A Tile-Based Coordination View of Asynchronous π-calculus*

GianLuigi Ferrari[1] and Ugo Montanari[2]

[1] Dipartimento di Informatica, Università di Pisa, giangi@di.unipi.it
[2] Computer Science Laboratory, SRI International, Menlo Park, ugo@csl.sri.com

Abstract. Tiles are rewrite rules with side effects, reminiscent of both Plotkin SOS and Meseguer rewriting logic rules. They are well suited for modeling coordination languages, since they can be composed both statically and dynamically via possibly complex synchronization and workflow mechanisms. In this paper, we give a tile-based bisimilarity semantics for the asynchronous π-calculus of Honda and Tokoro and prove it equivalent to the ordinary semantics. Two kinds of tiles are provided: *activity tiles* and *coordination tiles*. Activity tiles specify the basic interactions sequential processes are able to perform, without considering the operational environment where they live. Instead, coordination tiles control the global evolution of programs.

1 Introduction

Coordination is considered as a key concept for modeling and designing heterogeneous, distributed, open ended systems. It applies typically to systems consisting of a large number of software components, independently programmed in different programming languages, which may change their configuration during execution. Changes may be due to the external addition/deletion of software components and sites, or to the transmission of programs and resource references made possible by operating systems and languages supporting mobility.

Coordination languages (see e.g. [2, 7]) provide the constructs to define configurations and interaction protocols, but they assume as given the software agents being coordinated. Hence, systems are designed and developed in a structured way, starting from the basic computational components and adding suitable software modules called *coordinators*. This approach has several advantages. In fact, it increases the potential reuse of both software agents and coordinators with usually acceptable overheads. Moreover, it provides a simple but powerful technique to reason about and prove properties of systems: induction on their structure.

The main aim of this paper is to present an approach to coordination based on the *tile model* [11, 12, 13]. Tiles have been used for coordination formalisms

* Research supported in part by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione*; and Esprit Working Groups *CONFER2* and *COORDINA*. The second author is on leave from Dipartimento di Informatica, Pisa, Italy.

equipped with flexible synchronization primitives [20, 4]. Advantages of the tile model with respect to standard operational models of coordination languages are generality and formal foundations. Also it enables to compose systems both statically and dynamically, specifying possibly complex synchronization and workflow mechanisms.

In this paper, as a case study, we consider the asynchronous π-calculus [6, 14]. The π-calculus [19] is one of the best studied examples of *mobile* process calculi, namely calculi in which the communication topology among processes can dynamically evolve when computation progresses. In the π-calculus, mobility is achieved via the communication of *names* rather than of *processes*, as happens in so called *higher order* process calculi (e.g. [24]). The π-calculus has been used to model objects [25], higher order communication [23] and also higher order configuration languages [22]. The asynchronous π-calculus is a variant of the π-calculus where emission of messages is non-blocking.

We now shortly introduce the tile model. It is based on certain rewrite rules with side effects, called *tiles*, reminiscent of both SOS rules [21] and rewrite logic rules [18]. Also related are SOS contexts [16]. To match SOS notation as much as possible, here a tile has the form

$$\frac{a}{s \xrightarrow{b} s'}$$

and states that the *initial configuration* s of the system evolves to the *final configuration* s' producing an *effect* b. However the rewrite step is actually possible only if the subcomponents of s also evolve producing the *trigger* a. Both trigger and effect are called *observations*, and model the interaction, during a computation, of the system being described with its environment. More precisely, both system configurations are equipped with an *input* and an *output interface*, and the trigger just describes the evolution of the input interface from its initial to its final configuration. Similarly for the effect. It is convenient to visualize a tile as a two-dimensional structure (see Fig.1), where the horizontal dimension corresponds to the extension of the system, while the vertical dimension corresponds to the extension of the computation. Actually, we should also imagine a third dimension (the thickness of the tile), which models parallelism: configurations, observations, interfaces and tiles themselves are all supposed to consist of several subcomponents in parallel.

Both configurations and observations are assumed to be equipped with operations of parallel and sequential composition (represented by infix operators $\otimes$ and ; respectively) which allow us to build more parallel and larger components, extended horizontally for the configurations and vertically for the observations. Similarly, tiles themselves possess three operations of composition[3]:

[3] In general, tiles are also equipped with proof terms which distinguish between sequents with the same configurations and observations, but derived in different ways. Suitable axioms for normalizing proof terms are also provided [11, 12, 13]. Normalized proof terms actually model concurrent computations. Here we consider only *flat* tiles, i.e. without proof terms, since we are only interested in interleaving semantics.

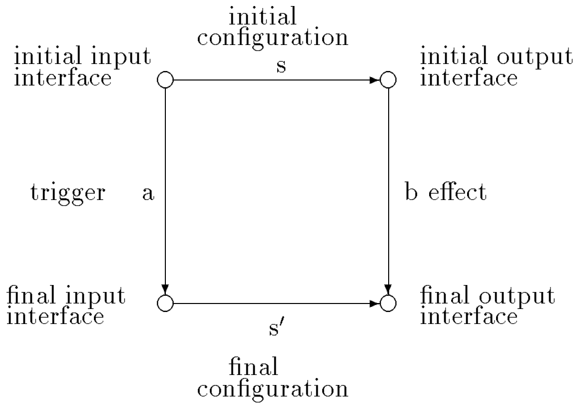

Fig. 1. A tile.

parallel (_⊗_), horizontal or *coordinate* (_∗_), and vertical or *sequential* (_·_) composition. If we consider tiles as logical sequents, it is natural to define the three operations via inference rules called *composition* rules (see Definition 6).

The operations of parallel and sequential composition are self explanatory. More interesting is the operation of coordination: the effect of the first tile acts as trigger of the second tile, and the resulting tile expresses the synchronized behavior of both. Thus horizontal composition models in a sense the dynamic creation of coordinators[4]. Computing in a *tile logic* consists of starting from a set of basic tiles called *rewrite rules* (and from a set of *auxiliary* tiles which depend on the version of the tile model at hand), and of applying the composition rules in Definition 6 in all possible ways.

A tile logic also can be seen as a double category [10] and tiles themselves as double cells. The categorical interpretation [11, 12] is useful since it makes the model more general (configurations and observations can be arrows of any category), allows for universal constructions (e.g. a tile logic is the double category freely generated by its rewrite rules) and suggests analogies with fruitful concepts of algebraic semantics, like institutions. However, the tile model is presented here in a purely logical form.

In this paper, configurations and observations are certain acyclic graphs called *agent graphs*. They are a small extension of term graphs [5], in that they may lack roots. Term graphs are like terms, but two term graphs may explicitly share some of their subterms. Thus in a term graph it is in general not allowed to copy the shared subterms to make the two terms disjoint, since this would yield a different term graph. An axiomatization of term graphs (which can be easily extended to agent graphs) in terms of gs-monoidal theories has been recently proposed by Corradini and Gadducci [8, 9], and it is reported in the Appendix.

[4] While tiles can be considered as a generalization of SOS inference rules, their algebraic structure is new. Larsen contexts [16] are analogous, but their algebraic structure is limited to ordinary terms and not axiomatized.

Agent graphs have important advantages over ordinary syntactic terms for representing configurations with shared, possibly bound names. In fact, bound names can be simply modelled as different, shared subterms which consist of the same constant. For instance, the two different CCS agents $(a.nil)\backslash a \mid (\overline{a}.nil)\backslash a$ and $(a.nil \mid \overline{a}.nil)\backslash a$ could be represented by the two term graphs $a_1.nil \mid \overline{a_2}.nil, a_1 := \nu, a_2 := \nu$ and $a.nil \mid \overline{a}.nil, a := \nu$, while, as terms, they would be identified. Another way of expressing this remark is that in agent graphs the handling of bound names (with the associated mechanisms for alpha conversion and for generating new names in mobile calculi) can be fully delegated to the underlying logic, while explicit mechanisms are needed in the case of terms. As we think it is clearly shown by the π-calculus case study in this paper, the combined use of tiles and agent graphs frees mobile calculi from complex name bookkeeping and its unpleasant consequences, like infinitely many names, infinite branching and the need for specialized versions of bisimilarity.

Tiles and agent graphs are also particularly well suited for the coordination approach. In fact the separation between software agents and coordinators can be reflected in the structure of rewrite rules. In this paper we distinguish two kinds of rules: activity rules and coordination rules. Activity rules describe the evolution of software agents, and they do not consider the operational context where agents live. As a consequence, they have no (i.e. empty) trigger, but only effect. Instead, coordination tiles define the behaviour of coordinators: they determine the evolution of software agents inside their operational environment by taking coordinating actions in response to actions of software agents.

As an example of an activity rule for the asynchronous π-calculus, let us consider the `Input` rule which models a message entering the system.

$$(\texttt{Input}) \qquad !(e) \xrightarrow{e:=input(e',a,b)} M(e',a,b)$$

The same rule is shown in graphical form (wires-and-boxes notation) on the left side of Fig2.

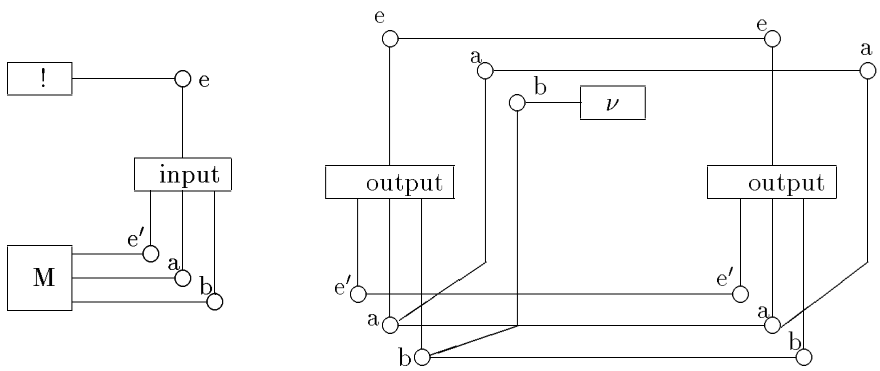

Fig. 2. Graphical representation of the rules `Input` and `Open`.

In this rule the trigger is the empty graph. The initial configuration is the agent graph with one variable and no root, which just discharges (creates?) the variable. The effect is the graph $e := input(e'a, b)$ and the final configuration is the graph $M(e', a, b)$. The initial interface of the effect is variable e, which models the sequential dependence of the present rewriting (event) from the previous events. In fact the effect replaces e in the output interface with a new event e'. Messages (resp. input observations) are represented by means of a symbol M (resp *input*) of the signature, with three variables: the current event, the subject and the object.

This rule models the case in which the incoming message has *new* names as subject and object. In fact, the input constructor creates variables a and b. Other (coordination) rules are used to instantiate subject, object or both. Notice that identifiers e, e', a and b we use in the description above have a purely local scope. The only name sharing mechanism between two configurations or tiles is via explicit composition. For instance, the fact that subject and object are new can be determined on the above rule purely by inspection.

As an example of a coordination rule, let us consider the `Open` rule which models the extrusion of a name while a message is leaving the system.

$$(\texttt{Open}) \qquad \frac{e := output(e', a, b), a, b}{e, a, b := \nu \xrightarrow{e:=output(e',a,b),a} e', a, b}$$

In this rule the precondition (trigger) is $e := output(e', a, b), a, b$, an agent graph with roots $\{e, a, b\}$ and variables $\{e', a, b\}$. The application of the rule causes the transformation of the initial configuration $e, a, b := \nu$ into the final configuration e', a, b (i.e. the restriction is removed) and the creation of effect $e := output(e', a, b), a$. Notice that while the trigger of the rule has as both source and target an event and two actions, the effect has only an event and *one* action as source. Thus the effect creates a new name, which is clearly "the same" name (following the wires) which occurred restricted in the initial configuration.

The paper is organized as follows. Section 2 introduces agent graphs, and the operations of parallel and sequential composition on them. Section 3 presents the tile model, in the simplified version needed in the paper, while Section 4 describes the asynchronous π-calculus. Section 5 defines a tile rewrite system for it, and outlines its equivalence with the ordinary semantics of Section 4.

2 Agent Graphs

In this section we introduce *agent graphs*, on which is based the data stucture we use for modeling configurations and observations. Agent graphs are similar to term graphs [5, 3, 8, 9], have a nice algebraic structure and can be finitely axiomatized essentially as gs-monoidal theories [11, 12, 8, 9]. We follow a style of presentation similar to [3].

Definition 1. *(many-sorted agent graphs)* Let us consider a set K of sorts, a many-sorted signature Σ containing symbols f of type $f : s \to w$ and p of type

$p : \epsilon \to w$, being $s \in K$ and $w, \epsilon \in K^*$, with ϵ the empty string. Furthermore let V be a totally ordered infinite set of sorted *names*, a name being denoted by n and similar letters. An *agent graph* is a triple $G = (S, rt, var)$ where:

- S is a finite set of sentences, which are assignments of the form $n := f(n_1, \ldots, n_k)$ or of the form $n := n'$, or terms of the form $p(n_1, \ldots, n_k)$, with the obvious restrictions among symbol types and name sorts. In addition, every name must be assigned at most once and no cycles (with the obvious meaning) must be present;
- rt is a list, without repetitions, of all the *roots* of G, i.e. the names which appear as left members of assignments of the form $n := n'$. Roots are usually denoted by r or similar letters;
- var is a list, without repetitions, of the *variables* of G, variables being both all the names which are not assigned in S and possibly other names which do not appear in S. Variables are usually denoted by v or similar letters.

The ordering of roots and variables must respect the ordering in V. Furthermore, agent graphs are defined up to isomorphic renaming.

Given an agent graph G, let u (resp. w) be the string of sorts of the roots (resp. variables). Then G can be seen as an arrow of *type* $u \to w$. We write $G : u \to w$ and call u and w the source and target of G respectively. □

We need a concise term-like notation for representing agent graphs. Given an agent graph $G = (S, rt, var)$, we actually replace all names which are left members of an assignment and which also occur exactly another time in S. For instance, given the set $K = \{a, b\}$ of sorts, the signature $\Sigma = \{f : a \to b, g : b \to a, h : a \to ba, p : \epsilon \to a\}$ and the typed names $n_1, n_3, n_4, v_3 : a$ and $r_1, n_2, v_1, v_2 : b$, let $G : b \to bba = (S, rt, var)$ with $S = \{r_1 := v_1, p(n_1), n_1 := h(n_2, n_3), n_2 := g(n_3), n_3 := f(v_1), n_4 := f(v_2)\}$, $rt = (r_1)$ and $var = (v_1, v_2, v_3)$. Notice that, given the signature and the type of G, all the names of G are univocally typed. The term-like notation for G is as follows:
$G : b \to bba = \{r_1 := v_1, p(h(g(n_3), n_3)), n_3 := f(v_1), n_4 := f(v_2)\}$.

Notice that if we consider as standard the names in the lists of roots and variables, isomorphic renaming is restricted to the names which are neither roots nor variables, in the example only n_3 and n_4. Furthermore, in this case an agent graph is fully specified when its type and its set of sentences are given, since the length of its list of variables can be recovered from its type.

If we restrict agent graphs to signatures with symbols $f : s \to w$ only, we obtain term graphs. Notice that in this case our notation is consistent with the ordinary record notation for terms and substitutions, and coincides with it when no names are left besides roots and variables. If this is not the case, the term corresponding to a term graph can be simply obtained by anyway executing the replacements specified by the assignments. For instance given the term graph
$G' : ba \to bba = \{r_1 := v_1, r_2 := h(g(n_1), n_1), n_1 := f(v_1), n_2 := f(v_2)\}$
we obtain the tuple of terms
$T : ba \to bba = \{r_1 := v_1, r_2 := h(g(f(v_1)), f(v_1))\}$.
Notice however that the same tuple of terms corresponds to different term

graphs. For instance the term graphs
$\{r_1 := v_1, r_2 := h(g(n_1), n_1), n_1 := f(v_1)\}$ and
$\{r_1 := v_1, r_2 := h(g(f(v_1)), f(v_1)), n_1 := f(v_2)\}$ both correspond to the same term as G'.

Particularly interesting are the following agent graphs which are called *atomic*. Assume $|u| = k$ and $|w| = i$, then:
generators: for every $f : s \to w$ and $p : \epsilon \to w$ in Σ

$$f : s \to w = \{r_1 := f(v_1, \ldots, v_i)\} \text{ and } p : \epsilon \to w = \{p(v_1, \ldots, v_i)\};$$

identities: id_u

$$id_u : u \to u = \{r_1 := v_1, \ldots, r_k := v_k\};$$

permutations: $\rho_{u,w}$

$$\rho_{u,w} : uw \to wu = \{r_1 := v_{k+1}, \ldots, r_i := v_{k+i}, r_{1+i} := v_1, \ldots, r_{k+i} := v_k\};$$

duplicators: ∇_u

$$\nabla_u : uu \to u = \{r_1 := v_1, \ldots, r_k := v_k, r_{k+1} := v_1, \ldots, r_{2k} := v_k\};$$

dischargers: $!_u$

$$!_u : \epsilon \to u = \{\}.$$

We now introduce two operations on agent graphs. The *sequential composition* of two agent graphs is obtained by gluing the list of variables of the first graph with the list of roots of the second, and it is defined only if their types are equal. The *parallel composition* instead is always defined, and it is a sort of disjoint union where root and variable lists are concatenated.

Definition 2. *(sequential and parallel composition of agent graphs)* Given two agent graphs $G_1 : u \to w = (S_1, rt_1, var_1)$ and $G_2 : w \to z = (S_2, rt_2, var_2)$, let us take two instances in their isomorphism classes such that $var_1 = rt_2$ and that no other names are shared between G_1 and G_2. Furthermore, let S be the set of clauses in S_2 of the form $r := n$ and let σ be the corresponding name substitution. The *sequential composition* of G_1 and G_2 is the agent graph $G_1; G_2 : u \to z = (S_1\sigma \cup S_2 \setminus S, rt_1, var_2)$.

Given two agent graphs $G_1 : u_1 \to w_1 = (S_1, rt_1, var_1)$ and $G_2 : u_2 \to w_2 = (S_2, rt_2, var_2)$, let us take two instances in their isomorphism classes such that no names are shared between G_1 and G_2. The *parallel composition* of G_1 and G_2 is the agent graph $G_1 : u_1u_2 \to w_1w_2 = (S_1 \cup S_2, rt_1\ rt_2, var_1\ var_2)$. □

The following theorem is an easy extension of a similar result proved in [8, 9] for term graphs.

Theorem 3. (decomposition of agent graphs) *Every agent graph can be obtained by evaluating an expression containing only atomic agents as constants, and sequential and parallel composition as operators.* □

For instance the agent graph G of our previous example can be represented as:
$G : b \to bba = \{r_1 := v_1; p(h(g(n_3), n_3)), n_3 := f(v_1), n_4 := f(v_2)\} =$
$((id_b \otimes (p; h; (g \otimes id_a); \nabla_a; f)); \nabla_b) \otimes (!_a; f) \otimes !_a$.

The following theorem, which is also an easy generalization of a result by Corradini and Gadducci [8, 9], gives a characterization of agent graphs in terms of *gs-monoidal theories.* A gs-monoidal theory is a logical theory similar to, but weaker then, the algebraic theory of terms and substitutions.

Theorem 4. (characterization of agent graphs) *The agent graphs on the signature Σ are the arrows of the opposite* gs-monoidal theory $\mathbf{GS^{op}}(\Sigma^{op})$ *generated by Σ^{op} (i.e. Σ with the opposite types for each symbol).* □

We call *agent theory on Σ* (in short $\mathbf{AG}(\Sigma)$) the theory of agent graphs on the signature Σ. In the Appendix we give the finitary axiomatization of gs-monoidal theories presented (in the one-sorted case) in [11, 12, 8, 9].

3 The Tile Model

We now describe the basic features of the *tile model*, in the version where both configurations and observations are agent graphs. Our presentation follows [12], but is made simpler, since the tile sequents we have here (which could be called *flat* sequents) have no associated proof term. However in the following we will call them simply tile sequents.

Definition 5. *(tile sequent, tile rewrite system)* Let Σ_h and Σ_v be two (many sorted) agent signatures, called the *horizontal* and the *vertical* signature respectively, on the same set of sorts K.

A Σ_h-Σ_v *tile sequent* is a quadruple $s \xrightarrow[b]{a} t$, where $s : u \to w$ and $t : y \to z$ are agent graphs on Σ_h, while $a : u \to y$ and $b : w \to z$ are agent graphs on Σ_v. Graphs s, t, a and b are called respectively the *initial configuration*, the *final configuration*, the *trigger* and the *effect* of the tile. Trigger and effect are called observations. Sort strings u, w, y and z are called respectively the *initial input interface*, the *initial output interface*, the *final input interface* and the *final output interface*.

A *tile rewrite system* (TRS) $\mathcal{R}$ is a triple $\langle \Sigma_h, \Sigma_v, R\rangle$, where R is a set of Σ_h-Σ_v sequents called *rewrite rules.* □

A TRS $\mathcal{R}$ can be considered as a logical theory, and new sequents can be derived from it via certain inference rules.

Definition 6. *(tile logic)* Let $\mathcal{R} = \langle \Sigma_h, \Sigma_v, R\rangle$ be a TRS. Then we say that $\mathcal{R}$ *entails* the class $\mathbf{R}$ of the *tile sequents* $s \xrightarrow[b]{a} t$ obtained by finitely many applications of the following inference rules *basic rules:*

$$(generators)\ \frac{s \xrightarrow[b]{a} t \in R}{s \xrightarrow[b]{a} t \in \mathbf{R}}$$

$$(h\text{-}refl)\ \frac{s : u \to w \in \mathbf{AG}(\Sigma_h)}{id_s = s \xrightarrow[id_w]{id_u} s \in \mathbf{R}} \qquad (v\text{-}refl)\ \frac{a : u \to w \in \mathbf{AG}(\Sigma_v)}{id_a = id_u \xrightarrow[a]{a} id_w \in \mathbf{R}};$$

composition rules:

$$(p\text{-}comp)\ \frac{\alpha = s \xrightarrow[b]{a} t, \alpha' = s' \xrightarrow[b']{a'} t' \in \mathbf{R}}{\alpha \otimes \alpha' = s \otimes s' \xrightarrow[b \otimes b']{a \otimes a'} t \otimes t' \in \mathbf{R}}$$

$$(h\text{-}comp)\ \frac{\alpha = s \xrightarrow[c]{a} t, \alpha' = s' \xrightarrow[b]{c} t' \in \mathbf{R}}{\alpha * \alpha' = s; s' \xrightarrow[b]{a} t; t' \in \mathbf{R}}$$

$$(v\text{-}comp)\ \frac{\alpha = s \xrightarrow[b]{a} r, \alpha' = r \xrightarrow[b']{a'} t \in \mathbf{R}}{\alpha \cdot \alpha' = s \xrightarrow[b;b']{a;a'} t \in \mathbf{R}};$$

auxiliary rules: (permutations)

$$\rho^{0,0}_{u,w} = \rho_{u,w} \xrightarrow[id_{wu}]{\rho_{u,w}} id_{wu} \in \mathbf{R} \qquad \rho^{0,1}_{u,w} = \rho_{w,u} \xrightarrow[\rho_{u,w}]{id_{wu}} id_{wu} \in \mathbf{R}$$

$$\rho^{1,0}_{u,w} = id_{wu} \xrightarrow[id_{wu}]{\rho_{w,u}} \rho_{u,w} \in \mathbf{R} \qquad \rho^{1,1}_{u,w} = id_{wu} \xrightarrow[\rho_{w,u}]{id_{wu}} \rho_{w,u} \in \mathbf{R}.$$

□

Basic rules provide the sequents corresponding to rewrite rules, together with suitable identity tiles, whose intuitive meaning is that an element of $\mathbf{AG}(\Sigma_h)$ can be rewritten to itself using only trivial trigger and effect. Similarly for $\mathbf{AG}(\Sigma_v)$. Composition rules provide all the possible ways in which sequents can be composed, while auxiliary rules are the counterpart of the atomic permutation graphs discussed above for agent theories.

For instance a tile denoted by $\rho^{0,1}_{u,w}$ consists of a horizontal permutation on the *initial configuration* (notice the character 0 as the *first* upper index) of the tile, and of the inverse permutation on the *effect observation* (notice the character 1 as the *second* upper index). The remaining sides are identities, and similarly for the other permutation tiles.

The role of permutation tiles is to permute the names on one vertex of the tile (the initial output interface in the example), still mantaining the same connections between the adjacent agent graphs. For instance, given any tile $\alpha = s \xrightarrow[b]{a} t$ with s and b having uw as target and source respectively, the composition

$(id_s * \rho^{0,1}_{w,u}) \cdot (\alpha * id_b)$ produces the tile $\alpha' = s; \rho_{u,w} \xrightarrow[\rho_{w,u};b]{a} t$. Here two permutations have been introduced, but the connections between the two involved agent graphs, represented by the composition $s; \rho_{u,w}; \rho_{w,u}; b$, are still the same as $s; b$, since $\rho_{u,w}; \rho_{w,u} = id_{uw}$.

It is easy to see that, by horizontal and vertical composition of the basic permutation tiles, it is possible to obtain permutation tiles $\rho_1 \xrightarrow[\rho_2]{\rho_3} \rho_4$ with arbitrary permutations ρ_1, ρ_2, ρ_3 and ρ_4 on the four sides, provided that $\rho_1; \rho_2 = \rho_3; \rho_4$.

It is straightforward to extend the notion of bisimilarity to deal with our framework.

Definition 7. *(tile bisimilarity).* Let $\mathcal{R} = \langle \Sigma_h, \Sigma_v, R \rangle$ be a TRS. A symmetric equivalence relation $\equiv_b \subseteq \mathbf{AG}(\Sigma_h) \times \mathbf{AG}(\Sigma_h)$ is a *tile bisimulation* for $\mathcal{R}$ if, whenever $s \equiv_b t$ for generic s, t elements of $\mathbf{AG}(\Sigma_h)$, then for any sequent $\alpha = s \xrightarrow[b]{a} s'$ entailed by $\mathcal{R}$ there exists a corresponding one $\beta = t \xrightarrow[b]{a} t'$ with $s' \equiv_b t'$. The maximal tile bisimulation equivalence is called *tile bisimilarity*, and denoted by $\sim_t$. □

Notice that this notion of bisimilarity is more general than the ordinary one, since it applies to pairs of system components which are incomplete, while the ordinary notion applies only to closed agents.

4 The Asynchronous π-calculus

The asynchronous π-calculus [14] is a variant of the π-calculus [19] where there is no output prefixing. We assume an infinite set of channel names $a, b, c \ldots$ and, following [1], we distinguish between processes (or agents) $P, Q, \ldots$, sequential processes or guards $G, H, \ldots$ and messages M. The syntax of the calculus is specified by the following grammar:

$$\begin{aligned} P &::= M \;\Big|\; G \;\Big|\; P \mid P \;\Big|\; \nu a P \\ M &::= \overline{a}b \\ G &::= \mathtt{0} \;\Big|\; a(b).P \;\Big|\; \tau.P \;\Big|\; !G. \end{aligned}$$

In $a(b).P$ and $(\nu b)P$ the occurrences of b are binding with scope P. We assume the standard notions of free names, bound names and substitution and we denote with $\mathtt{fn}(P)$ (resp. $\mathtt{bn}(P)$) the names free (resp. bound) in P.

Processes are defined up to a *structural congruence* $\equiv$. This is the smallest congruence such that:

1. processes which differ by α-conversion are equivalent;
2. $\mid$ is associative and commutative and $\mathtt{0}$ is its identity;

3. if $a \notin \mathtt{fn}(P)$ then $P \mid (\nu a Q) \equiv \nu a(P \mid Q)$;
4. $\nu a\, \nu b\, P \equiv \nu b\, \nu a\, P$;
5. $!G \equiv G \mid !G$.

Table 1 illustrates the structural rules of the operational semantics with early instantiation. The actions α are specified as follows

$$\alpha ::= \tau \;\Big|\; \overline{a}b \;\Big|\; \overline{a}(b) \;\Big|\; ab$$

The notions of free and bound names are naturally extended to actions α.

(τ)	$\tau.P \xrightarrow{\tau} P$	$(synch)$	$\overline{a}c \mid a(b).P \xrightarrow{\tau} P[c/b]$
(in)	$\mathsf{0} \xrightarrow{ab} \overline{a}b$	(out)	$\overline{a}b \xrightarrow{\overline{a}b} \mathsf{0}$
$(open)$	$\dfrac{P \xrightarrow{\overline{a}b} P' a \neq b}{\nu b P \xrightarrow{\overline{a}(b)} P'}$	(ν)	$\dfrac{P \xrightarrow{\alpha} P' a \notin n(\alpha)}{\nu a P \xrightarrow{\alpha} \nu a P'}$
$(comp)$	$\dfrac{P \xrightarrow{\alpha} P', \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	$(cong)$	$\dfrac{P \equiv Q\; Q \xrightarrow{\alpha} Q'\; Q' \equiv P'}{P \xrightarrow{\alpha} P'}$

Table 1. Labelled transition system with early instantiation

We now present the bisimilarity semantics for the asynchronous π-calculus.

Definition 8. A symmetric relation S on processes is a bisimulation if PSQ, $P \xrightarrow{\alpha} P'$, and $\mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset$ implies $Q \xrightarrow{\alpha} Q'$ and $P'SQ'$. Let the *asynchronous bisimilarity* $\sim_{HT}$ be the maximal bisimulation.

□

Recently, Amadio, Castellani and Sangiorgi [1] provided several alternative definitions of bisimilarity for the asynchronous π-calculus, which differ in the formulation of the input clause, and proved them all equivalent to the asynchronous bisimilarity $\sim_{HT}$.

Coordination and Names. Sequential processes G are autonomous entities and are at the source of any activity. Hence, sequential processes can be seen as software components possibly distributed over a network. In this perspective, process combinators define the *coordination language*. However, in the π-calculus (both synchronous and asynchronous), syntactic process combinators (i.e. parallel composition and restriction) are not the only language features which are relevant for defining configurations. In fact *free names* are used to this purpose, since they specify the *interfaces* of processes. Therefore the result of the operation of parallel composition heavily depends on which are the homonymous ports of the components (since $\mathtt{fn}(P \mid Q) = \mathtt{fn}(P) \cup \mathtt{fn}(Q)$).

This idea of viewing the free names as the specification of the communication interface leads to a more abstract representation of π-processes, and in particular of sequential processes.

Given any sequential process G, we assume the existence of a function *std* which returns an *abstract* sequential process $std(G)$ such that

$$G \equiv std(G)(a_1, \ldots, a_n)$$

for a suitable ordering of the free names $a_1, \ldots, a_n$ of G. We want *std* to be invariant under substitution (i.e. $std(G) = std(G\sigma)$ for any free name substitution σ) and under structural equivalence (i.e. if $G \equiv G'$ then $std(G) = std(G')$). For instance *std* could be defined by first selecting a standard representative G' of the equivalence class $[G]_\equiv$ and then writing $std(G) = \lambda a_1 . \ldots \lambda a_n . G'$, where $a_1, \ldots, a_n$ are the free names G' (and of G) listed according to the order of their first occurrence in the *preorder tree walk* of the abstract syntax tree of the sequential process G'.

It is now easy to see that for any process P we have an equivalent process of the form
$P \equiv \nu a_1 \ldots \nu a_k (G_1(\mathbf{a_1}) \mid \ldots \mid G_n(\mathbf{a_n}) \mid \overline{b_1} c_1 \mid \ldots \mid \overline{b_m} c_m)$
where $G_1, \ldots, G_n$ are abstract sequential processes and $\mathbf{a_1}, \ldots, \mathbf{a_n}$ are suitable tuples of names.

5 A Tile Rewrite System for the Asynchronous π-calculus

The aim of this section is to show how the framework provided by the tile model can be applied to study coordination disciplines for distributed mobile systems specified in the asynchronous π-calculus.

In what follows we introduce the components of the tile rewrite system, i.e. sorts, horizontal signature, vertical signature and rewrite rules. While the total number of rules will be infinite, finitely many rules will suffice to describe the evolution of any process.

We will then show the coincidence of our notion of bisimilarity $\sim_t$ with the bisimilarity semantics $\sim_{HT}$ of [14]. To allow for a concise statement of the property, we will constrain the initial configurations to be sequential processes. However any process becomes a sequential process when prefixed by a "start" action.

Sorts. The first sort, e, labels a single name in every agent graph representing a configuration, and this name is shared by all sequential processes and messages. Its role is to sequentialize all the activities, since here we are modelling the interleaving version of the calculus. The symbol we choose, e, is reminiscent of the word *event*, since the only name of sort e can be considered a name generated by the last transition and corresponding to the last event. The second sort, a, is reminiscent of *action*, and the names of this sort will correspond to names of the calculus. Thus, we remind, symbols of a many sorted signature for agent graphs on our two sorts are arrows from ϵ, e or a into strings of e's and a's.

Horizontal Signature. The symbols of Σ_h are as follows.

$$A_G : \epsilon \to e\ a^{|\mathtt{fn}(G)|} \quad (Agent)$$

$$M : \epsilon \to e\ a\ a \quad (Message)$$

$$\nu : a \to \epsilon \quad (Restriction)$$

where we have symbols A_G for all abstract sequential processes G.

Vertical Signature. The symbols of Σ_v are as follows.

$$input : e \to e\ a\ a \quad (Input)$$

$$output : e \to e\ a\ a \quad (Output)$$

$$tau : e \to e \quad (Tau)$$

Rewrite Rules. To match the SOS notation as closely as possible, from now on a tile $s \xrightarrow[b]{a} t$ will be represented as

$$\frac{a}{s \xrightarrow{b} t}$$

Following the SOS convention, the antecedent (trigger) a will be omitted when it is the empty agent graph.

Since configurations and observations of rules are agent graphs, we will use for them the notation developed in Section 2. We denote a name of type e just as e, rather than as n_e. Moreover we do not want to specify explicitely root and variable names. To this purpose, we add a term $!(n)$ for every name n of the graph which is not a root, but which is not used in any assignment. If n is a variable, we sometimes omit it completely. In this way the set of assignments just specifies a partial ordering, whose maxima (resp. minima) are roots (resp. variables). Moreover, when a variable is assigned to a root, we often use the same name as both root and variable name, and we just write this name instead of the assignment. Also we assume some total ordering of names, but we do not really need to specify it. In fact, due to the presence of permutation tiles in the logic (see Def.6), tiles with all the possible orderings will be generated anyway, no matter what is the ordering of names in rewrite rules, and thus the set of generated tiles will be the same.

There are two kinds of rewrite rules: *activity* rules and *coordination* rules. Activity rules describe the evolution of a sequential process and they depend on the specific structure of the process. Instead, coordination rules coordinate the evolution of sequential processes inside the operational environment.

Activity Rules. We first present the `Input` and `Output` rules for messages. They model messages entering and leaving the system.

$$(\texttt{Input})\quad !(e) \xrightarrow{e:=input(e',a,a')} M(e',a,a')$$

$$(\texttt{Output})\ M(e,a,b) \xrightarrow{e:=output(e',a,b),a,b} !(e'),!(a),!(b)$$

We now introduce the activity rules for sequential processes. We have a rule (without trigger) for every possible τ move of a sequential process, and for every possible synchronization of a sequential process and of a message. We employ two inference rules.

The first inference rule, called `Tau`, creates a tile for every possible τ move of a sequential process G.

$$G(\mathbf{a}) \xrightarrow{\tau} \nu a_1 \dots \nu a_k (G_1(\mathbf{a_1}) \mid \dots \mid G_n(\mathbf{a_n}) \mid \overline{b_1}c_1 \mid \dots \mid \overline{b_m}c_m)$$

(`Tau`) implies

$$A_G(e,\mathbf{a}) \xrightarrow{e:=tau(e'),\mathbf{a}} \begin{array}{l} A_{G_1}(e',\mathbf{a_1}),\dots,A_{G_n}(e',\mathbf{a_n}),\\ M(e',b_1,c_1),\dots,M(e',b_m,c_m),\\ a_1:=\nu,\dots,a_k:=\nu. \end{array}$$

The second activity tile, called `Synch`, describes the reception of messages.

$$G(\mathbf{a}) \mid \bar{b}c \xrightarrow{\tau} \nu a_1 \dots \nu a_k (G_1(\mathbf{a_1}) \mid \dots \mid G_n(\mathbf{a_n}) \mid \overline{b_1}c_1 \mid \dots \mid \overline{b_m}c_m)$$

(`Synch`) implies

$$A_G(e,\mathbf{a}), M(e,b,c) \xrightarrow{e:=tau(e'),\mathbf{a},b,c} \begin{array}{l} A_{G_1}(e',\mathbf{a_1}),\dots,A_{G_n}(e',\mathbf{a_n}),\\ M(e',b_1,c_1),\dots,M(e',b_m,c_m),\\ a_1:=\nu,\dots,a_k:=\nu. \end{array}$$

As an example let us consider the sequential process

$$G(a) = a(b).\nu c(c(d).\mathtt{0} \mid \bar{b}c)$$

and the transition

$$G(a) \mid \bar{a}a' \xrightarrow{\tau} \nu c(c(d).\mathtt{0} \mid \overline{a'}c).$$

The corresponding tile is:

$$A_G(e,a), M(e,a,a') \xrightarrow{e:=tau(e'),a,a'} A_{G'}(e',c), M(e',a',c), c:=\nu, !_a(a)$$

with $G'(c) = c(d).\mathtt{0}$.

Notice that while the transition and the rule above look similar, they are actually very different, in that the names occurring in the transition are global, while those in the rule are just used in the linear notation for it. Thus if we rename the free names a and a' in the transition, we obtain a different transition, while if we rename them in the rule, provided we keep the same ordering between them, we obtain the same rule. Moreover, if we change the order, we obtain a tile which differs from the previous one only by composition with permutation tiles.

Coordination Rules. We have the following rewrite rules.

$$\texttt{(Input Obj. Inst.)}\qquad \frac{e := input(e', a, b)}{e, !(b) \xrightarrow{e:=input(e',a,b),b} e', a, b}$$

$$\texttt{(Twin Input)}\qquad \frac{e := input(e', a, b)}{e \xrightarrow{e:=input(e',a,a)} e', a, b := a}$$

$$\texttt{(Open)}\qquad \frac{e := output(e', a, b), a, b}{e, a, b := \nu \xrightarrow{e:=output(e',a,b),a} e', a, b}$$

$$\texttt{(Event Output Sharing)}\qquad \frac{e := output(e', a, b), e'', a, b}{e, a, b, e'' := e \xrightarrow{e:=output(e',a,b),a,b} e', a, b, e'' := e'}$$

The first rule, `Input Object Instantiation`, having an *input* trigger, must be synchronized with an `Input` rule. It links the object of the message created by the `Input` rule to a name existing in the system. Similarly, the second tile `Twin Input` identifies subject and object of a created message. Besides those two, we need more rules to cover all the cases: `Input Subject Instantiation`, `Input Object Instantiation II` (when the subject is already instantiated), `Input Twin Instantiation`. We have also a rule `Twin Output`. We do not show these rules since they are very similar to the first two.

The `Open` rule has already been commented in the Introduction. The last rule, `Event Output Sharing`, has an extra event name e'' in the trigger, which is identified with e in the initial and with e' in the final configuration. This rule corresponds directly to the *comp* rule in Table 1. Here we have several other sharing tiles and all of them allow for some trigger to be transmitted unmodified as effect, but sharing some name. More precisely, we have `x Output Sharing`, where `x` besides `Event` can be `Subject`, `Object`, `Event Twin` or `Event Extruding`, `Action Twin` or `Action Extruding`.

Similarly we have `x y Input Sharing`, where `x` can be empty, `Subject Instantiated`, `Object Instantiated`, `Subject/Object Instantiated`, `Twin` and also `Twin Instantiated`; while `y` can be `Event`, `Subject` or `Object`. Finally we have `Event Tau Sharing`. We do not show all these rules.

While there is an infinite number of activity tiles, it is possible to show the following finiteness result.

Proposition 9. (finiteness) *For any given sequential process G, the rewrite rules needed to entail all the tile sequents having $A_{std(G)}$ as initial configuration are finitely many.*

The proof relies on the fact that in every reachable configuration at most those abstract sequential processes can appear, which correspond to syntactic subagents of G. Thus only a finite number of activity tiles can apply.

We now prove the coincidence of our notion of bisimilarity $\sim_t$ with the bisimilarity of [14].

Proposition 10. (tile semantics vs. SOS semantics) *Let* $\{a_1, \ldots, a_m\} = \mathtt{fn}(G_1)$, $\{b_1, \ldots, b_n\} = \mathtt{fn}(G_2)$, $F = \mathtt{fn}(G_1) \cup \mathtt{fn}(G_2)$, $\{c_1, \ldots, c_h\} = F \setminus \mathtt{fn}(G_1)$ *and* $\{d_1, \ldots, d_k\} = F \setminus \mathtt{fn}(G_2)$. *Then*

$$G_1(a_1, \ldots, a_m) \sim_{HT} G_2(b_1, \ldots, b_m)$$
$$\textit{if and only if}$$
$$\{A_{G_1}(e, a_1, \ldots, a_m), !_a(c_1), \ldots, !_a(c_h)\} \sim_t \{A_{G_2}(e, b_1, \ldots, b_n), !_a(d_1), \ldots, !_a(d_k)\}$$

where G_1 *and* G_2 *are abstract sequential processes.* □

We now briefly outline the proof of Proposition 10.

1. We define a "fully typed" version of the transition relation $P \xrightarrow{\alpha} P'$ where P' remembers all the free variables of P and where an extended label α contains all these variables. bisimilarity in the fully typed LTS coincides with bisimilarity in the ordinary LTS.
2. We establish a direct correspondence $\Leftrightarrow$ between processes P and those agent graphs H on Σ_h having type $H : \epsilon \to u$, where u contains only an occurrence of sort e. We have $P \Leftrightarrow H$ iff
$P \equiv \nu a_1 \ldots \nu a_k (G_1(\mathbf{a_1}) \mid \ldots \mid G_n(\mathbf{a_n}) \mid \overline{b_1} c_1 \mid \ldots \mid \overline{b_m} c_m)$ and
$H = \{A_{G_1}(e, \mathbf{a_1}), \ldots, A_{G_n}(e, \mathbf{a_n}), M(e, b_1, c_1), \ldots, M(e, b_m, c_m),$
$a_1 := \nu, \ldots, a_k := \nu\}$.
We show that $P \Leftrightarrow H$ implies $P\sigma \Leftrightarrow H; \rho$ for every injective substitution σ, where permutation ρ is the projection of σ on the ordered set of names actually present in P. We establish also a similar correspondence $\Updownarrow$ between extended labels α and certain agent graphs O on Σ_v.
3. We show that if $P \Leftrightarrow H$ then we have $P \xrightarrow{\alpha} P'$ in the fully typed LTS iff for every permutation ρ we can entail the tile $H \xrightarrow{O;\rho} H'; \rho$, with $\alpha \Updownarrow O$ and $P' \Leftrightarrow H'$. The two equal permutations ρ are created by the permutation tiles. Permutation tiles are needed in order to "twist the wires" while simulating SOS proofs in the tile model.
4. We show that if $P_1 \Leftrightarrow H_1$ and $P_2 \Leftrightarrow H_2$, then we have that $P_1 \sim_{HT} P_2$ iff $H_1 \sim_t H_2$.

6 Conclusion

In the paper we have shown a version of the tile model particularly well suited for defining coordination models of systems with bound names, extrusion and generation of new names. As a case study, we have also presented a tile-based semantics for the asynchronous π-calculus of Honda and Tokoro and proved it equivalent to the ordinary bisimilarity semantics. An advantage of the tile approach is the full compositionality of the underlying logic, which is able to handle computations of (nonclosed) system components as they were new rewrite rules specifying complex coordinators. Another innovative aspect is related to the use of agent graphs, a simple extension of term graphs, for representing configurations and observations: no global names exist in the system, but just operators for specifying name sharing among contiguous sequential processes, and rules for controlling the behavior accordingly. All the name handling issues, including alpha conversion and new name generation are thus delegated to the underlying logic.

The case study in the paper concerns the asynchronous π-calculus, but, due to our coordination viewpoint, its only constructs relevant for our model are parallel composition, restriction and name sharing. Thus we are confident that the same approach (in particular the notion of abstract sequential process) applies to all the calculi with similar configuration structure.

Since we wanted to adhere as much as possible to the Honda-Tokoro presentation of the asynchronous π-calculus, we used very simple coordination mechanisms. In particular we delegated to activity rules also the synchronization of sequential processes. Thus the observations of our rules always contain at most one constructor (symbol) of the vertical signature. However in general triggers and effects of coordinators can be arbitrarily complicated, and their agent graphs can specify complex causal dependencies (workflows) between tasks to be executed by sequential processes or by other coordinators.

7 Acknowledgments

We would like to thank Andrea Corradini and Fabio Gadducci for their help with the basic definitions of graphs and tiles, and Carolyn Talcott for several interesting discussions on related issues.

References

1. R. Amadio, I. Castellani; D. Sangiorgi, *On Bisimulations for the Asynchronous π-calculus*. CONCUR'96, LNCS, 1996.
2. J. Andreoli, C. Hankin, D. Le Metayer (Eds), *Coordination Programming: Mechanisms, Models and Semantics*, Imperial College Press, 1996.
3. Z.M. Ariola, J.W. Klop, *Equational Term Graph Rewriting*, Fundamenta Informaticae 26, 207–240, 1996.
4. Bruni, R. and Montanari, U., *Zero-Safe Nets, or Transition Synchronization Made Simple*, to appear in Proc. Express'97, Santa Margherita, September 1997.
5. H.P. Barendregt, M.C.J.D. van Eekelrn, J.R.W. Glauert. J.R. Kennaway. M.J. Plasmeijer, M.R. Sleep, Term Graph Reduction, *Proc. PARLE*, Springer LNCS 259, 141–158, 1987.
6. G. Boudol, Asynchrony and the π-calculus (note), *Rapport de Recherche 1702*, INRIA Sophia-Antipolis, May 1992.
7. N. Carriero, D. Gelenter, *Coordination Languages and Their Significance. Communications of the ACM*, 35(2), 97–107, 1992,
8. A. Corradini, F. Gadducci, *An Algebraic Presentation of Term Graphs via Gs-Monoidal Categories*, submitted for publication. Available at http://www.di.unipi.it/ gadducci/papers/aptg.ps, 1997.

9. A. Corradini, F. Gadducci, *A 2-Categorical Presentation of Term Graph Rewriting*, Proc. CTCS'97, Springer LNCS, to appear, 1997.
10. C. Ehresmann, *Catégories Structurées*: I and II, Ann. Éc. Norm. Sup. 80, Paris (1963), 349-426; III, Topo. et Géo. diff. V, Paris (1963).
11. F. Gadducci, *On the Algebraic Approach to Concurrent Term Rewriting*, PhD Thesis, Università di Pisa, Pisa. Technical Report TD-96-02, Department of Computer Science, University of Pisa, 1996.
12. F. Gadducci, U. Montanari, *The Tile Model*, Technical Report TR-96-27, Department of Computer Science, University of Pisa, 1996.
13. F. Gadducci, U. Montanari, *Tiles, Rewriting Rules and CCS*, in Proc. 1st international workshop on Rewriting Logic and Applications, J. Meseguer Ed., ENTCS 4 (1996), pp.1-19.
14. K. Honda, M. Tokoro, *An Object Calculus for Asynchronous Communication*, In: M. Tokoro, O. Nierstrasz, P. Wegner, Eds., Object-Based Concurrent Computing, Springer LNCS 612, 21–51, 1992.
15. G. Kelly, R. Street, *Review of the Elements of 2-categories*, Lecture Notes in Mathematics 420, 75–103, 1974.
16. K.G. Larsen, L. Xinxin, *Compositionality Through an Operational Semantics of Contexts*, in Journal of Logic and Computation, vol.1, n.6, 1991 (conference version in Proc. ICALP'90, Springer-Verlag, LNCS 443, 1990).
17. F. W. Lawvere, *Functorial Semantics of Algebraic Theories*, Proc. National Academy of Science **50**, 1963, pp. 869-872.
18. J. Meseguer, *Conditional Rewriting Logic as a Unified Model of Concurrency*, Theoretical Computer Science **96**, 1992, pp. 73-155.
19. R. Milner, J. Parrow and D. Walker. *A calculus of mobile processes* (parts I and II). *Information and Computation*, 100:1–77, 1992.
20. U. Montanari and F. Rossi, Graph Rewriting and Constraint Solving for Modelling Distributed Systems with Synchronization, in: Paolo Ciancarini and Chris Hankin, Eds., Coordination Languages and Models, LNCS 1061, 1996, pp. 12-27. Full paper submitted for publication.
21. G. Plotkin, *A Structural Approach to Operational Semantics*, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
22. M. Radestock, S. Eisenbach, *Semantics of Higher Order Coordination Languages*, in *Proc. COORDINATION'96*, Springer LNCS 1061, 339-356, 1996.
23. D. Sangiorgi. *Expressing mobility in process algebras: first-order and higher-order paradigms.* PhD Thesis CST-99-93, University of Edinburgh, 1992.
24. B. Thomsen. *Plain Chocs*, *Acta Informatica*, 30, 1993.
25. D. Walker. *π-calculus semantics for object-oriented programming languages.* In *Proc. TACS'91.* Springer Verlag, 1995.

Appendix: Gs-Monoidal Theories

Here the interested reader may find the axiomatic definition (taken from [11, 12, 8, 9]) of gs-monoidal theories. They are similar to the ordinary algebraic (Lawvere) theories [17], the differences being the extended signature and the missing naturality axioms for duplicators and dischargers. Gs-monoidal theories are monoidal theories, since the naturality axiom of permutations holds instead.

Definition 11. *(graphs)* A *graph* G is a 4-tuple $\langle O_G, A_G, \delta_0, \delta_1\rangle$: O_G, A_G are sets whose elements are called respectively *objects* and *arrows* (ranged over by $a, b, \ldots$ and $f, g, \ldots$), and $\delta_0, \delta_1 : A_G \to O_G$ are functions, called respectively *source* and *target*. A graph G is *reflexive* if there exists an *identity* function $id : O_G \to A_G$ such that $\delta_0(id(a)) = \delta_1(id(a)) = a$ for all $a \in O_G$; it is *with pairing* if its class O_G of objects forms a monoid; it is *monoidal* if it is reflexive with pairing and also its class of arrows forms a monoid, such that id, δ_0 and δ_1 respect the neutral element and the monoidal operation. □

Definition 12. *(hyper-signatures)* A *hyper-signature* $\Sigma_S = \langle S, \delta\rangle$ is a set of operators S, and a function $\delta = \langle \delta_0, \delta_1\rangle : S \to \underline{\mathbb{N}} \times \underline{\mathbb{N}}$, where $\underline{\mathbb{N}}$ is the set of underlined natural numbers. □

Definition 13. *(one-sorted gs-monoidal theories)* Given a hyper-signature Σ_S, the associated *gs-monoidal theory* $\mathbf{GS}(\Sigma_S)$ is the monoidal graph with objects the elements of the commutative monoid $(\underline{\mathbb{N}}, \otimes, \underline{0})$ (where $\underline{0}$ is the neutral object and the sum is defined as $\underline{n} \otimes \underline{m} = \underline{n+m}$); and arrows those generated by the following inference rules:

$$(generators)\ \frac{f : \underline{n} \to \underline{m} \in S}{f : \underline{n} \to \underline{m} \in \mathbf{GS}(\Sigma_S)} \qquad (sum)\ \frac{s : \underline{n} \to \underline{m}, t : \underline{n}' \to \underline{m}'}{s \otimes t : \underline{n} \otimes \underline{n}' \to \underline{m} \otimes \underline{m}'}$$

$$(identities)\ \frac{\underline{n} \in \underline{\mathbb{N}}}{id_{\underline{n}} : \underline{n} \to \underline{n}} \qquad (composition)\ \frac{s : \underline{n} \to \underline{m}, t : \underline{m} \to \underline{k}}{s; t : \underline{n} \to \underline{k}}$$

$$(duplicators)\ \frac{\underline{n} \in \underline{\mathbb{N}}}{\nabla_{\underline{n}} : \underline{n} \to \underline{n} \otimes \underline{n}} \qquad (dischargers)\ \frac{\underline{n} \in \underline{\mathbb{N}}}{!_{\underline{n}} : \underline{n} \to \underline{0}}$$

$$(permutations)\ \frac{\underline{n}, \underline{m} \in \underline{\mathbb{N}}}{\rho_{\underline{n},\underline{m}} : \underline{n} \otimes \underline{m} \to \underline{m} \otimes \underline{n}}$$

Moreover, the composition operator ; is associative, and the monoid of arrows satisfies the functoriality axiom

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

whenever both sides are defined; the identity axiom $id_{\underline{n}}; s = s = s; id_{\underline{m}}$ for all $s : \underline{n} \to \underline{m}$; the monoidality axioms

$$id_{\underline{n}\otimes\underline{m}} = id_{\underline{n}} \otimes id_{\underline{m}} \qquad \rho_{\underline{n}\otimes\underline{m},\underline{p}} = (id_{\underline{n}} \otimes \rho_{\underline{m},\underline{p}}); (\rho_{\underline{n},\underline{p}} \otimes id_{\underline{m}})$$
$$!_{\underline{n}\otimes\underline{m}} = !_{\underline{n}} \otimes !_{\underline{m}} \qquad \nabla_{\underline{n}\otimes\underline{m}} = (\nabla_{\underline{n}} \otimes \nabla_{\underline{m}}); (id_{\underline{n}} \otimes \rho_{\underline{n},\underline{m}} \otimes id_{\underline{n}})$$
$$!_{\underline{0}} = \nabla_{\underline{0}} = \rho_{\underline{0},\underline{0}} = id_{\underline{0}} \qquad \rho_{\underline{0},\underline{n}} = \rho_{\underline{n},\underline{0}} = id_{\underline{n}}$$

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

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

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

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

for all $s : \underline{n} \to \underline{m}, t : \underline{p} \to \underline{q} \in S$. □

This article was processed using the LaTeX macro package with LLNCS style