controls, and a set of names X , for each arrow $t : \underline{n} \rightarrow \underline{m} \in CT_X(\mathcal{K})$, containing the names $x_1, \dots, x_l \in X$, there exists an arrow $t' : l \otimes \underline{n} \rightarrow \underline{m} \in CT_{\emptyset}(\mathcal{K})$ such that $t = (\langle x_1 \rangle \otimes \dots \otimes \langle x_l \rangle \otimes id_{\underline{n}}); t'$.*

The proof can be easily given by induction on the structure of terms, but it can be recovered also by general results on the characterization of *functional completeness* in [68]. Thus, the following proposition immediately holds.

Proposition 4.3 (Encoding closed statics). *Given a set \mathcal{K} of controls, and a set of names X , the following properties hold:*

(1) *the morphism $G' : \mathbf{GS}_{E^{\mathcal{K}, X}}(\Sigma^{\mathcal{K}, X}) \rightarrow CT_X(\mathcal{K})$, defined inductively as*

$$G'(\sigma_x) = \langle x \rangle \quad G'(\sigma_{k(t_1, \dots, t_n)}) = k(t_1, \dots, t_n)$$

can be extended to a gs-monoidal theory morphism;

(2) the morphism $G^o : CT_X(\mathcal{K}) \rightarrow \mathbf{GS}_{E^{\mathcal{X},X}}(\Sigma^{\mathcal{X},X})$, defined inductively as

$$G^o(\langle x \rangle) = \sigma_x \quad G^o(k(t_1, \dots, t_n)) = \sigma_{k(t_1, \dots, t_n)} \text{ for } t_i \in CT_0(\mathcal{K})$$

can be extended to a gs-monoidal theory morphism;

(3) the morphisms G' and G^o are inverse (up to equivalence) to each other.

We can now spell out the precise correspondence between closed action calculi with names and gs-monoidal theories.

Definition 4.12 (*Control theories*). Given a closed calculus with names $CAC_X(\mathcal{K})$, whose reaction relation \searrow is generated from the reaction relation \searrow_R , the associated gs-monoidal rewrite theory $\mathcal{R}_{\mathcal{K},X}$ is the 4-tuple $\langle \Sigma^{\mathcal{X},X}, E^{\mathcal{X},X}, L, R \rangle$, where $r_{s,t} : [G^o(s)] \rightarrow [G^o(t)] \in R$ for all $s \searrow_R t$.

Since each extended term obtained by an application of the *controls* rule in Definition 4.8 is simulated by a new operator, the gs-monoidal theory associated to a calculus generates sequents that may only perform reductions ‘at the top’: It is impossible to rewrite ‘inside’ any such operators. Thus, the following result can be easily proved by structural induction.

Proposition 4.4 (*Encoding dynamics*). Let $CAC_X(\mathcal{K})$ be a closed calculus with names, and let $\mathcal{R}_{\mathcal{K},X}$ be the associated gs-monoidal rewrite theory. Furthermore, let $s, t : \underline{n} \rightarrow \underline{m}$ be extended terms, and let $[G^o(s)], [G^o(t)] : \underline{n} \rightarrow \underline{m}$ be the corresponding elements of $\mathbf{GS}_{E^{\mathcal{X},X}}(\Sigma^{\mathcal{X},X})$.

- If $s \searrow t$, then the gs-monoidal sequent $[G^o(s)] \Rightarrow [G^o(t)] : \underline{n} \rightarrow \underline{m}$ is entailed by $\mathcal{R}_{\mathcal{K},X}$;
- if the gs-monoidal sequent $[G^o(s)] \Rightarrow u : \underline{n} \rightarrow \underline{m}$ is entailed by $\mathcal{R}_{\mathcal{K},X}$, then there exists an extended term t such that $s \searrow t$ and $u = [G^o(t)]$.

5. Tile logic

5.1. On algebraic tile logic

Differently from rewriting logic and action calculi, each rule in tile logic aims at describing the possible behaviour of an open system, that is, some kind of process *module*, whose evolution is dynamically dependent on the synchronization of its sub-components. *Algebraic tile systems* represent an easy formalism for describing the behaviour of reactive systems; they were introduced in [30], where also the corresponding models were studied.

Definition 5.1 (*Tile systems*). An algebraic tile system \mathcal{T} is a 6-tuple $\langle \Sigma_h, E_h, \Sigma_v, E_v, L, R \rangle$, where Σ_h, Σ_v are signatures, E_h, E_v are sets of equations (over $\mathbf{A}(\Sigma_h)$ and $G(\Sigma_v)$, respectively), L is a set of labels and $R \subseteq L \times \mathbf{A}_{E_h}(\Sigma_h) \times G_{E_v}(\Sigma_v) \times G_{E_v}(\Sigma_v) \times \mathbf{A}_{E_h}(\Sigma_h)$

Table 3
Inference rules for algebraic tile sequents

Basic sequents. *Generators and identities:*

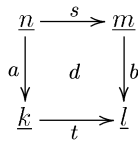
$$\begin{array}{c}
 r : s \xrightarrow[a]{b} t \in R \\
 \text{(gen)} \frac{}{s \xrightarrow[a]{b} t} \\
 \text{(v-ref)} \frac{a : \underline{n} \rightarrow \underline{k} \in G_{E_v}(\Sigma_v)}{id_{\underline{n}} \xrightarrow[a]{a} id_{\underline{k}}} \quad \text{(h-ref)} \frac{s : \underline{n} \rightarrow \underline{m} \in A_{E_h}(\Sigma_h)}{s \xrightarrow[id_{\underline{n}}]{id_{\underline{m}}} s}
 \end{array}$$

Composed sequents. *Parallel, horizontal and vertical compositions:*

$$\begin{array}{c}
 \text{(vert)} \frac{s_1 \xrightarrow[b_1]{a_1} t, \quad t \xrightarrow[b_2]{a_2} t_1}{s_1 \xrightarrow[b_1;b_2]{a_1;a_2} t_1} \\
 \text{(par)} \frac{s_1 \xrightarrow[b_1]{a_1} t_1, \quad s_2 \xrightarrow[b_2]{a_2} t_2}{s_1 \otimes s_2 \xrightarrow[b_1 \otimes b_2]{a_1 \otimes a_2} t_1 \otimes t_2} \quad \text{(hor)} \frac{s_1 \xrightarrow[b]{a_1} t_1, \quad s_2 \xrightarrow[b_1]{a_2} t_2}{s_1; s_2 \xrightarrow[b_1]{a_1} t_1; t_2}
 \end{array}$$

is a set of labeled rules, such that for all $(d, s, a, b, t) \in R$, we have $s : \underline{n} \rightarrow \underline{m}, t : \underline{k} \rightarrow \underline{l}$ if and only if $a : \underline{n} \rightarrow \underline{k}, b : \underline{m} \rightarrow \underline{l}$.

For $(d, s, a, b, t) \in R$ we use the notation $d : s \xrightarrow[a]{b} t$, or we depict it as a *tile* making explicit the source and target of each operator.



As for rewrite theories, the rules of a tile system are its basic sequents. In the following, we say that the *initial configuration* s evolves to the *final configuration* t , using a *trigger* a and producing an *effect* b , if the sequent $s \xrightarrow[a]{b} t$ can be obtained by finitely many applications of certain inference rules.

Definition 5.2 (*Tile sequents*). Let $\mathcal{T} = \langle \Sigma_h, E_h, \Sigma_v, E_v, L, R \rangle$ be an algebraic tile system. We say that \mathcal{T} entails the algebraic tile sequent $s \xrightarrow[a]{b} t$, written $\mathcal{T} \vdash s \xrightarrow[a]{b} t$, if and only if it can be obtained by a finite number of applications of the inference rules given in Table 3.

Basic rules provide the generators of the sequents, together with suitable identity arrows, whose intuitive meaning is e.g. that an element of $A_{E_h}(\Sigma_h)$ stays idle during a rewrite, showing no effect and using no trigger. Composition rules express the way

in which sequents can be combined, either sequentially (*vert*), or executing them in parallel (*par*), or nesting one inside the other (*hor*).

Similarly to rewrite theories, also for tile systems a correspondence between deductions and (concurrent) computations can be established, decorating each sequent with a proof term that encodes the causes of the computation. Different, yet computationally equivalent deductions may be equated via suitable sets of axioms for proof terms, extending those for rewriting logic. For a comparison between the axioms for rewriting logic and those for algebraic tile logic, we refer the reader to [30].

Since we are not interested here in the computational structure of proof terms, it is immediate to recover different tile systems, simply by different choices of the theories where the components of a tile live. In particular, we will consider *gs-monoidal tile systems* for the case study in Section 5.2.2. As algebraic tile systems generalize rewrite theories, thus *gs-monoidal* or *monoidal* tile systems generalize *gs-monoidal* or *monoidal* rewrite theories, as defined in Section 3.3.

5.2. Review of applications

5.2.1. A finite schema for non-determinism and parallelism: on ccs

Process (Description) Algebras [5,35,61] offer a constructive way to describe *concurrent systems*, considered as structured entities (the *processes*) interacting by some synchronization mechanism. The central idea is that a process is a term of an algebra over a set of process constructors, on the assumption that algebraic operators represent basic features of a concurrent system. Maybe, the most representative example is the *Calculus of Communicating Systems* (shortly, *ccs*) introduced by Milner [60]. The structure of *ccs* processes is given by the signature Σ_{ccs} described below.

Definition 5.3 (*One-sorted signature for ccs*). The one-sorted signature Σ_{ccs} associated to *ccs* contains the constant $\text{nil} : \underline{0} \rightarrow \underline{1}$, denoting the *inactive* state; the *prefix* operators $\{\alpha, \bar{\alpha} : \underline{1} \rightarrow \underline{1} \mid \alpha \in \text{Act} \setminus \{\tau\}\}$; the *silent action* operator $\tau : \underline{1} \rightarrow \underline{1}$; the *restriction* operators $\{\backslash_{\alpha} : \underline{1} \rightarrow \underline{1} \mid \alpha \in \text{Act} \setminus \{\tau\}\}$; and finally, the *parallel composition* $\parallel : \underline{2} \rightarrow \underline{1}$ and the *non-deterministic choice* $+$: $\underline{2} \rightarrow \underline{1}$ operators.

We denote by Σ_p the restriction of Σ_{ccs} to the *prefix* the *silent action* operators, ranged over by $\mu : \underline{1} \rightarrow \underline{1}$.

The operational semantics of *ccs* is given by a transition system T_{ccs} , presented in the *sos* style via a set of rules with *side-conditions*; information on the *actions* performed by the transitions in the premise is needed, before applying a rule. Moreover, they assure that the rewriting steps are never performed inside a prefix, since the order in which the rewrites are actually executed is important: The correct operational behaviour of the agent $P = \alpha.\beta.\text{nil}$ is expressed by saying that it executes first α and then β . Both properties are easily recast via the features of the inference rules for tiles.

Definition 5.4 (Tile system for CCS). The algebraic tile system \mathcal{T}_{CCS} for CCS is the 6-tuple $\langle \Sigma_{\text{CCS}}, \emptyset, \Sigma_p, \emptyset, L_{\text{CCS}}, R_{\text{CCS}} \rangle$, where R_{CCS} is the set of labeled rules given below

$$\begin{array}{ll}
act_\mu : \mu \xrightarrow[\mu]{id} id & res_\alpha : \backslash_\alpha \xrightarrow[\mu]{\mu} \backslash_\alpha \quad \text{for } \mu \notin \{\alpha, \bar{\alpha}\} \\
+_\mu^l : + \xrightarrow[\mu]{\mu \otimes id} id \otimes !_\perp & +_\mu^r : + \xrightarrow[\mu]{id \otimes \mu} !_\perp \otimes id \\
\bar{\zeta}_\mu^l : \parallel \xrightarrow[\mu]{\mu \otimes id} \parallel & \bar{\zeta}_\mu^r : \parallel \xrightarrow[\mu]{id \otimes \mu} \parallel & \bar{\zeta}_\alpha : \parallel \xrightarrow[\tau]{\alpha \otimes \bar{\alpha}} \parallel
\end{array}$$

and where id is a shorthand for id_\perp .

Note that there is exactly one tile for each operational rule of CCS; both act_μ and res_α are parametric with respect to the prefix operators, since the corresponding rules are so. The effect μ indicates that the process is actually ‘running’, outputting the prefix μ . For example, the rule act_μ prefixes an idle process with μ , and then starts the execution, consuming that same action. There are also three rules dealing with the parallel operator: $\bar{\zeta}_\mu^s$ synchronizes two running processes, while $\bar{\zeta}_\mu^l$ and $\bar{\zeta}_\mu^r$ perform an asynchronous move, taking a running and an idle process.

As a simple example (needed later in Section 5.3), consider the process $P_e = ((\alpha.nil + \beta.nil) \parallel \bar{\alpha}.nil) \backslash_\alpha$ and the transition $P_e \xrightarrow{\tau} (nil \parallel nil) \backslash_\alpha$. The entailment of the latter is simulated via the following deduction steps, where the blank trigger and id are shorthands for id_\perp and id_\perp , respectively.

$$\begin{array}{c}
\frac{nil \rightarrow nil \quad \alpha \xrightarrow[\alpha]{id} id}{id} \\
\frac{nil; \alpha \rightarrow nil \quad nil; \beta \rightarrow nil; \beta}{id} \\
\frac{(nil; \alpha) \otimes (nil; \beta) \xrightarrow[\alpha \otimes id]{id} nil \otimes (nil; \beta) \quad + \xrightarrow[\alpha]{\alpha \otimes id} +}{((nil; \alpha) \otimes (nil; \beta)); + \rightarrow nil} \\
\vdots \\
\frac{((nil; \alpha) \otimes (nil; \beta)); + \rightarrow nil \quad nil; \bar{\alpha} \rightarrow nil \quad \parallel \xrightarrow[\tau]{\alpha \otimes \bar{\alpha}} \parallel}{(((((nil; \alpha) \otimes (nil; \beta)); +) \otimes (nil; \bar{\alpha})); \parallel) \xrightarrow[\tau]{\tau} (nil \otimes nil); \parallel} \quad \backslash_\alpha \xrightarrow[\tau]{\tau} \backslash_\alpha \\
\frac{(((((nil; \alpha) \otimes (nil; \beta)); +) \otimes (nil; \bar{\alpha})); \parallel); \backslash_\alpha \rightarrow (nil \otimes nil); \parallel; \backslash_\alpha}{\tau}
\end{array}$$

Proposition 2.1 implies that there is a bijection between the set of CCS processes and the arrows of the algebraic theory $\mathbf{A}(\Sigma_{\text{CCS}})$ with source $\underline{0}$ and target $\underline{1}$; we denote by $[P]$ the equivalence class associated to the process P .

Proposition 5.1 (Interleaving correspondence). *Given CCS processes P, Q , the transition $P \xrightarrow[\mu]{\mu} Q$ is entailed by the CCS transition system T_{CCS} if and only if the sequent $[P] \rightarrow [Q]$ is entailed by the tile system \mathcal{T}_{CCS} .*

There are however many more sequents entailed by \mathcal{T}_{ccs} than transitions in T_{ccs} , since the transition system is defined only over *closed processes*, whose image are arrows with source $\underline{0}$ and target $\underline{1}$. In fact, as we already remarked, tile systems are naturally equipped with a rewrite relation over open configurations. In our case, *open processes* are roughly processes containing place-holders for unspecified subcomponents, and they are represented by arrows whose source is different from $\underline{0}$.

A relevant part of the studies on process algebras is actually devoted to the notion of *behavioural semantics*, that is, to the study of those equivalences that equate closed processes exhibiting the same observable behaviour (i.e., that are able to perform the same actions). We refer the interested reader to [30] for an introduction and some results on *tile bisimulation*, including its correspondence with the usual notion of *strong bisimulation* for process algebras. We refer to [74] for an analysis on the relevance of open processes for bisimilarity, in a formalism related to tile systems.

5.2.2. Channels as wires: on the asynchronous π -calculus

Recent extensions of the process algebra paradigm involve calculi with higher-order features such as process mobility. Early (and still typical) examples are the π -calculus [65] and the *calculus of higher-order communicating systems* (shortly CHOCS, introduced by Thomsen [76]). The tile system for the asynchronous π -calculus presented in [28] was based on the idea of viewing the free names of a process as part of the specification of the communication interface. In this section we intend to make this intuition more precise, presenting a very simple type system for basic π -processes. Each sentence is of the form $\Gamma \triangleright P$, for P a basic π -process and Γ an *ambient*, that is, an ordered list of names, with no repetition, containing all those occurring free in P .

Definition 5.5 (*Type system for basic processes*). The set *TBPP* of *typed basic processes* for the asynchronous π -calculus is the set of sentences $\Gamma \triangleright P$, for P a basic process and Γ an ambient, inductively generated by the following set of axioms and inference rules:

$$\begin{array}{ccc} \overline{\emptyset \triangleright 0} & \overline{x, y \triangleright \bar{x}y} & \overline{x \triangleright \bar{x}x} \\ \frac{y, x, \Gamma \triangleright P}{x, \Gamma \triangleright x(y).P} & \frac{x, \Gamma \triangleright P}{\Gamma \triangleright (vx)P} & \frac{\Gamma \triangleright P_1, \Gamma \triangleright P_2}{\Gamma \triangleright P_1|P_2} \\ \frac{\Gamma \triangleright P}{\Gamma, x \triangleright P} & \frac{\Gamma_1, x_1, x_2, \Gamma_2 \triangleright P}{\Gamma_1, x_2, x_1, \Gamma_2 \triangleright P} & \end{array}$$

The next result states that our type system is well-behaved with respect to the structural congruence on processes presented in Definition A.1.

Proposition 5.2. *Given an ambient Γ , and structurally congruent basic processes P, Q (i.e., such that $P \cong Q$), $\Gamma \triangleright P$ if and only if $\Gamma \triangleright Q$.*

Since the type system is compatible with the structural congruence over basic processes, we can extend it including ambients, in order to identify those processes which are structurally equivalent up to renaming of their free names.

Definition 5.6 (*Typed structural congruence*). Given typed basic processes $\Gamma_1 \triangleright P$, $\Gamma_2 \triangleright Q$, for ambients $\Gamma_1 = x_1, \dots, x_n$ and $\Gamma_2 = y_1, \dots, y_n$ of the same length, we say that they are structurally congruent, and we write $\Gamma_1 \triangleright P \approx \Gamma_2 \triangleright Q$, if $P\{y_1/x_1, \dots, y_n/x_n\} \cong Q$.

Our tile system \mathcal{T}_π for the asynchronous π -calculus needs a horizontal signature where typed basic processes are interpreted. The hyper-signature Σ_π has sorts e and a . The first symbol is reminiscent of the word *event*, since the terms of sort e can be considered as processes generated by a transition. The second sort, a , is reminiscent of *action*, and the names of this sort will correspond to the names of the calculus. The operators of Σ_π are $\{in : e \otimes a \rightarrow e \otimes a, out : \varepsilon \rightarrow e \otimes a \otimes a, v : a \rightarrow \varepsilon\}$. Obviously, the operator *in* corresponds to the input prefixing, and *out* to the output; note that the only argument of *v*, simulating restriction, is a name and not a process.

The first step for the simulation of the π -calculus is to show an encoding of typed basic processes into arrows of a suitable theory, built out of Σ_π , and proving that such encoding preserves the typed structural congruence. In order to simplify the presentation, showing at the same time the expressive power of our framework, we interpret some of the operators of the calculus in terms of the auxiliary structure of the theory itself. To this aim, we consider the reverse of a gs-monoidal theory, a *cogs-monoidal theory*, on Σ_π . Namely, the set of arrows of $\mathbf{CoGS}(\Sigma_\pi)$ is obtained via the same inference rules for $\mathbf{GS}(\Sigma_\pi)$, but the operators $!_e$ and ∇_e are replaced by the operators $new_e : \varepsilon \rightarrow e$ and $\Delta_e : e \otimes e \rightarrow e$ (and similarly for sort a), equipped with a set of axioms which is the dual of those holding for dischargers and duplicators.

Definition 5.7 (*Typed encoding*). The *encoding* for typed basic processes is the mapping $\llbracket - \rrbracket : TBPP \rightarrow \mathbf{CoGS}(\Sigma_\pi)$ defined by structural recursion on the proof of typing.

- $\llbracket \emptyset \triangleright 0 \rrbracket = new_e$
- $\llbracket x, y \triangleright \bar{x}y \rrbracket = out$
- $\llbracket x \triangleright \bar{x}x \rrbracket = out; (id_e \otimes \Delta_a)$
- $\llbracket x, \Gamma \triangleright x(y).P \rrbracket = \llbracket y, x, \Gamma \triangleright P \rrbracket; (in \otimes id_{[x, \Gamma]}); (id_e \otimes \Delta_a \otimes id_{[\Gamma]})$
- $\llbracket \Gamma \triangleright (vx).P \rrbracket = \llbracket x, \Gamma \triangleright P \rrbracket; (id_e \otimes v \otimes id_{[\Gamma]})$
- $\llbracket \Gamma \triangleright P_1 \mid P_2 \rrbracket = (\llbracket \Gamma \triangleright P_1 \rrbracket \otimes \llbracket \Gamma \triangleright P_2 \rrbracket); \Delta_{e \otimes [\Gamma]}$
- $\llbracket \Gamma, x \triangleright P \rrbracket = \llbracket \Gamma \triangleright P \rrbracket \otimes new_a$
- $\llbracket \Gamma_1, x_1, x_2, \Gamma_2 \triangleright P \rrbracket = \llbracket \Gamma_1, x_2, x_1, \Gamma_2 \triangleright P \rrbracket; (id_{e \otimes [\Gamma_1]} \otimes \rho_{a,a} \otimes id_{[\Gamma_2]})$

where we inductively define $\llbracket \emptyset \rrbracket = \varepsilon$ and $\llbracket x, \Gamma \rrbracket = a \otimes [\Gamma]$.

The mapping is well-defined, in the sense that the final result is independent of the choice of the proof; a typed basic process $\Gamma \triangleright P$ is mapped into a term with source ε and target $e \otimes [\Gamma]$.

Example 5.1 (*How to map a typed basic process*). In order to give some intuition about the intended meaning of the previous inference rules, we show here the

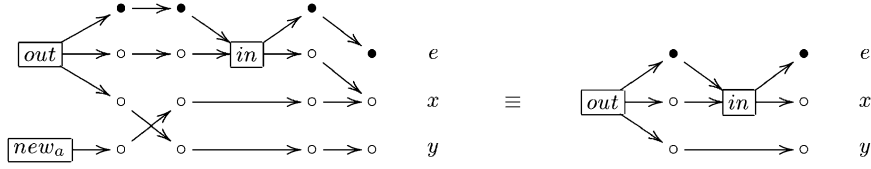


Fig. 3. Graphical equivalence due to gs-monoidal axioms

explicit construction of the mapping for the typed basic process $x, y, w \triangleright x(z).\bar{z}y \mid \bar{x}w$. We first consider the derivation tree associated to the subprocess $x, y, w \triangleright x(z).\bar{z}y$.

$$\begin{array}{c}
 \frac{}{[z, y \triangleright \bar{z}y] = out} \\
 \frac{}{[z, y, x \triangleright \bar{z}y] = out \otimes new_a} \\
 \frac{}{[z, x, y \triangleright \bar{z}y] = (out \otimes new_a); (id_{e \otimes a} \otimes \rho_{a,a})} \\
 \frac{}{[x, y \triangleright x(z).\bar{z}y] = (out \otimes new_a); (id_{e \otimes a} \otimes \rho_{a,a}); (in \otimes id_{a \otimes a}); (id_e \otimes \Delta_a \otimes id_a)} \quad 1 \\
 \frac{}{[x, y \triangleright x(z).\bar{z}y] = out; (in \otimes new_a \otimes id_a); (id_e \otimes \Delta_a \otimes id_a)} \quad 2 \\
 \frac{}{[x, y \triangleright x(z).\bar{z}y] = out; (in \otimes id_a)} \quad 3 \\
 \frac{}{[x, y, w \triangleright x(z).\bar{z}y] = (out; (in \otimes id_a)) \otimes new_a} \\
 \frac{}{[x, y, w \triangleright x(z).\bar{z}y] = (out \otimes new_a); (in \otimes id_{a \otimes a})}
 \end{array}$$

Note the use of Δ_a at step 1 for imposing the matching of names, identifying the subject of the input with one of the names of the ambient. Since that name does not occur in the output, the matching is later eliminated. Steps 2 and 3 are normalization steps, applying the functoriality axiom and various monoidality and coherence axioms.⁷ Fig. 3 depicts the wire-and-box diagram for the term given by the mapping before step 2 and after step 3, where the symbol \bullet represents elements of sort e , and the symbol \circ represents names of sort a .

The following derivation tree maps the subprocess $x, y, w \triangleright \bar{x}w$.

$$\begin{array}{c}
 \frac{}{[x, w \triangleright \bar{x}w] = out} \\
 \frac{}{[x, w, y \triangleright \bar{x}w] = out \otimes new_a} \\
 \frac{}{[x, y, w \triangleright \bar{x}w] = (out \otimes new_a); (id_e \otimes id_a \otimes \rho_{a,a})}
 \end{array}$$

Thus, the typed basic process $x, y, w \triangleright x(z).\bar{z}y \mid \bar{x}w$ is mapped into the term

$$(((out \otimes new_a); (in \otimes id_{a \otimes a})) \otimes ((out \otimes new_a); (id_e \otimes id_a \otimes \rho_{a,a}))); \Delta_{e \otimes a \otimes a \otimes a}$$

which is equivalent to the term $(out \otimes out); (in \otimes \rho_{a, e \otimes a}); (\Delta_{e \otimes a} \otimes id_{a \otimes a})$. The latter can be depicted by both wire-and-box diagrams in Fig. 4: The first diagram is term-like,

⁷ Among others, the first step applies the derived axiom $\rho_{a, \varepsilon} = id_\varepsilon$, while the second one applies the coherence axiom $(id_a \otimes new_a); \Delta_a = id_a$.

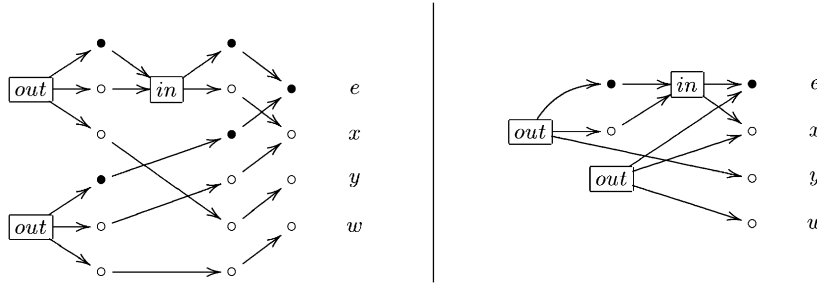


Fig. 4. Two diagrams for the term $[x, y, w \triangleright x(z).\bar{z}y \mid \bar{x}w]$.

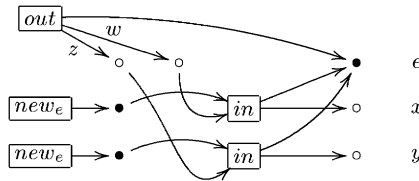


Fig. 5. A term with a forbidden name-sharing situation.

in the sense that it mirrors faithfully the structure of the term; the second is more compact, since all the wires connecting only ‘dots’ have been collapsed. The compact style will be adopted in the rest of the section.

The mapping $[-]$ is clearly sound with respect to structural congruence. For instance, the typed process $a, b, c \triangleright \bar{a}c \mid a(d).\bar{d}b$ is structurally congruent to $x, y, w \triangleright x(z).\bar{z}y \mid \bar{x}w$, since the underlying basic processes are structurally congruent up to α -conversion of the ambients, and it is easy to check that those two typed processes are mapped to the same term. However, the mapping is not surjective, because there are terms with target $e \otimes [I]$, e.g. the one in Fig. 5, that are not in the image of any typed process; they represent some kind of name-sharing situation which is not allowed in the standard construction for basic processes. Furthermore, the mapping is not injective, either. In fact, it induces a coarser equivalence than structural congruence, and such an equivalence is equationally characterized by adding the axioms $(vw)x(y).P \cong x(y).(vw)P$ for all P and $w \notin \{x, y\}$ to the equational presentation of process given in the appendix. Nevertheless, the lack of completeness of our encoding rises no problem, since the additional axioms does not add any further branching: This is formalised by Theorem 5.3, establishing a correspondence between tile entailment and the operational semantics of the calculus.

The next step in the characterization of our tile system is the vertical hyper-signature, Σ_{act} . It contains only one operator, namely $\tau : e \rightarrow e$, obviously denoting the occurrence of a reduction.

The final step is the definition of the rules. The leading intuition is to consider a *sequential* process P (that is, either an output or a process whose top operator is an input prefix) as some kind of software component, possibly distributed over a network, while the process combinators (i.e., parallel composition and restriction) represent the coordination language. Our tile system thus needs only two rules: The first simulates the occurrence of a reduction (i.e., the actual activity of a component); the second simulates the flow of information over the network (i.e., the coordination of the components).

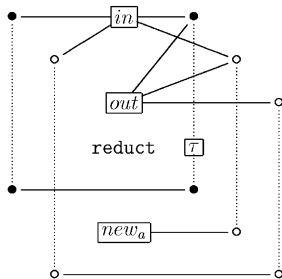
Definition 5.8 (*Tile system for the asynchronous π -calculus*). The tile system \mathcal{T}_π for the asynchronous π -calculus is the 6-tuple $\langle \Sigma_\pi, \emptyset, \Sigma_{act}, \emptyset, L_\pi, R_\pi \rangle$, where $R_\pi \subseteq L_\pi \times \mathbf{CoGS}(\Sigma_\pi) \times G(\Sigma_{act}) \times G(\Sigma_{act}) \times \mathbf{CoGS}(\Sigma_\pi)$, for $L_\pi = \{\mathbf{reduct}, \mathbf{flow}\}$, is the set of labeled rules below.

We first present the rules in sequent form.

$$\mathbf{reduct}: (in \otimes out); (\Delta_{e \otimes a} \otimes id_a) \xrightarrow{\tau \otimes id_a \otimes id_a} id_e \otimes new_a \otimes id_a$$

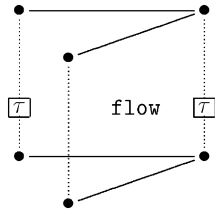
$$\mathbf{flow}: \Delta_e \xrightarrow[\tau]{\tau \otimes id_e} \Delta_e$$

A tile sequent can be represented as a wire-and-box diagram, simply obtained by gluing together the diagrams for the four components of the tile, via the shared interfaces. Note also that all the relevant information is completely represented by the wiring, and only the ordering on the names of source and target should be provided, since it is essential in computing either the horizontal or the vertical composition. Actually, also such an ordering is immaterial for rules, since it will always be possible to obtain tiles with different orderings of names in the interfaces by composing with suitable vertical identities. Thus, the **reduct** and **flow** rules are represented graphically by the wire-and-box diagrams displayed below; to make these diagrams more understandable, we used dotted lines for the vertical wires.



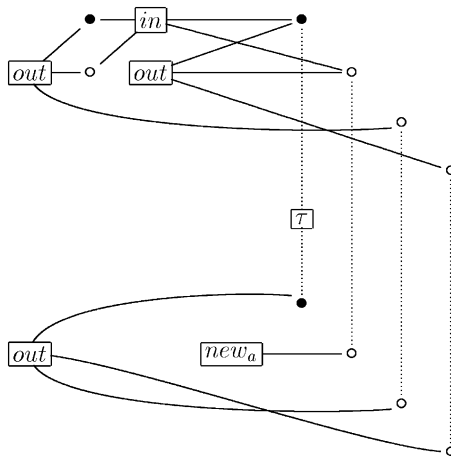
The trigger of the **reduct** rule is $id_{e \otimes a}$; it refers to the process enabled by the firing of the input prefix. The initial configuration is $(in \otimes out); (\Delta_{e \otimes a} \otimes id_a)$, denoting the simultaneous presence of an input prefix and an output; they can synchronize, since

their subjects coincide.



The application of the `flow` rule is needed to let the actions flow through parallel composition. The trigger of the rule is $\tau \otimes id_e$, and it denotes the occurrence of two processes, specifying that one of them is performing a reduction.

In order to make the flowing of information better understood, we consider again the typed basic process $x, y, w \triangleright x(z).\bar{z}y \mid \bar{x}w$, and we show how the transition $x(z).\bar{z}y \mid \bar{x}w \rightarrow_b \bar{w}y$ is simulated. The corresponding tile is depicted below: Intuitively, the `reduct` rule is first placed in parallel with the identity tile for id_a , corresponding to the introduction of the name y ; it is then instantiated with the identity tile for the term $z, y \triangleright \bar{z}y$.



We close with a correspondence result, analogous to Theorem 4.1 for the action calculus PIC, relating reduction steps and tiles.

Theorem 5.3 (Reduction as tiling). *Given P, Q basic processes, and an ambient Γ such that $\Gamma \triangleright P$ is well-typed, the following properties hold:*

- if $P \rightarrow_b Q$, then $[\Gamma \triangleright P] \xrightarrow{\tau \otimes id_{[\Gamma]}} [\Gamma \triangleright Q]$;
- if $[\Gamma \triangleright P] \xrightarrow{\tau \otimes id_{[\Gamma]}} t$, then there exists $\Gamma \triangleright R$ well-typed such that $P \rightarrow_b R$ and $t = [\Gamma \triangleright R]$.

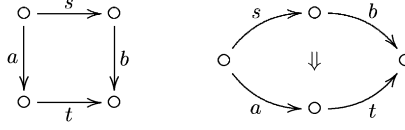


Fig. 6. A tile (on the left) and the associated *stretched* rewrite rule.

5.3. Stretching tiles

Recall that the main point in using tiles with respect to rewriting logic is the ability to express suitable constraints on the dynamic behaviour of the subcomponents of an open system. Such constraints are in general difficult to recast using (unconditional) rewrite theories, because rewriting steps can be freely contextualized. A solution is to build theories over a signature which is the disjoint union of both horizontal and vertical signatures, and then *stretching* tiles into ordinary rewrite rules, as depicted in Fig. 6. The details of the encoding of tile logic into rewriting logic are not trivial; thus, we prefer to give some intuitions, largely borrowing from [11,58], and closing the section with its application to the tile specification for CCS described in Section 5.2.1.

A few correspondence results have been proved, all of them basically stating that, given a tile system \mathcal{T} , the tile sequent $s \xrightarrow{a} t$ is entailed by \mathcal{T} if and only if the sequent $s; b \rightarrow a; t$ is entailed by $\mathcal{R}_{\mathcal{T}}$, the stretched version of our tile system, and its proof (i.e., the associated proof term) satisfies some additional constraints. A relevant fact is that, for a large class of tile systems (called *uniform*), the additional constraints reduce to checking that the source and target of the sequent can be correctly partitioned.

Let us consider now the tile system \mathcal{T}_{CCS} for CCS described in Definition 5.4. The associated (stretched) rewrite theory $\mathcal{R}_{\mathcal{T}_{\text{CCS}}}$ is the 4-tuple $\langle \Sigma_{\text{CCS}} \uplus \Sigma_p, \emptyset, L_{\text{CCS}}, \hat{R}_{\text{CCS}} \rangle$. The set of labels coincides with that of \mathcal{T}_{CCS} , while the set \hat{R}_{CCS} of labeled rules is given below.

$$\begin{aligned}
 act_{\mu} &: \underline{\mu}[\underline{\mu}(x)] \rightarrow x & res_{\alpha} &: \underline{\mu}[x \setminus \alpha] \rightarrow \underline{\mu}[x] \setminus \alpha \quad \text{for } \mu \notin \{\alpha, \bar{\alpha}\} \\
 +_{\mu}^l &: \underline{\mu}[x + y] \rightarrow \underline{\mu}[x] & +_{\mu}^r &: \underline{\mu}[x + y] \rightarrow \underline{\mu}[y] \\
 \xi_{\mu}^l &: \underline{\mu}[x | y] \rightarrow \underline{\mu}[x] | [y] & \xi_{\mu}^r &: \underline{\mu}[x | y] \rightarrow [x] | \underline{\mu}[y] \\
 \xi_{\alpha}^s &: \underline{\tau}[x | y] \rightarrow \underline{\alpha}[x] | \underline{\bar{\alpha}}[y]
 \end{aligned}$$

For notational convenience we put between brackets $[-]$ the elements of $\mathbf{A}(\Sigma_{\text{CCS}})$. Let us consider our running example, the process $P_e = ((\alpha.nil + \beta.nil) | \bar{\alpha}.nil) \setminus \alpha$ and the transition $P_e \xrightarrow{\tau} (nil | nil) \setminus \alpha$. The associated tile deduction can be seen in Section 5.2.1. The corresponding rewriting logic deduction is instead presented below.

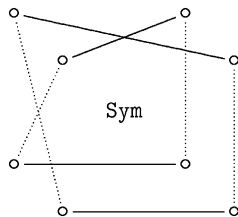
$$\begin{aligned}
 &\underline{\tau}[(\alpha.nil + \beta.nil) | \bar{\alpha}.nil] \setminus \alpha \rightarrow (\underline{\tau}[(\alpha.nil + \beta.nil) | \bar{\alpha}.nil]) \setminus \alpha \\
 &\rightarrow (\underline{\alpha}[\alpha.nil + \beta.nil] | \underline{\bar{\alpha}}[\bar{\alpha}.nil]) \setminus \alpha \rightarrow (\underline{\alpha}[\alpha.nil] | \underline{\bar{\alpha}}[\bar{\alpha}.nil]) \setminus \alpha \\
 &\rightarrow (\underline{\alpha}[\alpha.nil] | [nil]) \setminus \alpha \rightarrow ([nil] | [nil]) \setminus \alpha = [(nil | nil) \setminus \alpha].
 \end{aligned}$$

Note that we proceed top-down in the proof deduction; we have here a kind of ‘goal-oriented’ evaluation of sos inference rules, where we start with the initial configuration and the expected action. In fact, a typical query in a tile system could be: ‘Derive some of the tiles with initial configuration s and effect b ’. This corresponds to considering sequents in rewriting logic with source s ; b and to applying a rewrite step that simulates a tile computation with initial configuration s and effect b . A sequence of steps is ‘successful’ if the associated target is an element of $\mathbf{A}(\Sigma_{\text{ccs}})$.

Thus, we need in general to define some *strategies* for exploring the tree of non-deterministic rewriting steps until a successful configuration is reached. We leave all the details about the implementation of such tile strategies in rewriting logic to the papers we referred to, and we just state the following result, relating tile sequents entailed by \mathcal{T}_{ccs} and rewriting logic sequents entailed by $\mathcal{R}_{\mathcal{T}_{\text{ccs}}}$ (where $[P]$ denotes the term of $\mathbf{A}(\Sigma_{\text{ccs}})$ associated to the process P).

Theorem 5.4 (Sequent correspondence). *Given ccs processes P, Q , the sequent $[P] \xrightarrow{\mu} [Q]$ is entailed by the algebraic tile system \mathcal{T}_{ccs} if and only if the sequent $\underline{\mu}[P] \rightarrow [Q]$ is entailed by the rewrite theory $\mathcal{R}_{\mathcal{T}_{\text{ccs}}}$.*

Note however that the correspondence between tile and rewriting logic actually holds for more general versions of tile logic, called *process* and *term* tile logic, introduced in [8,10,12]. In this case, the set of inference rules is enriched by a set of *auxiliary* tiles, ensuring that both configurations and effects share a common algebraic structure. These tiles are used for taking into account the complex rearrangements of the interface of a cell that may occur during a computation, in order to deal with the interweaving between the two spatial dimensions of a tile specification. Without giving any details, we simply show below the unique auxiliary tile for process tile logic, where the horizontal and vertical categories share a common symmetric monoidal structure.



Process and term tile logic are thus powerful specification tools, generalizing their algebraic counterpart with a flexible mechanism allowing complex interactions between the two dimensions of a tile system. As an example, it is easy to recover full ccs, that is, the extension of the finite case described earlier with a *replication* operator [11]. The main result of [10,12] is the introduction of a suitable semantic framework for term tile logic, based on the categorical notion of *hyper-transformation* [4]. It allows for the study of the notion of concurrent computation in this extended setting, as already done for rewrite theories [54] and algebraic tile systems [30].

The correspondence of process and term tile logic with rewriting logic has been extended to the implementation level. In fact, taking advantage of the reflectivity features

of rewriting logic implementations [7,16,17] and of the normal forms [9] available for several classes of graphs, it is possible to make executable some significant tile specifications by translating them into the languages Maude and ELAN [11,15].

6. An informal overview of the features

This final section has the ambitious task of summing up the discussion so far, in order to provide an informal, short comparison between the different formalisms we glimpsed at, trying to find out a common layout were they can be recast. We apologize to all those authors, involved in the areas we touched upon with our paper, and whose work has been overlooked;⁸ space limitations, and the need not to let the focus wander, forced us to leave out many formalisms proposed recently.

We ask now the reader to allow for a detour, in order to give a more faithful account of the respective issues dealt with by the three proposals we analyzed. To this aim, we intend to recall some of the metaformalisms proposed in the *concurrency* area, *circa* early Eighties, for the specification of rule-based reactive and concurrent systems. While the use of *transition systems* [45,67] for the characterization of operational semantics was well-understood, a substantial amount of work in the literature has been devoted to the specification mechanisms needed in order to give a finitary description of a possibly *infinite* set of transitions. In other words, the notion could be rephrased this way: ‘Are there guidelines for generating inductively a possibly infinite transition system, which is able to take into account the spatial distribution and the temporal constraints of the underlying computing system’?

The SOS approach to operational semantics, as pioneered by Gordon Plotkin [71], is based on the assumption that a *labeled* transition system could be generated *via* a set of inference rules. Each rule is guided by the structure of the terms, and it denotes a conditional evolution of the computing system. In general, such conditions are global, since the premises of a rule may impose arbitrary assumptions on the behaviour of the subcomponents of the state under analysis. This is the way the original, *interleaving* operational semantics for CCS was presented [60]. Suitable extensions such as [25] have been then used for giving an account of *truly concurrent* semantics for process algebras.

The solution exemplified by Berry and Boudol with their *chemical abstract machine* (shortly, CHAM) [6] is loosely based on the firing steps approach to the semantics for Petri nets, as recalled in Section 3.2.1. A basic operation is assumed, namely multiset union, as forming the basis for a ‘soup’ of agents, representing a distributed state where those agents live and freely interact. The transition system is generated from a set of basic rules, with the addition of suitable operations for *airlock* and *cooling*. These operations allow to ‘freeze’ a state, imposing a sort of ‘encapsulation’ between its components, expressing constraints over their behaviour.

⁸ A typical example is the *span categories* by Bob Walters et al. [43,44]. They put the emphasis on the presentation of data structures as arrows of suitable enriched monoidal categories, closely related to our own gs-monoidal theories.

A third solution is exemplified by the so-called *Petri nets are monoids* paradigm [57]. The idea is to consider the transitions as elements of a (possibly commutative) monoid, freely generated from a set of basic rules, considered here as the founding elements of the monoid. Thus, transitions may recover information on the spatial distribution of a system, as shown by the correspondence results with the *process* semantics for nets [26,59]. *Structured transition systems* [18,22,29] represent an obvious generalization of the paradigm; states may enjoy now a rather complex structure, which is lifted to the level of transitions, still assuming the basic rules as the founding ingredients.

Rewriting logic (in its unconditional version) is tightly related to the structured transition system framework,⁹ since it integrates the *Petri nets are monoids* paradigm with techniques from the algebraic specification area. States are now elements of a term algebra, and the basic rules represent operations of an *algebra of proofs*. The transitions are then obtained as elements of an algebra built over those basic operators, extended with new operators lifted from the algebra over states. The use of universal algebra guarantees enough flexibility and expressive power, producing a rich formalism where to recast other logics, while at the same time offering suitable tools for the executability of specifications. In addition, the freeness of the closure guaranteed by the unconditional approach offers important properties of modularity and compositionality in the specification of a system, allowing for the incremental description, verification and optimization of the single components.

Action calculi can be instead considered as an evolution of the CHAM approach. In fact, the results on closed action calculi show that controls, more than abstraction, lie at the core of the approach, and controls offer a general way of *encapsulating* one agent into the other. As shown by the PIC example, the monoidal operator is the counterpart of the multiset union, while the prefix operator of the asynchronous π -calculus is encoded by the *box* control, allowing for the internal ‘boxing’ of basic π -processes. Such a mechanism allows a great generality in the specification procedure; it extends unconditional, gs-monoidal rewriting logic (which denotes an action calculus with controls of fixed rank 0), and it allows for the dynamic changing of the topology of the encapsulated agent. At the same time, the trade-off is in the ‘operator explosion’: Operators performing the same operation, but with different arities, are needed when decisions should be taken at the level of the whole state. This is the case for non-deterministic choice, simulated by further extending the calculus PIC with a control *choice* of variable rank [63].

Tile logic can be considered as an extension of the rewriting logic paradigm *via* the use of suitable formats for representing generic configurations and observations of open systems. A complex system can then be described as a *structured* composition of simpler entities, such that the global behaviour can be specified as a coordination (via triggers/effects) of local evolutions of the subcomponents, without requiring a centralized control. Thus, the formalism ensures properties of compositionality and modularity for the resulting specification mechanism, subsuming at the same time most of the usual sos presentations (thus getting rid of conditional rules). Moreover, the additional dimension can also be used to recast features that are typical of

⁹ Actually, they have been developed in parallel.

higher-order calculi, as shown by the treatment of names in the encoding of the asynchronous π -calculus. The trade-off lies in the complex notation that may be needed for the two-dimensional representation of process computations. Nevertheless, implementability is guaranteed by the encoding of tile logic into unconditional rewriting logic.

Appendix. On the asynchronous π -calculus

We recall the syntax for the monadic, asynchronous π -calculus, and its reduction semantics [38]. With respect to sos-style semantics, there is no explicit labeling on transitions. A suitable congruence relation over terms allows generating the transition relation from a set of rules over terms in normal form.

Definition A.1 (*Basic π -processes*). Let *Names* be a set of atomic *names*, ranged over by x, y, z, \dots , equipped with an involutive function \bar{x} . A *basic π -process* (also, a basic process) is a term generated by the following syntax:

$$P ::= 0, \bar{x}y, x(y).P, (\nu x)P, P_1 | P_2$$

and quotiented by the *structural* axioms

$$P \cong Q \quad \text{for } P, Q \text{ } \alpha\text{-convertible}$$

$$P | Q \cong Q | P$$

$$P | (Q | R) \cong (P | Q) | R \quad P | 0 \cong P$$

$$(\nu x)(\nu y)P \cong (\nu y)(\nu x)P$$

$$(\nu x)(P | Q) \cong P | (\nu x)Q \quad \text{for } x \notin \text{fn}(P).$$

We let P, Q, R, \dots range over the set $BPProc$ of basic π -processes.

We assume as usual the standard definitions for free and bound names of a process, as well as for the operation of substitution used later. Given a process P , its dynamic behaviour can be described as a reduction relation obtained by closing a set of basic rules under structural congruence.

Definition A.2 (*Reduction semantics of basic processes*). The *reduction relation* for basic processes is the relation $B_\pi \subseteq BPProc \times BPProc$, closed under structural congruence, inductively generated by the following set of axioms and inference rules:

$$\overline{x(y).P | \bar{x}w \rightarrow_b P\left\{\frac{w}{y}\right\}}$$

$$\frac{P \rightarrow_b Q}{(\nu x)P \rightarrow_b (\nu x)Q}$$

$$\frac{P \rightarrow_b Q}{P | R \rightarrow_b Q | R}$$

where $P \rightarrow_b Q$ means that $\langle P, Q \rangle \in B_\pi$.

Thus, the output operator $\bar{x}y$ is non-blocking, in the sense that it cannot prefix any other process, as it happens instead for the input operator $x(y)$. The intended meaning

for the reduction rules is intuitive. In particular, the rule for synchronization states that, if we have an output operator ready to communicate along the channel x a name z , and we have a process ready to receive such a name over the same channel x , then the result is the process where each occurrence of the place-holder y is replaced by z .

References

- [1] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy, Explicit substitutions, *J. Funct. Programming* 1 (1991) 375–416.
- [2] A. Barber, Ph. Gardner, M. Hasegawa, G. Plotkin, From action calculi to linear logic, in: M. Nielsen, W. Thomas (Eds.), *Computer Science Logic, Lecture Notes in Computer Science*, Vol. 1414, Springer, Berlin, 1998, pp. 78–97.
- [3] H. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, North-Holland, Amsterdam, 1984.
- [4] A. Bastiani, C. Ehresmann, Multiple functors I: Limits relative to double categories, *Cahiers Topologie et Géom. Différentielle* 15 (1974) 545–621.
- [5] J.A. Bergstra, J.W. Klop, Process algebra for synchronous communication, *Inform. and Comput.* 60 (1984) 109–137.
- [6] G. Berry, G. Boudol, The chemical abstract machine, *Theoret. Comput. Sci.* 96 (1992) 217–248.
- [7] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, C. Ringeissen, An overview of ELAN, in: C. Kirchner, H. Kirchner, (Eds.), *Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science*, Vol. 15, Elsevier Science, Amsterdam, 1999.
- [8] R. Bruni, Tile logic for synchronized rewriting of concurrent systems, Ph.D. Thesis, Department of Computer Science, University of Pisa, 1998.
- [9] R. Bruni, F. Gadducci, U. Montanari, Normal forms for partitions and relations, in: J.L. Fiadeiro (Ed.), *Recent Trends in Algebraic Development Techniques, Lecture Notes in Computer Science*, Vol. 1589, Springer, Berlin, 1999, pp. 31–47.
- [10] R. Bruni, J. Meseguer, U. Montanari, Process and term tile logic, Technical Report SRI-CSL-98-06, SRI International, 1998. (Also Technical Report TR-98-09, Department of Computer Science, University of Pisa.)
- [11] R. Bruni, J. Meseguer, U. Montanari, Executable tile specifications for process calculi, in: J.-P. Finance (Ed.), *Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science*, Vol. 1577, Springer, Berlin, 1999, pp. 60–76.
- [12] R. Bruni, J. Meseguer, U. Montanari, Symmetric and cartesian double categories as a semantic framework for tile logic, *Math. Struct. Comput. Sci.* 2002, in preparation.
- [13] R. Bruni, U. Montanari, Cartesian closed double categories, their lambda-notation, and the pi-calculus, in: *Logic in Computer Science*, IEEE Computer Society Press, New York, 1999, pp. 246–265.
- [14] R. Bruni, U. Montanari, Zero-safe nets: Comparing the collective and individual token approaches, *Inform. and Comput.* 156 (2000) 46–89.
- [15] R. Bruni, U. Montanari, J. Meseguer, Internal strategies in a rewriting implementation of tile systems, in: C. Kirchner, H. Kirchner (Eds.), *Rewriting Logic and its Applications, Electronic Notes in Theoretical Computer Science*, Vol. 15, Elsevier Science, Amsterdam, 1999.
- [16] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J. Quesada, Maude: specification and programming in rewriting logic. Online manual, available at URL: (<http://maude.csl.sri.com/manual>), 1999.
- [17] M.G. Clavel, S. Eker, P. Lincoln, J. Meseguer, Principles of Maude, in: J. Meseguer (Ed.), *Rewriting Logic and Applications, Electronic Notes in Computer Science*, Vol. 4, Elsevier, Amsterdam, 1996.
- [18] A. Corradini, G. Ferrari, U. Montanari, Transition systems with algebraic structure as models of computations, in: I. Guessarian (Ed.), *Semantics of Systems of Concurrent Processes, Lecture Notes in Computer Science*, Vol. 469, Springer, Berlin, 1990, pp. 185–222.
- [19] A. Corradini, F. Gadducci, A 2-categorical presentation of term graph rewriting, in: E. Moggi, G. Rosolini (Eds.), *Category Theory and Computer Science, Lecture Notes in Computer Science*, Vol. 1290, Springer, Berlin, 1997, pp. 87–105.

- [20] A. Corradini, F. Gadducci, An algebraic presentation of term graphs, via gs-monoidal categories, *Appl. Categorical Struct.* 7 (1999) 299–331.
- [21] A. Corradini, F. Gadducci, U. Montanari, Relating two categorical models of term rewriting, in: J. Hsiang (Ed.), *Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Vol. 914, Springer, Berlin, 1995, pp. 225–240.
- [22] A. Corradini, U. Montanari, An algebraic semantics for structured transition systems and its application to logic programs, *Theoret. Comput. Sci.* 103 (1992) 51–106.
- [23] V.-E. Căzănescu, Gh. Ștefănescu, Towards a new algebraic foundation of flowchart scheme theory, *Fund. Inform.* 13 (1990) 171–210.
- [24] V.-E. Căzănescu, Gh. Ștefănescu, A general result on abstract flowchart schemes with applications to the study of accessibility, reduction and minimization, *Theoret. Comput. Sci.* 99 (1992) 1–63.
- [25] P. Degano, R. De Nicola, U. Montanari, A distributed operational semantics for CCS based on condition/event systems, *Acta Inform.* 26 (1988) 59–91.
- [26] P. Degano, J. Meseguer, U. Montanari, Axiomatizing the algebra of net computations and processes, *Acta Inform.* 33 (1996) 641–667.
- [27] R. Diaconescu, K. Futatsugi, CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification, *AMAST Series in Computing*, World Scientific, Singapore, 1998.
- [28] G. Ferrari, U. Montanari, Tile formats for located and mobile systems, *Inform. and Comput.* 156 (2000) 173–235.
- [29] G. Ferrari, U. Montanari, M. Mowbray, Structured transition systems with parametric observations: minimal realizations and observational congruences, *Math. Struct. Comput. Sci.* 7 (1997) 241–282.
- [30] F. Gadducci, U. Montanari, The tile model. in: G. Plotkin, C. Stirling, M. Tofte (Eds.), *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, Cambridge, MA, 2000.
- [31] Ph. Gardner, A name-free account of action calculi, in: S. Brookes, M. Main, A. Melton, M. Mislove (Eds.), *Mathematical Foundations of Programming Semantics*, Electronic Notes in Computer Science, Vol. 1, Elsevier, Amsterdam, 1995.
- [32] Ph. Gardner, Closed action calculi, *Theoret. Comput. Sci.* 228 (1999) 77–103.
- [33] Ph. Gardner, M. Hasegawa, Types and models for higher-order action calculi, in: M. Abadi, T. Ito (Eds.), *Theoretical aspects of computer software*, Lecture Notes in Computer Science, Vol. 1281, Springer, Berlin, 1997, pp. 583–603.
- [34] M. Hasegawa, *Models of Sharing Graphs*, Ph.D. Thesis, Department of Computer Science, University of Edinburgh, 1997.
- [35] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
- [36] H.-J. Hönke, On partial algebras, in: B. Csákány, E. Fried, E.T. Schmidt (Eds.), *Universal Algebra*, *Colloquia Mathematica Societatis János Bolyai*, Vol. 29, North-Holland, Amsterdam, 1977, pp. 373–412.
- [37] H.-J. Hönke, On partial recursive definitions and programs, in: M. Karpiński (Ed.), *Fundamentals of Computation Theory*, Lecture Notes in Computer Science, Vol. 56, Springer, Berlin, 1977, pp. 260–274.
- [38] K. Honda, M. Tokoro, An object calculus for asynchronous communication, in: M. Tokoro, O. Nierstrasz, P. Wegner (Eds.), *Object-based Concurrent Computing*, Lecture Notes in Computer Science, Vol. 612, Springer, Berlin, 1991, pp. 21–51.
- [39] G. Hotz, Eine algebraisierung des syntheseproblems von schaltkreisen, I und II, *Elektron. Inf. Verarb. Kybern.* 1 (1965) 185–206, 209–231.
- [40] G. Huet, J.-J. Lévy, Computations in orthogonal rewriting systems, I, in: J.-L. Lassez, G. Plotkin (Eds.), *Computational Logic: Essays in honour of Alan Robinson*, MIT Press, Cambridge, MA, 1991, pp. 395–414.
- [41] B. Jacobs, Semantics of weakening and contraction, *Ann. Pure Appl. Logic* 69 (1994) 73–106.
- [42] A. Jeffrey, Premonoidal categories and a graphical view of programs, Online report, available at URL: <http://www.cogs.susx.ac.uk/users/alanje/premon/>.
- [43] P. Katis, N. Sabadini, R.F.C. Walters, Bicategories of processes, *J. Pure Appl. Algebra* 115 (1997) 141–178.
- [44] P. Katis, N. Sabadini, R.F.C. Walters, SPAN(Graph): a categorical algebra of transition systems, in: M. Johnson (Ed.), *Algebraic Methodology and Software Technology*, Lecture Notes in Computer Science, Vol. 1349, Springer, Berlin, 1997, pp. 307–321.

- [45] R. Keller, Formal verifications of parallel programs, *Comm. ACM* 7 (1976) 371–384.
- [46] J.W. Klop, Term rewriting systems, in: S. Abramsky, D. Gabbay, T. Maibaum (Eds.), *Handbook of Logic in Computer Science*, Vol. 1, Oxford University Press, Oxford, 1992, pp. 1–116.
- [47] A. Kock, G.E. Reyes, Doctrines in categorical logic, in: J. Barwise (Ed.), *Handbook of Mathematical Logic*, North-Holland, Amsterdam, 1977, pp. 283–313.
- [48] Y. Lafont, Equational reasoning with 2-dimensional diagrams, in: H. Comon, J.-P. Jouannaud (Eds.), *Term Rewriting*, Lecture Notes in Computer Science, Vol. 909, Springer, Berlin, 1995, pp. 170–195.
- [49] C. Laneve, U. Montanari, Axiomatizing permutation equivalence in the λ -calculus, in: H. Kirchner, G. Levi (Eds.), *Algebraic and Logic Programming*, Lecture Notes in Computer Science, Vol. 632, Springer, Berlin, 1992, pp. 350–363.
- [50] F.W. Lawvere, Functorial semantics of algebraic theories, *Proc. Nat. Acad. Sci.* 50 (1963) 869–872.
- [51] J.-J. Lévy, Optimal reductions in the lambda-calculus, in: J.P. Seldin, J.R. Hindley (Eds.), *Combinatory Logic, Lambda Calculus and Formalism: Essays in Honour of Haskell B. Curry*, Academic Press, New York, 1980, pp. 159–191.
- [52] S. Mac Lane, *Categories for the Working Mathematician*, Springer, Berlin, 1971.
- [53] N. Martí-Oliet, J. Meseguer, Rewriting logic as a logical and semantic framework, in: J. Meseguer (Ed.), *Rewriting Logic and Applications*, Electronic Notes in Computer Science, Vol. 4, Elsevier Science, Amsterdam, 1997. Available at URL: <http://www.elsevier.nl/locate/entcs/volume4.html/>. (An extended version is to appear in D. Gabbay (Ed.), *Handbook of Philosophical Logic*, Vol. B, Kluwer Academic Publishers, Dordrecht.)
- [54] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, *Theoret. Comput. Sci.* 96 (1992) 73–155.
- [55] J. Meseguer, Rewriting logic as a semantic framework for concurrency: A progress report, in: U. Montanari, V. Sassone (Eds.), *Concurrency Theory*, Lecture Notes in Computer Science, Vol. 1119, Springer, Berlin, 1996, pp. 331–372.
- [56] J. Meseguer, Membership algebra as a logical framework for equational specification, in: F. Parisi-Presicce (Ed.), *Recent Trends in Algebraic Development Techniques*, Lecture Notes in Computer Science, Vol. 1376, Springer, Berlin, 1998, pp. 18–61.
- [57] J. Meseguer, U. Montanari, Petri nets are monoids, *Inform. and Comput.* 88 (1990) 105–155.
- [58] J. Meseguer, U. Montanari, Mapping tile logic into rewriting logic, in: F. Parisi-Presicce (Ed.), *Recent Trends in Algebraic Development Techniques*, Lecture Notes in Computer Science, Vol. 1376, Springer, Berlin, 1998, pp. 219–233.
- [59] J. Meseguer, U. Montanari, V. Sassone, Representation theorems for Petri nets, in: C. Freksa, M. Jantzen, R. Valk (Eds.), *Foundations of Computer Science: Potential – Theory – Cognition*, Lecture Notes in Computer Science, Vol. 1337, Springer, Berlin, 1997, pp. 239–249.
- [60] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, Vol. 92, Springer, Berlin, 1980.
- [61] R. Milner, *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
- [62] R. Milner, Higher-order action calculi, in: E. Börger, Y. Gurevich, K. Meinke (Eds.), *Computer Science Logic*, Lecture Notes in Computer Science, Vol. 832, Springer, Berlin, 1994, pp. 238–260.
- [63] R. Milner, Calculi for interaction, *Acta Inform.* 33 (1996) 707–737.
- [64] R. Milner, Action structures and the π -calculus, in: H. Schwichtenberg (Ed.), *Logic of Computation*, Nato ASI Series F, Vol. 157, Springer, Berlin, 1997.
- [65] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, Part I and II, *Inform. and Comput.* 100 (1992) 1–77.
- [66] E. Moggi, Notions of computation and monads, *Inform. and Comput.* 93 (1991) 55–92.
- [67] D. Park, Concurrency and automata on infinite sequences, in: P. Deussen (Ed.), *Theoretical Computer Science*, Lecture Notes in Computer Science, Vol. 104, Springer, Berlin, 1981, pp. 167–183.
- [68] D. Pavlović, Categorical logic of names and abstraction in action calculi, *Math. Struct. Comput. Sci.* 7 (1997) 619–637.
- [69] C.A. Petri, *Kommunikation mit Automaten*, Ph.D. Thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
- [70] M. Pfender, Universal algebra in s-monoidal categories, Technical Report 95-22, Department of Mathematics, University of Munich, 1974.

- [71] G. Plotkin, A structural approach to operational semantics, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
- [72] J. Power, E. Robinson, Premonoidal categories and notions of computation, *Math. Struct. Comput. Sci.* 7 (1998) 453–468.
- [73] W. Reisig, *Petri Nets: An Introduction*, EACTS Monographs on Theoretical Computer Science, Springer, Berlin, 1985.
- [74] A. Rensink, Bisimilarity of open terms, *Inform. and Comput.* 156 (2000) 345–385.
- [75] M.R. Sleep, M.J. Plasmeijer, M.C. van Eekelen (Eds.), *Term Graph Rewriting: Theory and Practice*, Wiley, New York, 1993.
- [76] B. Thomsen, A theory of higher-order communicating systems, *Inform. and Comput.* 116 (1995) 38–57.