Electronic Notes in Theoretical Computer Science 71 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume71.html 20 pages

Tiling transactions in rewriting logic [1]

Roberto Bruni [a,2] José Meseguer [b] and Ugo Montanari [a]

[a] *Computer Science Department, University of Pisa, Italy.*
{bruni,ugo}@di.unipi.it

[b] *CS Department, University of Illinois at Urbana-Champaign, USA.*
meseguer@cs.uiuc.edu

Abstract

We propose a modular high-level approach to the specification of transactions in rewriting logic, where the operational and the abstract views are related by suitable adjunctions between categories of tile theories and of rewrite theories.

Key words: tile logic, rewriting logic, category theory, transactions, zero-safe nets, Petri nets.

1 Introduction

The enormous growth of the World Wide Web has increased the demand for global computing applications, where the "orchestration" of the flow of data and of mobile processes is a key issue. While synchronous communication on the web is unrealistic, and thus asynchronous formal models are preferred, many applications often require a coordination layer between distributed components that are designed and implemented separately (e.g. in e-commerce or on-line auction systems). For this purpose, platforms like BizTalk and Javaspaces exploit a centralized *transaction manager* (TM) to guarantee the so-called ACID—Atomicity, Consistency, Isolation, and Durability—properties (e.g. if a transaction aborts then a consistent configuration must be restored). Nevertheless, TM's are not a panacea, since their presence raises several questions that involve both theoretical aspects and pragmatics (perhaps even ethics). For example: (1) the lack of a formal abstract model; (2) the heavy task overload on the servers running TM's; and (3) when two or more organizations are involved in a transaction, which TM should take control?

[1] Research supported by IST-2001-32747 Project AGILE, by the Italian MIUR Project COMETA, and by ONR Grant N00014-02-1-0715.
[2] The first author is also supported by an Italian CNR fellowship for research on Information Sciences and Technologies, and by the CS Department of the University of Illinois at Urbana-Champaign.

$$\frac{}{\mu.P \xrightarrow{\mu} P}\ (act_\mu)$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ (lpar_\mu)$$

$$\frac{P \xrightarrow{\mu} P'}{Q \mid P \xrightarrow{\mu} Q \mid P'}\ (rpar_\mu)$$

$$\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\bar{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ (com_\lambda)$$

structural congruence

$$\begin{array}{rrcl} (ass) & P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R \\ (sym) & P \mid Q & \equiv & Q \mid P \\ (id) & P \mid \mathbf{0} & \equiv & P \end{array}$$

rewrite rules

$$\begin{array}{rrcl} (sync_\lambda) & \lambda.P \mid \bar{\lambda}.Q & \Rightarrow & P \mid Q \end{array}$$

Figure 1. LTS vs reduction semantics illustrated.

In this paper, we propose a high-level specification formalism for distributed transactions together with a meta-theoretic approach, where two views are kept distinct, but are formally related: (i) an abstract view, where transactions are seen as atomic activities that can take place independently from the rest of the system; and (ii) a refined view, where the coordination layer is made explicit. These two views reflect the two principal ways for defining the dynamics of many calculi: (a) by defining a rewriting (or "reduction") semantics over terms up to a suitable structural congruence (in the style of the CHAM [1] or, more generally, *rewriting logic* [13]); and (b) by means of *labeled transition systems* (LTS) specified in the SOS style [17], where transition labels are the means for coordinating system components.

A typical example that illustrates the different flavor of (a) and (b) is given by the elementary CCS-like calculus with inactive process $\mathbf{0}$, action prefix $\mu.\_$ (with actions $\mu \in A \uplus \bar{A} \uplus \{\tau\}$) and parallel composition $\_ \mid \_$, whose two semantics are compared in Figure 1 (actually, reductions under action prefix must be forbidden). Suppose that a process P in the calculus above is used to model a network through which the users Q_1 and Q_2 are willing to communicate on a certain channel a. Then the SOS rules of the LTS semantics specify how the communication must be propagated through the network, while the rewriting semantics just assumes that the network can be rearranged in such a way that Q_1 and Q_2 can locally "shake-hands."

Thanks to their generality as specification and logical frameworks, we choose *rewriting logic* and *tile logic* as suitable candidates for the formal modeling of the views (i) and (ii) above, respectively.

Rewriting logic (RL) [13] not only supports the reduction paradigm, but also exploits proof terms of rewrites as first class citizens, endowing the system with an algebra of computations that can be further abstracted to characterize behavioural equivalences. Furthermore, proof terms precisely characterize concurrent computations. These features make RL an expressive semantic framework for concurrency, parallelism and interaction, and for representing other logics.

Tile logic (TL) [10] is an extension of RL that links the most interesting features of LTS and reduction semantics. Tile logic exploits a three-dimensional view of concurrent systems: the horizontal dimension (space) is devoted to the modeling of states and components; the vertical dimension (time) models labeled steps; and the third dimension (concurrency) accounts for the distribution of activities and resources. This separation of concerns makes it possible to select different flavors of TL simply by fine tuning the algebraic structure of the elements in space and time and by fixing their interplay. Recent applications of TL concern an interactive view of Logic Programming [7] and a meta-theory of concurrent semantics centered around causal and spatial aspects [6]. This paper proposes a further application area, namely the semantics of distributed transactions.

There are two more reasons motivating our choice. The first motivation is that, under reasonable circumstances, TL specifications can be translated to executable RL specifications, by exploiting reflection and meta-strategies [8] to control rewrites (see e.g. [15,4,2]). Since transactions are essentially "selected" computation patterns, they can be directly translated in meta-strategies that bridge the gap between the refined and the abstract level. The second reason is that the simplest class of tile theories (called *zero-safe nets*), where both configurations and observations are multisets of basic elements, has already been shown to extend ordinary P/T Petri nets (that are just a special case of rewrite theories) with the notion of concurrent transaction. Hence a generalization of this net-based account of transactions to arbitrary tile and rewrite theories yields a high-level and expressive specification formalism that is amenable to a large field of applications for which the net modeling would not fit adequately or would require complex encodings.

The main reference for zero-safe nets is [5]. Besides ordinary places, called *stable*, zero-safe nets come equipped with *zero places*, which are empty in any stable marking; a *transaction* is a concurrent computation which may use zero tokens as triggers, but defines an evolution between stable markings only. The abstract view of a zero-safe net N is an ordinary P/T Petri net whose places are the stable places of N, and whose transitions are the basic transactions of N. The relation between zero-safe nets and their abstract counterparts is expressed by a categorical coreflection. The paper [3] presents a distributed implementation of zero-safe nets, where centralized TM's are replaced by a fully distributed commit algorithm. Note that it is the interpreter's task to guarantee that transactions are executed correctly (or aborted if they cannot be completed). The ideas proposed in this paper can be used as a basis for more general distributed transaction algorithms. In fact, we generalize the relationship between zero-safe nets and P/T Petri nets to tile theories (system design level) and rewrite theories (abstract level) to define a general framework for transactions, where: (1) the low-level view of the system is given by a tile machine running under the ACID properties of transactions; (2) the high-level view of the system is an ordinary rewrite theory generated by the (refined) tile theories, such that there is one rewrite rule for each concurrent transaction of the tile machine.

Our main result is the definition of a coreflection between the category of rewrite theories and the category of tile theories with suitable refinement morphisms. This leads to a conceptual clarification of how TL and RL are related and gives a faithful description of the effective communication mechanism needed in rewrite rules to model coordination. As an example, we give a formal justification for the claim that in passing from the LTS to the reduction semantics in Figure 1 there is a loss of information about the way in which synchronization is achieved.

Structure of the paper. In § 2, we fix the categorical presentation of rewrite theories (§ 2.1) and of tile theories (§ 2.2). In § 3, we recall the theory of zero-safe nets and the algebraic constructions carried out in [5]. In § 4, we extend the zero-safe approach to tiles, studying the algebraic constructions for computational and abstract models. It is worth noting that the construction for zero-safe nets now becomes just a special case of the more general theory developed in this paper. Our technique is illustrated by a simple example in § 5. Conclusions are drawn in § 6. Appendices A and B recall some preliminary notion about double categories and adjunctions, respectively. Moreover, two tables by the end Appendix B summarizes all the categories and all the constructions discussed in the paper.

2 Computads and categories of computations

2.1 Rewriting logic and 2-computads

The main ingredients of RL are the signature of configurations Σ, the set of structural axioms E, and the set of rewrite rules R over the congruence classes $[t]_E$ (of Σ-terms t modulo the axioms in E). Then, proof terms form a cartesian 2-category generated by the rewrites in R via simple inference rules (see e.g. [12,13]).

Here, we give a more abstract presentation of rewriting logic by taking configurations in a (strict) monoidal category $\mathbf{C}$. We assume that the reader has some familiarity with category theory. An arrow f with domain $\mathsf{d}(f) = a$ and codomain $\mathsf{c}(f) = b$ is written $f\colon a \to b$. We denote each identity by the object name itself, arrow composition (in diagrammatic order) by $\_;\_$, the monoidal tensor product by $\_\otimes\_$ and its unit element by e. Since we always consider *strict* monoidal categories and functors, in the following we shall omit the word "strict."

The standard case follows by taking as $\mathbf{C}$ the Lawvere theory $\mathcal{L}_{\Sigma,E}$ associated with the equational theory (Σ, E) [11]. Roughly, the cartesian category $\mathcal{L}_{\Sigma,E}$ has underlined natural numbers as objects ($\underline{n}$ represents a set with n ordered variables for which we use standard names $x_1,\ldots,x_n$), and the tuples of (equivalence classes of) terms $[t]_E$ as arrows ($f\colon \underline{n} \to \underline{m}$ is an m-tuple of terms over $x_1,\ldots,x_n$), with composition given by term substitution. The cartesian product of $\mathcal{L}_{\Sigma,E}$ gives the tensor product $\otimes$ on configurations. We use the terminology *2-computads*, borrowed from [18,19], for this abstract flavor of rewrite theories.

Definition 2.1 A *2-computad* is a 4-tuple $\mathcal{C} = (\mathbf{C}, R, \mathbf{l}, \mathbf{r})$, where $\mathbf{C}$ is the monoidal category of configurations, R is a set of *rule names*, and $\mathbf{l}, \mathbf{r}\colon R \to \mathbf{C}$ are the source and target functions denoting the lefthand side and the righthand side of each rule $r \in R$, with the constraints that: (1) $\mathsf{d}(\mathbf{l}(r)) = \mathsf{d}(\mathbf{r}(r))$, and (2) $\mathsf{c}(\mathbf{l}(r)) = \mathsf{c}(\mathbf{r}(r))$.

Note that if $\mathbf{C}$ has only one object (the unit e), then sequential composition coincides with the tensor product (by monoidality and functoriality of $\otimes$), and the constraints (1–2) of Definition 2.1 are trivially satisfied.

From the computational point of view, the arrows in $\mathbf{C}$ are the configurations of the system, which can be composed in parallel ($\otimes$) and sequentially (;). Domains

$$\frac{f{:}\,a \to b \in \mathbf{C}}{f{:}\,f \Rightarrow f}\ (id)$$

$$\frac{r \in R}{r{:}\,\mathbf{l}(r) \Rightarrow \mathbf{r}(r)}\ (gen)$$

$$\frac{\alpha_1{:}\,f_1 \Rightarrow g_1,\ \alpha_2{:}\,f_2 \Rightarrow g_2}{\alpha_1 \otimes \alpha_2{:}\,f_1 \otimes f_2 \Rightarrow g_1 \otimes g_2}\ (par)$$

$$\frac{\alpha_i{:}\,f_i \Rightarrow g_i,\ i=1,2,\ \mathsf{c}(f_1)=\mathsf{d}(f_2)}{\alpha_1 * \alpha_2{:}\,f_1;f_2 \Rightarrow g_1;g_2}\ (hseq)$$

$$\frac{\alpha{:}\,f \Rightarrow g,\ \beta{:}\,g \Rightarrow h}{\alpha \cdot \beta{:}\,f \Rightarrow h}\ (vseq)$$

(a) Inference rules.

$$\begin{aligned}
\alpha \cdot g &= f \cdot \alpha = \alpha \\
\alpha * b &= a * \alpha = \alpha \\
f;g &= f * g \\
\alpha \otimes e &= e \otimes \alpha = \alpha \\
(\alpha_1 * \alpha_2) \cdot (\beta_1 * \beta_2) &= (\alpha_1 \cdot \beta_1) * (\alpha_2 \cdot \beta_2) \\
(\alpha_1 * \alpha_2) \otimes (\beta_1 * \beta_2) &= (\alpha_1 \otimes \beta_1) * (\alpha_2 \otimes \beta_2) \\
(\alpha_1 \cdot \alpha_2) \otimes (\beta_1 \cdot \beta_2) &= (\alpha_1 \otimes \beta_1) \cdot (\alpha_2 \otimes \beta_2)
\end{aligned}$$

(b) Equations.

Figure 2. The cells in $rw(\mathcal{C})$.

and codomains model, respectively, the input and output interfaces of components. A rule $r \in R$ models a basic reduction from the configuration $\mathbf{l}(r)$ to $\mathbf{r}(r)$. Each reduction r can take place independently from the context where $\mathbf{l}(r)$ resides, thus any configuration $f;\mathbf{l}(r);g$ can be rewritten to $f;\mathbf{r}(r);g$ by applying r. Moreover, given two rules $r_1, r_2 \in R$ and the configurations $\mathbf{l}(r_1);\mathbf{l}(r_2)$ and $\mathbf{l}(r_1) \otimes \mathbf{l}(r_2)$, then concurrent reductions are possible that lead to $\mathbf{r}(r_1);\mathbf{r}(r_2)$ and $\mathbf{r}(r_1) \otimes \mathbf{r}(r_2)$, respectively. This yields a 2-category whose cells are concurrent computations.

Definition 2.2 Given a 2-computad $\mathcal{C} = (\mathbf{C}, R, \mathbf{l}, \mathbf{r})$, the monoidal 2-category $rw(\mathcal{C})$ has the same objects and arrows as $\mathbf{C}$, and cells defined by the inference rules in Figure 2(a) modulo the laws of monoidal 2-categories in Figure 2(b) (valid whenever both sides of the equations are correctly defined cells).

Compositions $*$ and $\cdot$ are called *horizontal* and *vertical*, respectively, according to the graphical convention of composing configurations horizontally from left to right and computations vertically from top to bottom. For example, the rewrite r, with arguments f and inside the context g is denoted by the proof term $f * r * g$:

$$\begin{tikzcd} a \arrow[r, "f"] & \mathsf{d}(\mathbf{l}(r)) \arrow[r, bend left, "\mathbf{l}(r)"] \arrow[r, bend right, "\mathbf{r}(r)"'] \arrow[r, phantom, "\Downarrow r"] & \mathsf{c}(\mathbf{l}(r)) \arrow[r, "g"] & b \end{tikzcd}$$

Definition 2.3 A *2-computad morphism* between $(\mathbf{C}_1, R_1, \mathbf{l}_1, \mathbf{r}_1)$ and $(\mathbf{C}_2, R_2, \mathbf{l}_2, \mathbf{r}_2)$ is a pair $(\mathscr{C}, \mathscr{R})$ where $\mathscr{C}{:}\,\mathbf{C}_1 \to \mathbf{C}_2$ is a monoidal functor, and $\mathscr{R}{:}\,R_1 \to R_2$ is a function such that for any $r \in R_1$: (1) $\mathscr{C}(\mathbf{l}_1(r)) = \mathbf{l}_2(\mathscr{R}(r))$ and (2) $\mathscr{C}(\mathbf{r}_1(r)) = \mathbf{r}_2(\mathscr{R}(r))$. We let **2Comp** be the category whose objects are 2-computads and whose arrows are 2-computad morphisms (with obvious identities and pairwise composition). Moreover, we denote by $\mathbf{2Comp}^{\mathrm{c}}$ the full subcategory of **2Comp** whose objects are 2-computads with a commutative monoidal category of configurations (i.e., where the tensor product $\otimes$ is commutative).

2-computads $\mathcal{C}$ are related to their computations $rw(\mathcal{C})$ via an adjunction.

$$\begin{tikzcd} \text{initial input interface}\ \circ \arrow[r, "s"] \arrow[d, "u"'] \arrow[dr, phantom, "\alpha"] & \circ\ \text{initial output interface} \arrow[d, "v"] \\ \text{final input interface}\ \circ \arrow[r, "t"'] & \circ\ \text{final output interface} \end{tikzcd}$$

Figure 3. The tile $\alpha : s \xrightarrow[v]{u} t$.

Proposition 2.4 *Let* **2MCat** *be the category of monoidal 2-categories (as objects) and monoidal 2-functors (as arrows). The obvious forgetful functor* $\mathcal{U}_2$: **2MCat** $\to$ **2Comp** *has a left adjoint* $\mathcal{F}_2$: **2Comp** $\to$ **2MCat** *with* $\mathcal{F}_2(\mathcal{C}) \simeq rw(\mathcal{C})$.

2.2 Tile logic and D-computads

Tiles extend ordinary rewrite rules with the possibility of changing the input and output interfaces during system evolution. The way in which they are changed is expressed by arrows in a vertical category. More generally, vertical arrows model the information passed between interfaces. Graphically, this amounts to representing rules as rectangles, whence the name *tile*. The tile in Figure 3 is written $\alpha : s \xrightarrow[v]{u} t$, and states that the *initial configuration* s can evolve to the *final configuration* t, producing the *effect* v when the *trigger* u is provided by the components connected to the input interface of s. The arrows s, u, v and t form the *border* of α and are conventionally denoted by the initials of the four main compass points (e.g. $\mathbf{n}(\alpha) = s$).

Definition 2.5 A *D-computad* is a 7-tuple $\mathcal{D} = (\mathbf{H}, \mathbf{V}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$, where $\mathbf{H}$ is the monoidal category of configurations, $\mathbf{V}$ is the monoidal category of observations, T is a set of *tile names*, $\mathbf{n}, \mathbf{s} : T \to \mathbf{H}$, and $\mathbf{w}, \mathbf{e} : T \to \mathbf{V}$ are the (bidimensional) source and target functions denoting, respectively, the initial and final configurations, the trigger, and the effect of each tile $r \in T$, with the constraints that:

(i) the categories $\mathbf{H}$ and $\mathbf{V}$ have the same objects;
(ii) $\mathsf{d}(\mathbf{n}(r)) = \mathsf{d}(\mathbf{w}(r))$, for any $r \in T$;
(iii) $\mathsf{c}(\mathbf{n}(r)) = \mathsf{d}(\mathbf{e}(r))$, for any $r \in T$;
(iv) $\mathsf{d}(\mathbf{s}(r)) = \mathsf{c}(\mathbf{w}(r))$, for any $r \in T$;
(v) $\mathsf{c}(\mathbf{s}(r)) = \mathsf{c}(\mathbf{e}(r))$, for any $r \in T$.

It is immediate that any 2-computad is just a particular D-computad whose vertical category $\mathbf{V}$ of observations is the discrete category of objects in the horizontal category $\mathbf{H}$ (by taking $\mathbf{C} = \mathbf{H}$ and $\mathbf{l} = \mathbf{n}$ and $\mathbf{r} = \mathbf{s}$). Note that tile rewrites cannot be applied in arbitrary contexts. For example, the tile r can be applied to $f; \mathbf{n}(r); g$ only if its trigger $\mathbf{w}(r)$ can be coordinated with f, and its effect $\mathbf{e}(r)$ with g:

$$\begin{tikzcd} a \arrow[r, "f"] \arrow[d, dotted] & \mathsf{d}(\mathbf{n}(r)) \arrow[r, "\mathbf{n}(r)"] \arrow[d, "\mathbf{w}(r)"'] \arrow[dr, phantom, "\Downarrow r"] & \mathsf{c}(\mathbf{n}(r)) \arrow[r, "g"] \arrow[d, "\mathbf{e}(r)"] & b \arrow[d, dotted] \\ ? \arrow[r, dotted] & \mathsf{d}(\mathbf{s}(r)) \arrow[r, "\mathbf{s}(r)"] & \mathsf{c}(\mathbf{s}(r)) \arrow[r, dotted] & ? \end{tikzcd}$$

Like rewrite rules, tiles can be composed horizontally, vertically, and in parallel to generate larger steps. The three compositions are illustrated in Figure 4. Due to

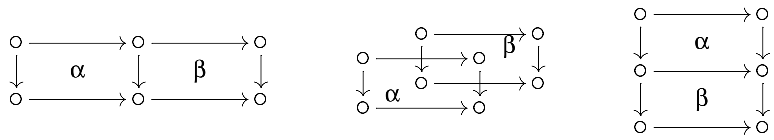

Figure 4. Horizontal, parallel and vertical tile compositions.

$$\frac{t\colon a \to b \in \mathbf{H}}{t\colon t \xrightarrow[b]{a} t}\ (hid) \qquad \frac{\alpha_1\colon s_1 \xrightarrow[w]{u} t_1,\ \alpha_2\colon s_2 \xrightarrow[v]{w} t_2}{\alpha_1 * \alpha_2\colon s_1;s_2 \xrightarrow[v]{u} t_1;t_2}\ (hseq)$$

$$\frac{u\colon a \to b \in \mathbf{V}}{u\colon a \xrightarrow[u]{u} b}\ (vid) \qquad \frac{\alpha_1\colon s \xrightarrow[v_1]{u_1} q,\ \alpha_2\colon q \xrightarrow[v_2]{u_2} t}{\alpha_1 \cdot \alpha_2\colon s \xrightarrow[v_1;v_2]{u_1;u_2} t}\ (vseq)$$

$$\frac{r \in T}{r\colon \mathbf{n}(r) \xrightarrow[\mathbf{e}(r)]{\mathbf{w}(r)} \mathbf{s}(r)}\ (gen) \qquad \frac{\alpha_1\colon s_1 \xrightarrow[v_1]{u_1} t_1,\ \alpha_2\colon s_2 \xrightarrow[v_2]{u_2} t_2}{\alpha_1 \otimes \alpha_2\colon s_1 \otimes s_2 \xrightarrow[v_1 \otimes v_2]{u_1 \otimes u_2} t_1 \otimes t_2}\ (par)$$

Figure 5. The cells in $tl(\mathcal{D})$.

space limitation, we refer to [9,15,2] for the theory of ordinary and monoidal double categories. For the reader's convenience, some basics are recalled in Appendix A. Roughly, the elements of monoidal double categories are cells analogous to the rectangle in Figure 3 and have two sequential compositions (horizontal and vertical) and a tensor product, all the operations being mutually functorial.

Definition 2.6 Given a D-computad $\mathcal{D} = (\mathbf{H}, \mathbf{V}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$, the monoidal double category $tl(\mathcal{D})$ has horizontal 1-category $\mathbf{H}$, vertical 1-category $\mathbf{V}$, and (double) cells defined by the inference rules in Figure 5, modulo the laws of monoidal double categories (cf. [2,15] for details).

Definition 2.7 A *D-computad morphism* between $\mathcal{D}_1$ and $\mathcal{D}_2$ is a triple $(\mathscr{H}, \mathscr{V}, \mathscr{T})$ such that $\mathscr{H}\colon \mathbf{H}_1 \to \mathbf{H}_2$ and $\mathscr{V}\colon \mathbf{V}_1 \to \mathbf{V}_2$ are monoidal functors, and $\mathscr{T}\colon T_1 \to T_2$ is a function such that:

(i) the functors $\mathscr{H}$ and $\mathscr{V}$ coincide on objects;

(ii) $\mathscr{H}(\mathbf{n}_1(r)) = \mathbf{n}_2(\mathscr{T}(r))$, for any $r \in T_1$;

(iii) $\mathscr{H}(\mathbf{s}_1(r)) = \mathbf{s}_2(\mathscr{T}(r))$, for any $r \in T_1$;

(iv) $\mathscr{V}(\mathbf{w}_1(r)) = \mathbf{w}_2(\mathscr{T}(r))$, for any $r \in T_1$;

(v) $\mathscr{V}(\mathbf{e}_1(r)) = \mathbf{e}_2(\mathscr{T}(r))$, for any $r \in T_1$.

We let **DComp** denote the category whose objects are D-computads and whose arrows are D-computad morphisms. Moreover, we denote by $\mathbf{DComp}^{\mathrm{c}}$ the full subcategory of **DComp** consisting of D-computads whose configurations and observations are commutative monoidal categories.

A D-computad $\mathcal{D}$ is related to the monoidal double category $tl(\mathcal{D})$ of its computations via an adjunction.

Proposition 2.8 *Let* **DMCat** *be the category of monoidal double categories (as objects) and monoidal double functors (as arrows). The obvious forgetful functor* $\mathscr{U}_{\mathrm{d}}$: **DMCat** $\to$ **DComp** *has a left adjoint* $\mathscr{F}_{\mathrm{d}}$ *with* $\mathscr{F}_{\mathrm{d}}(\mathcal{D}) \simeq tl(\mathcal{D})$.

3 Review of the zero-safe approach

P/T Petri nets are graphs whose set of nodes is the free commutative monoid $S^{\oplus}$ over the places S, and whose arcs are called transitions. A *Petri net morphism* is a graph morphism that in addition preserves the monoidal structure of markings (i.e. a graph morphism whose node component is a monoid morphism). The category **Petri** has Petri nets as objects and Petri net morphisms as arrows.

Since $S^{\oplus}$ can be regarded as a monoidal category having a unique object (the unit e), the elements of $S^{\oplus}$ as arrows, and composition given by $m;m' = m \oplus m'$, then P/T Petri nets can be regarded as 2-computads by a direct translation of transitions into rewrite rules. In fact, each $m \in S^{\oplus}$ exactly defines a multiset of places (*marking*) and any transition t with pre-set m and post-set m' can be seen as a rewrite $t: m \to m'$. Note that rewrites can be applied (concurrently) inside any larger multiset (see [16] for the RL specification of several kinds of nets).

Proposition 3.1 *The category* **Petri** *is isomorphic to the full subcategory of* $\mathbf{2Comp}^{\mathrm{c}}$ *(and hence of* **2Comp***) whose objects are 2-computads of the form* $(S^{\oplus}, T, \mathbf{l}, \mathbf{r})$.

A zero-safe net [5] is a P/T Petri net whose set of places S is partitioned into two disjoint subsets of *stable* places L and *zero* places Z, and whose transitions in a transactional way, as we explain below.

The key idea is that transitions can be fired only as part of transactions that lead from stable markings (i.e. elements of $L^{\oplus}$) to stable markings. Starting from a stable marking, the net computes by firing transitions that can fetch tokens of both kinds. After each firing, only the zero tokens in the post-set are made available for the successive firings: the stable tokens in the post-set will be made available to the system only at commit time, when no zero token involved in the transaction is left. This assumption introduces a coordination mechanism between transitions that can be implemented in distributed languages [3]. While zero tokens are useful at the specification level for modeling coordination, at the abstract level the system can be viewed as an ordinary P/T Petri net, whose places are the stable places of the system and whose transitions are the basic transactions. The advantage is that the zero-safe specification is in general simpler and more natural than its abstract view (finite specifications can yield infinitely many transactions). Furthermore, the abstract view can be defined via a categorical adjunction as recalled below from [5].

For example, let us consider the zero-safe net with two stable places a and b, a zero place z and two transitions $t_1: a \to z \oplus b$ and $t_2: b \oplus z \to a$. Then, if the initial marking is a, no transaction can be performed, as the token in b produced by a firing of t_1 would not be available immediately, and thus t_2 would not be enabled.

Instead, if the initial marking is $a \oplus b$ then t_1 can be fired first and then the token initially present in b can be used together with the token in z produced by t_1 to enable t_2 and close the transaction (whose commit releases fresh tokens in a and b).

Let **ZPetri** be the category of zero-safe nets and the obvious graph homomorphisms between them (preserving place partitioning to stable and zero), with the additional condition that distinct zero places have disjoint multisets as images.

The first step is to define a category **HCatZPetri** of zero-safe nets whose set of transitions has a (commutative) monoidal operation $\otimes$, a horizontal sequential operation $*$ (that concatenates on zero places only and behaves as the parallel composition on stable pre- and post-sets), and identities, quotiented out by suitable axioms. The morphisms of **HCatZPetri** are zero-safe net morphisms preserving all the additional structure. Horizontal composition allows building transactions that exploit the flow of zero tokens. There is an adjunction between **ZPetri** and **HCatZPetri**. We let $\mathscr{Z}$: **ZPetri** $\to$ **HCatZPetri** denote the free functor.

The second step is the characterization of basic transactions: given a transition α of a net in **HCatZPetri**, we say that $\alpha{:}m \to m'$ with m and m' stable is *prime* if it cannot be decomposed as the concurrent execution of two other non-trivial transitions. Formally, α is prime if $\alpha \neq e$ and if whenever $\alpha = \beta_1 \otimes \beta_2$ then $\beta_1 = e \vee \beta_2 = e$. Given a zero-safe net N, prime arrows in $\mathscr{Z}(N)$ are shown to exactly model the (basic) transactions of N, defining an implementation of the abstract net. Hence, a *refinement morphism* $\mathscr{R}{:}N_1 \to N_2$ is a zero-safe net morphism $\hat{\mathscr{R}}{:}N_1 \to \mathscr{Z}(N_2)$ that maps transitions either to prime arrows or to transitions of N_2.

In the example discussed above, we have $t_1 \otimes t_2{:}a \oplus b \oplus z \to b \oplus z \oplus a$, while $t_1 * t_2{:}a \oplus b \to b \oplus a$ (the token in z produced by t_1 is consumed by t_2). Moreover, $t_1 * t_2$ is a prime arrow (the only one), while e.g. $(t_1 \otimes t_1) * (t_2 \otimes t_2){:}a \oplus b \oplus a \oplus b \to b \oplus a \oplus b \oplus a$ is not a prime arrow because it can be decomposed as $(t_1 * t_2) \otimes (t_1 * t_2)$. Hence the abstract net has two places (a and b) and one transition $t{:}a \oplus b \to b \oplus a$ which can be mapped by a refinement morphism to the prime arrow $t_1 * t_2$. (We refer to [5] for more detailed examples.)

The third step is the definition of the category **ZSN** of zero-safe nets (as objects) and refinement morphisms (as arrows). In fact, refinement morphisms can be composed via a lifting that preserves primality. The category **Petri** is a coreflective subcategory of **ZSN**. Moreover, the right adjoint $\mathscr{A}_z$: **ZSN** $\to$ **Petri** maps zero-safe nets to their abstract counterparts, and the counit of the adjunction maps transitions of the abstract net to the transactions they represent. The properties of adjunctions show that $\mathscr{Z}$ and $\mathscr{A}_z$ are the "best" feasible constructions (up to isomorphism).

4 Zero-safe rewrite theories

We first explain in detail the analogy between zero-safe nets and tiles, and then generalize the constructions in [5] to tiles and rewrite theories.

4.1 Zero-safe nets as tiles

As noticed at the begininning of Section 3, the free commutative monoid $S^{\oplus}$ over the places S can be seen as a category with a unique object e. Moreover, if L is the set of stable places and Z is the set of zero places of a zero-safe net, it is easy to see that $(L \uplus Z)^{\oplus} \simeq L^{\oplus} \times Z^{\oplus}$. We already noticed that P/T Petri nets are just 2-computads of the form $(L^{\oplus}, T, \mathbf{l}, \mathbf{r})$, for $\mathbf{l}$ and $\mathbf{r}$ the pre- and post-set functions, and that the notion of net morphism coincides with that of 2-computad morphism (Proposition 3.1). Analogously, a zero-safe net can be regarded as the D-computad $(L^{\oplus}, Z^{\oplus}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$ where: (i) the pre-set of $t \in T$ is $\mathbf{n}(t) \oplus \mathbf{w}(t)$; and (ii) the post-set of $t \in T$ is $\mathbf{s}(t) \oplus \mathbf{e}(t)$. Then, it can be easily verified that the additional algebraic structure of transitions in the objects of **HCatZPetri** is just given by the ordinary identity, parallel and horizontal composition of tiles (but note that here the parallel composition is commutative). For example, if a_i, b_i are stable places, z is a zero place, $t_1 \colon a_1 \xrightarrow[z]{e} b_1$ is a transition from a_1 to $b_1 \oplus z$ and $t_2 \colon a_2 \xrightarrow[e]{z} b_2$ is a transition from $a_2 \oplus z$ to b_2, then their horizontal composition $t_1 * t_2 \colon a_1 ; a_2 \xrightarrow[e]{e} b_1 ; b_2$ forms a transaction from $a_1 \oplus a_2 = a_1 ; a_2$ to $b_1 \oplus b_2 = b_1 ; b_2$.

However, at the morphism level, **DComp** is more permissive than **ZPetri**, because the images of two distinct vertical arrows (e.g. zero places) are not necessarily disjoint multisets. This property is central to the lifting of refinement morphisms used in **ZSN** for arrow composition. Thus **ZPetri** is strictly included in the full subcategory of **DComp** whose objects are all the D-computads of the form $(L^{\oplus}, Z^{\oplus}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$. To make the correspondence more precise, we can restrict D-computad morphisms to satisfy an extended notion of the disjoint image property.

Definition 4.1 A D-computad morphism $(\mathscr{H}, \mathscr{V}, \mathscr{T})$ from $\mathcal{D}_1$ to $\mathcal{D}_2$ is *disjoint* if the functor $\mathscr{V}$ is injective on objects and faithful on arrows. We call **ZComp** the category of D-computads as objects and disjoint D-computad morphisms as arrows, and we let $\mathbf{ZComp}^{\mathrm{c}}$ denote the full subcategory of **ZComp** whose objects are D-computads over commutative monoidal categories of configurations and observations.

Proposition 4.2 *The category* **ZPetri** *is naturally isomorphic to the full subcategory of* $\mathbf{ZComp}^{\mathrm{c}}$ *whose objects are D-computads of the form* $(L^{\oplus}, Z^{\oplus}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$.

As exemplified by the construction $tl(\mathcal{D})$, D-computads have standard horizontal and parallel compositions, hence we can define the category **HCatZComp**, where: (1) the objects are D-computads whose set of tiles possesses a monoidal operation $\otimes$ and horizontal composition $*$ with horizontal identities for observations (but neither the vertical composition $\cdot$ nor the vertical identities for configurations are considered); and (2) the arrows are disjoint D-computad morphisms preserving all the additional structure. We let $\mathbf{HCatZComp}^{\mathrm{c}}$ be the full subcategory of **HCatZComp** whose D-computads have a commutative parallel composition $\otimes$.

Proposition 4.3 *The category* **HCatZPetri** *is isomorphic to the full subcategory of* $\mathbf{HCatZComp}^{\mathrm{c}}$ *whose objects are D-computads of the form* $(L^{\oplus}, Z^{\oplus}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$.

Proposition 4.4 *There is an obvious adjunction between* **ZComp** *and* **HCatZComp** *that builds the horizontal computations of tiles. We let* $\mathscr{D}$ *denote the free functor. Analogously, there is an adjunction between* $\mathbf{ZComp}^{\mathrm{c}}$ *and* $\mathbf{HCatZComp}^{\mathrm{c}}$ *and we let* $\mathscr{D}^{\mathrm{c}}$ *denote the corresponding free functor. Then, the diagram of functors and obvious embeddings*

$$\begin{tikzcd}
\mathbf{ZPetri} \arrow[r, "\mathscr{Z}"] \arrow[d, hook] \arrow[dr, phantom, "\simeq"] & \mathbf{HCatZPetri} \arrow[d, hook] \\
\mathbf{ZComp}^{\mathrm{c}} \arrow[r, "\mathscr{D}^{\mathrm{c}}"'] & \mathbf{HCatZComp}^{\mathrm{c}}
\end{tikzcd}$$

commutes (up to natural isomorphism).

In Section 4.2 we show how to generalize the notion of refinement morphism in such a way that the coreflection between **Petri** (abstract view) and **ZSN** (specification view) can be properly extended to rewrite and tile theories.

4.2 From tiles to transactional rewrite rules

The idea is that, starting from a given configuration, double computads can begin rewriting it, producing observations that must be coordinated in the continuation of the transaction. Enabled rewrites can be executed concurrently. A transaction is completed when all actions have been coordinated (the global trigger and effect must be identities, as the transaction can be executed in isolation). At the abstract level, each transaction is thus an ordinary rewrite rule. The bidimensional representation of tiles marks a clear distinction between system configurations and the structure involved in the coordination of rewrites. Conceptually, this resembles the zero-safe approach, and the abstract view can be defined by generalizing the algebraic construction based on refinement morphisms. The first step is to generalize the notion of primality, so as to characterize the basic transactions.

Definition 4.5 Given a tile α of a D-computad in **HCatZComp**, we say that α is *prime* if it cannot be decomposed as the concurrent execution of two other non-trivial tiles. Formally, $\alpha\colon s \xrightarrow[b]{a} t$, $\alpha \neq e$ is prime if a and b are identities and

$$\alpha = \beta_1 \otimes \beta_2 \quad \Longrightarrow \quad \beta_1 = e \vee \beta_2 = e.$$

Unfortunately the above constraint is not strong enough for guaranteeing that prime arrows represent atomic activities. In fact, suppose that $\alpha = \beta_1 * \beta_2$ with β_1 and β_2 prime, such that the trigger of β_2 (and hence the effect of β_1, which must be equal) is an identity arrow, then it would not be correct to assume that β_1 and β_2 are interacting in the same transactions (unless β_1 or β_2 are object identities). The difference w.r.t. the case of zero-safe nets is due to the fact that in **HCatZPetri**, if $\alpha = \beta_1 * \beta_2$ and the trigger of β_2 is e (the only possible identity), then $\alpha = \beta_1 \otimes \beta_2$ and the normal constraint can be applied. To guarantee atomicity, we must avoid any possible embedding between basic transactions.

Definition 4.6 A prime tile $\alpha_1 : s_1 \xrightarrow[b_1]{a_1} t_1$ is *elementary* if

$$\alpha_1 = \beta_1 * (u_1 \otimes \alpha_2 \otimes u_2) * \beta_2 \text{ with } \alpha_2 : s_2 \xrightarrow[b_2]{a_2} t_2 \quad \Longrightarrow \quad \alpha_1 = \alpha_2 \vee \alpha_2 = a_2.$$

Since $*$ and $\otimes$ are the only operations for composing tiles in **HCatZPetri**, the context $\beta_1 * (u_1 \otimes \_ \otimes u_2) * \beta_2$, where u_1 and u_2 are suitable horizontal identities, models the more general situation for embedding a transaction inside another. Of course, identities of objects like a_2 are not considered as transactions and can be used in elementary tiles.

Definition 4.7 A *computad refinement morphism* $\mathscr{M} : \mathcal{D}_1 \to \mathcal{D}_2$ is a disjoint D-computad morphism $\hat{\mathscr{M}} : \mathcal{D}_1 \to \mathscr{D}[\mathcal{D}_2]$ sending tiles either to tiles of $\mathcal{D}_2$ or to elementary elements of $\mathscr{D}[\mathcal{D}_2]$.

Lemma 4.8 *Given a computad refinement morphism $\mathscr{M} : \mathcal{D}_1 \to \mathcal{D}_2$, let us denote by $\widetilde{\mathscr{M}} : \mathscr{D}[\mathcal{D}_1] \to \mathscr{D}[\mathcal{D}_2]$ its unique extension in* **HCatZComp** *by means of the adjunction $\mathscr{D}$. Then, $\widetilde{\mathscr{M}}$ preserves elementary tiles.*

Proof (Sketch) We must show that, if α is elementary in $\mathscr{D}[\mathcal{D}_1]$, then $\widetilde{\mathscr{M}}(\alpha)$ is also elementary. We fix a representation of α as the horizontal composition of n tiles of the form $u_i \otimes \alpha_i \otimes v_i$ for $i = 1..n$, where the α_i's are basic tiles in $\mathcal{D}_1$ and then we proceed by contradiction by showing that if

$$\widetilde{\mathscr{M}}(\alpha) = \Big(\mathscr{M}(u_1) \otimes \mathscr{M}(\alpha_1) \otimes \mathscr{M}(v_1)\Big) * \ldots * \Big(\mathscr{M}(u_n) \otimes \mathscr{M}(\alpha_n) \otimes \mathscr{M}(v_n)\Big)$$

is not elementary, then, by exploiting the faithfulness of disjoint D-computad morphisms, α also can be shown to be non-elementary, contradicting the hypothesis. (The key fact is that each $\mathscr{M}(\alpha_i)$ must be a basic tile of $\mathcal{D}_2$, by elementarity of α.)

Thanks to Lemma 4.8, the composition of two computad refinement morphisms $\mathscr{M}_1 : \mathcal{D}_1 \to \mathcal{D}_2$ and $\mathscr{M}_2 : \mathcal{D}_2 \to \mathcal{D}_3$ is defined as the morphism $\mathscr{M}_1 ; \widetilde{\mathscr{M}_2}$, and it is again a computad refinement morphism. Thus, together with the obvious identities, computad refinement morphisms form a category.

Definition 4.9 The category **RComp** has D-computads as objects and computad refinement morphisms as arrows.

The analogy between nets and computads can now be fully exploited, leading to the main result of the paper.

Theorem 4.10 *The category* **2Comp** *is a coreflective subcategory of* **RComp**.

Proof (Sketch) First we show that the obvious inclusion $\mathscr{I}$ of **2Comp** into **RComp** is full and faithful. If $\mathcal{D}$ is a 2-computad, then the elementary arrows of $\mathscr{D}[\mathcal{D}]$ are just the rewrite rules of $\mathcal{D}$. This means that, given any 2-computads $\mathcal{D}_1$ and $\mathcal{D}_2$, any computad refinement morphism is just a 2-computad morphism. On the other hand, it is obvious that any 2-computad morphism is also a computad refinement morphism, because it maps transitions into transitions. Next, we must show that $\mathscr{I}$ has a right adjoint $\mathscr{A}_{\mathrm{d}}$. Given a D-computad $\mathcal{D} = (\mathbf{H}, \mathbf{V}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$, let $\mathscr{A}_{\mathrm{d}}[\mathcal{D}]$

$\mathtt{act}_\mu{:}\mu.x_1 \xrightarrow[\mu]{1} x_1 \quad \mathtt{lpar}_\mu{:}x_1 \mid x_2 \xrightarrow[\mu]{\mu\otimes\underline{1}} x_1 \mid x_2 \quad \mathtt{rpar}_\mu{:}x_1 \mid x_2 \xrightarrow[\mu]{\underline{1}\otimes\mu} x_1 \mid x_2 \quad \mathtt{com}_\lambda{:}x_1 \mid x_2 \xrightarrow[\underline{1}]{\lambda\otimes\bar{\lambda}} x_1 \mid x_2$

Figure 6. Tiles for the SOS rules in Figure 1

be the D-computad having the same horizontal 1-category of $\mathcal{D}$, the discrete vertical 1-category given by the objects of $\mathbf{V}$, and as tiles all the elementary tiles of $\mathscr{D}[\mathcal{D}]$ (with obvious borders). Since the vertical 1-category of $\mathscr{A}_{\mathrm{d}}[\mathcal{D}]$ is discrete, it is obvious that its tiles are ordinary rewrite rules, and therefore $\mathscr{A}_{\mathrm{d}}[\mathcal{D}]$ is just a 2-computad. The mapping $\mathscr{A}_{\mathrm{d}}$ can be extended to a functor by mapping each computad refinement morphism $\mathscr{M}: \mathcal{D}_1 \to \mathcal{D}_2$ into its lifted version, with domain restricted to the tiles in $\mathscr{A}_{\mathrm{d}}[\mathcal{D}_1]$. The definition is correct, because the lifting preserves the "elementary" property. The proof of adjunction follows from the definition of computad refinement morphism.

The right adjoint $\mathscr{A}_{\mathrm{d}}$ characterizes the abstract behaviours of D-computads by associating with a D-computad $\mathcal{D} = (\mathbf{H}, \mathbf{V}, T, \mathbf{n}, \mathbf{s}, \mathbf{w}, \mathbf{e})$, a 2-computad $\mathscr{A}_{\mathrm{d}}[\mathcal{D}]$ having the same horizontal 1-category of $\mathcal{D}$ and as rewrite rules all the elementary tiles of $\mathscr{D}[\mathcal{D}]$ (the counit maps rewrite rules of $\mathscr{A}_{\mathrm{d}}[\mathcal{D}]$ to the tile transactions they represent).

An analogous construction is possible also when a commutative tensor product of tiles is considered, yielding the category $\mathbf{RComp}^{\mathrm{c}}$ of which $\mathbf{2Comp}^{\mathrm{c}}$ is a coreflective subcategory. We denote by $\mathscr{A}_{\mathrm{d}}^{\mathrm{c}}$ the corresponding right adjoint.

Finally, the coreflection of **Petri** in **ZSN** becomes just a special case of the more general coreflection between $\mathbf{2Comp}^{\mathrm{c}}$ and $\mathbf{RComp}^{\mathrm{c}}$.

Proposition 4.11 *The diagram of functors and straightforward embeddings*

$$\begin{tikzcd} \mathbf{ZSN} \arrow[r, "\mathscr{A}_{\mathrm{z}}"] \arrow[d, hook] \arrow[dr, phantom, "\simeq"] & \mathbf{Petri} \arrow[d, hook] \\ \mathbf{RComp}^{\mathrm{c}} \arrow[r, "\mathscr{A}_{\mathrm{d}}^{\mathrm{c}}"'] & \mathbf{2Comp}^{\mathrm{c}} \end{tikzcd}$$

commutes (up to natural isomorphism).

5 Example

To illustrate our construction, let us consider again the simple process calculus defined in the Introduction (see Figure 1). The D-computad CCS corresponding to the LTS can be easily defined by a straightforward translation of the SOS rules (see examples in [2,15]). We take the free cartesian category (Lawvere theory) **Proc** generated by the process signature as the category of configurations. The vertical category is obtained by taking the free monoidal category over the actions μ (regarded as arrows from $\underline{1}$ to $\underline{1}$). The tiles are illustrated in Figure 6.

Let us assume that a transaction should be given by the synchronization of two processes. In this case, after the synchronization, the τ action should not be propagated further, as the rest of the system can evolve independently. For this reason, *elementary transactions* (where $1 \leq i < j \leq n$)

$$\mathbb{C}_n^{i,j-i}[x_1,\ldots,\lambda.x_i,\ldots,\bar{\lambda}.x_j,\ldots,x_n] \Rightarrow \mathbb{C}_n^{i,j-i}[x_1,\ldots,x_n]$$

synchronization contexts (where $l,k \geq 0$ and $i,r \geq 1$)

$$\mathbb{C}_{i+l+r+k}^{i,l+r} ::= (\mathbb{L}_l^i \mid \mathbb{R}_{i+l,k}^r)$$

$$\mathbb{L}_0^1 ::= [\_1] \qquad \mathbb{R}_{n,0}^1 ::= [\_n{+}1]$$

$$\mathbb{L}_{l+1}^i ::= (\mathbb{L}_l^i \mid [\_i{+}l{+}1]) \qquad \mathbb{R}_{n,k+1}^r ::= (\mathbb{R}_{n,k}^r \mid [\_n{+}r{+}k{+}1])$$

$$\mathbb{L}_l^{i+1} ::= ([\_1] \mid \mathbb{R}_{1,l}^i) \qquad \mathbb{R}_{n,k}^{r+1} ::= ([\_n{+}1] \mid \mathbb{R}_{n+1,k}^r)$$

Figure 7. CCS abstract transactions.

in the rule com we define the effect of a synchronization to be just an identity. Note that this solution can introduce reductions under action prefixes, in the same way as the rule *sync* in Figure 1. To prevent such reductions the standard solution is to introduce a "top" operator and enforce rewriting at the top (or use order-sorted theories that distinguish between sequential and concurrent processes). Here, for the sake of simplicity, we assume that action prefix is declared as a *frozen* operator, so that the rewrite engine (e.g. the Maude interpreter) cannot rewrite under action prefixes.

By applying the construction $\mathscr{A}_{\mathrm{d}}$ to CCS we obtain a 2-computad $\mathscr{A}_{\mathrm{d}}[\mathsf{CCS}]$ that models the atomic reductions available at the abstract level of the system. The rewrite rules in $\mathscr{A}_{\mathrm{d}}[\mathsf{CCS}]$ are the elementary tiles of $\mathscr{D}[\mathsf{CCS}]$. A generic (stable) state is an arbitrary parallel composition of sequential processes (i.e., either **0** or processes guarded by action prefix). Since reductions cannot be performed under action prefixes, the relevant part of the state can be depicted as a binary tree (internal nodes are labeled by parallel composition as in the ordinary view of terms as trees) whose leaves are labeled by sequential processes.

A generic transaction requires the occurrence of two complementary tiles, say $\mathtt{act}_\lambda$ and $\mathtt{act}_{\bar{\lambda}}$, in two leaves of the tree, the subsequent propagation of their observations λ and $\bar{\lambda}$ toward the top of the tree (via $\mathtt{lpar}_\lambda$, $\mathtt{rpar}_\lambda$, $\mathtt{lpar}_{\bar{\lambda}}$, and $\mathtt{rpar}_{\bar{\lambda}}$), until their first common ancestor (i.e. the node associated with the least parallel composition enclosing both sequential processes) receives the two triggers and can coordinate them via $\mathtt{com}_\lambda$. All the other nodes in the tree do not actively participate in the transaction.

Each transaction has the form $\mathbb{C}[x_1,\ldots,\lambda.x_i,\ldots,\bar{\lambda}.x_j,\ldots,x_n] \Rightarrow \mathbb{C}[x_1,\ldots,x_n]$ for a suitable context $\mathbb{C}$ with n holes, built using only parallel composition, which defines the binary synchronization tree going from the two interacting components $\lambda.x_i$ and $\bar{\lambda}.x_j$ to the first common parallel operator enclosing them. It is worth noting that each elementary transaction is uniquely determined by its left-hand side.

Formally, the interesting contexts for elementary transactions are defined by the grammar in Figure 7. As sketched in Figure 8, a context $\mathbb{C}_n^{i,m}$ defines a synchronization tree with n leaves (the holes of the context), whose ith and jth leaves want to interact (with $j = i + m$). Since the root is the first common parallel operator enclosing the above leaves, it follows that it can be divided into two subtrees: $\mathbb{L}_l^i$

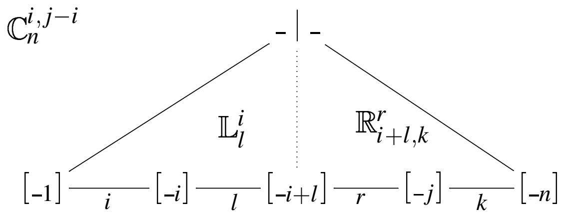

Figure 8. Synchronization contexts, graphically (where $j = i+l+r$ and $n = j+k$).

containing the first $i+l$ leaves; and $\mathbb{R}^r_{i+l,k}$ containing the remaining $r+k$ leaves, with $l+r=m$. The two subtrees are characterized by the fact that at every branching one child is a hole, while the other child is the subtree containing one of the two interacting positions. For example, we have the derivations below:

$$\begin{aligned}
&\mathbb{C}_5^{2,3} \to \mathbb{L}_1^2 \mid \mathbb{R}^2_{3,0} \to ([_1] \mid \mathbb{R}^1_{1,1}) \mid \mathbb{R}^2_{3,0} \to ([_1] \mid (\mathbb{R}^1_{1,0} \mid [_3])) \mid \mathbb{R}^2_{3,0} \to \\
&\qquad ([_1] \mid ([_2] \mid [_3])) \mid \mathbb{R}^2_{3,0} \to ([_1] \mid ([_2] \mid [_3])) \mid ([_4] \mid \mathbb{R}^1_{4,0}) \to \\
&\qquad ([_1] \mid ([_2] \mid [_3])) \mid ([_4] \mid [_5]) \\
&\mathbb{C}_5^{2,3} \to \mathbb{L}_0^2 \mid \mathbb{R}^3_{2,0} \to^* ([_1] \mid [_2]) \mid \mathbb{R}^3_{2,0} \to^* ([_1] \mid [_2]) \mid ([_3] \mid ([_4] \mid [_5]))
\end{aligned}$$

On the other hand, the context $([\_1] \mid [\_2]) \mid (([\_3] \mid [\_4]) \mid [\_5])$ cannot be generated from $\mathbb{C}_5^{2,3}$, because the subterm $[\_3] \mid [\_4]$ is inessential to the transaction between the second ($i=2$) and fifth ($j=i+3$) holes.

Thus, the abstract view of the D-computad is a 2-computad with infinitely many rewrite rules, one for each possible (binary) synchronization tree connecting two complementary action prefixes.

For example the transaction $\alpha = \left(\mathtt{act}_\lambda \otimes \underline{1} \otimes \mathtt{act}_{\bar\lambda}\right) * \left(\mathtt{lpar}_\lambda \otimes \bar\lambda\right) * \mathtt{com}_\lambda$ defines a reduction $(\lambda.x_1 \mid x_2) \mid \bar\lambda.x_3 \Rightarrow (x_1 \mid x_2) \mid x_3$ (obtained by taking $\mathbb{C}_3^{1,2} \to \mathbb{L}_1^1 \mid \mathbb{R}^1_{2,0}$), while $\beta = \left(\mathtt{act}_\lambda \otimes \underline{1} \otimes \mathtt{act}_{\bar\lambda}\right) * \left(\lambda \otimes \mathtt{rpar}_{\bar\lambda}\right) * \mathtt{com}_\lambda$ defines a reduction $\lambda.x_1 \mid (x_2 \mid \bar\lambda.x_3) \Rightarrow x_1 \mid (x_2 \mid x_3)$ (obtained by taking $\mathbb{C}_3^{1,2} \to \mathbb{L}_0^1 \mid \mathbb{R}^2_{1,0}$).

Note that concurrent transactions can take place under parallel composition (but not under prefixes, which are frozen). This is because, once an elementary transaction α has been closed by the tile $\mathtt{com}_\lambda$, the context where it is embedded does not take part in that transaction (by α being elementary) but can participate in other transactions disjoint from α.

If we take configurations in $\mathbf{Proc}/\!\equiv$ (i.e. processes modulo associativity, commutativity and identity of parallel composition), then the abstract 2-computad has still infinitely many rewrite rules (the same as before), but now many of them have the same lefthand and righthand sides. For example, the transactions α and β above are now two distinct ways of performing the reduction $\lambda.x_1 \mid x_2 \mid \bar\lambda.x_3 \Rightarrow x_1 \mid x_2 \mid x_3$.

Finally, the relationship between $\mathscr{A}_{\mathrm{d}}[\mathsf{CCS}/\!\equiv]$ and the reduction system in Figure 1 can be expressed by the 2-computad morphism that sends the rule $sync_\lambda$ (i.e. obtained as an instance of $\mathbb{C}_2^{1,1}$) in Figure 1 to the transaction $(\mathtt{act}_\lambda \otimes \mathtt{act}_{\bar\lambda}) * \mathtt{com}_\lambda$, showing that this is just a possible way of synchronizing two processes.

6 Conclusion

We have extended the zero-safe approach of [5] to the more general framework of tile and rewrite theories. The coreflection between the abstract and the specification view relates the two principal operational models based on LTS and reductions and provides a systematic general approach to the definition of transactions. In fact, the universal property of coreflections guarantees that the abstract system is the best possible representation (among rewrite theories) of the concurrent transactions of its corresponding tile theory. It is worth noting that the representation results for zero-safe nets presented in [5] now follow from the more general constructions defined here. Let us finally mention that the horizontal composition of D-computads has some analogies with conditional rewriting logic, but we leave the study of the precise correspondence between these two specification options for future work.

Acknowledgment. We warmfully thank Narciso Martí-Oliet for his many comments on a preliminary draft of the paper, which have been very helpful in improving the quality of our submission. We also thank the anonymous referees for their useful suggestions.

References

[1] Berry, G. and G. Boudol, *The chemical abstract machine*, Theoret. Comput. Sci. **96** (1992), pp. 217–248.

[2] Bruni, R., "Tile Logic for Synchronized Rewriting of Concurrent Systems," Ph.D. thesis, Computer Science Department, University of Pisa (1999).

[3] Bruni, R., C. Laneve and U. Montanari, *Orchestrating transactions in join calculus*, in *Proc. CONCUR 2002*, Lect. Notes in Comput. Sci. (2002), to appear.

[4] Bruni, R., J. Meseguer and U. Montanari, *Process and term tile logic*, Technical Report SRI-CSL-98-06, SRI International (1998).

[5] Bruni, R. and U. Montanari, *Zero-safe nets: Comparing the collective and individual token approaches*, Inform. and Comput. **156** (2000), pp. 46–89.

[6] Bruni, R. and U. Montanari, *Dynamic connectors for concurrency*, Theoret. Comput. Sci. **281**(1-2) (2002), pp. 131–176.

[7] Bruni, R., U. Montanari and F. Rossi, *An interactive semantics of logic programming*, Theory and Practice of Logic Programming. **1** (2001), pp. 647–690.

[8] Clavel, M. and J. Meseguer, *Reflection and strategies in rewriting logic*, in *Proc. WRLA'96*, Elect. Notes in Th. Comput. Sci. **4** (1996).

[9] Ehresmann, E., *Catégories structurèes: I–II*, Annales École Normal Superieur **80** (1963), pp. 349–426.

[10] Gadducci, F. and U. Montanari, *The tile model*, in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, 2000. pp. 133–166.

[11] Lawvere, F.W., *Functorial semantics of algebraic theories*, Proc. National Academy of Sciences **50** (1963), pp. 869–872.

[12] Meseguer, J., *Rewriting as a unified model of concurrency*, Technical Report SRI-CSL-90-02R, SRI International (1990).

[13] Meseguer, J., *Conditional rewriting logic as a unified model of concurrency*, Theoret. Comput. Sci. **96** (1992), pp. 73–155.

[14] Meseguer, J. and U. Montanari, *Petri nets are monoids*, Inform. and Comput. **88**(2) (1990), pp. 105–155.

[15] Meseguer, J. and U. Montanari, *Mapping tile logic into rewriting logic*, in *Proc. WADT'97*, Lect. Notes in Comput. Sci. **1376** (1998), pp. 62–91.

[16] Meseguer, J., P. Ölveczky and M.-O. Stehr, *Rewriting logic as a unifying framework for Petri nets*, in *Advances in Petri Nets: Unifying Petri Nets*, Lect. Notes in Comput. Sci. **2128**, Springer Verlag, 2001. pp. 250–303.

[17] Plotkin, G., *A structural approach to operational semantics*, Technical Report DAIMI FN-19, Aarhus University, Computer Science Department (1981).

[18] Street, R., *Higher categories, strings, cubes and simplex equations*, Applied Categorical Structures **3** (1995), pp. 29–77.

[19] Street, R., *Categorical structures*, in: M. Hazewinkel, editor, *Handbook of Algebra*, Elsevier Science, 1996. pp. 529–577.

A Monoidal Double Categories

A *double category* is an internal category in **Cat**, the category of categories (as objects) and functors (as arrows). More naïvely, they can be defined as below:

Definition A.1 A *double category* consists of a collection $a, b, c, \ldots$ of *objects*, a collection $h, g, f, \ldots$ of *horizontal arrows*, a collection $v, u, w, \ldots$ of *vertical arrows* and a collection $\alpha, \beta, \gamma, \ldots$ of *cells*.

Objects and horizontal arrows form the *horizontal 1-category* with identity a for each object a, and composition $\_;\_$. Similarly, objects and vertical arrows form the *vertical 1-category*, with identity a for each object a, and composition $\_;\_$.

Cells are assigned *horizontal source* and *target* (which are vertical arrows) and *vertical source* and *target* (which are horizontal arrows); furthermore sources and targets must be *compatible*, in the sense that they must form a square-shaped diagram like the one below, for which we use the notation $\alpha\colon h \xrightarrow[u]{v} g$.

$$
\begin{tikzcd}
a \arrow[r, "h"] \arrow[d, "v"'] \arrow[dr, phantom, "\alpha"] & b \arrow[d, "u"] \\
c \arrow[r, "g"'] & d
\end{tikzcd}
$$

Cells can be composed both horizontally ($*$) and vertically ($\cdot$) as follows: if $\alpha : h \xrightarrow[u]{v} g$, $\beta : f \xrightarrow[w]{u} k$, and $\gamma : g \xrightarrow[x]{z} p$, then $\alpha * \beta : h;f \xrightarrow[w]{v} g;k$, and $\alpha \cdot \gamma : h \xrightarrow[u;x]{v;z} p$. Moreover, given a fourth cell $\delta : k \xrightarrow[y]{x} q$, the following *exchange law* holds:

$$(\alpha \cdot \gamma) * (\beta \cdot \delta) = (\alpha * \beta) \cdot (\gamma * \delta)$$

Under these rules, cells form both a horizontal category and a vertical category, with identities $1_v : a \xrightarrow[v]{v} c$ and $1^h : h \xrightarrow[b]{a} h$, respectively. Given $1^h : h \xrightarrow[b]{a} h$ and $1^g : g \xrightarrow[c]{b} g$, the equation $1^h * 1^g = 1^{h;g}$ must hold (and similarly for vertical composition of horizontal identities).

Furthermore, horizontal and vertical identities of identities coincide, i.e., $1_a = 1^a$ and are denoted just by a (analogously, 1^h and 1_v are just denoted by h and v).

A *double functor* $\mathscr{G} : \mathbf{D}_1 \to \mathbf{D}_2$ is a 4-tuple of functions (one for objects, one for horizontal arrows, one for vertical arrows, and one for cells), preserving identities and compositions of all kinds. We let **DCat** be the category of double categories (as objects) and double functors (as arrows).

Definition A.2 A *monoidal double category* is a double category $\mathbf{D}$ equipped with a double functor $\otimes : \mathbf{D} \times \mathbf{D} \to \mathbf{D}$ (the *tensor product*) and with an object e (the *unit*) such that: (1) $(\otimes \times 1_{\mathbf{D}});\otimes = (1_{\mathbf{D}} \times \otimes);\otimes$, and (2) $(e \times 1_{\mathbf{D}});\otimes = (1_{\mathbf{D}} \times e);\otimes = 1_{\mathbf{D}}$.

A monoidal double category can be equivalently defined either as an internal category in **MCat**, the category of monoidal categories (as objects) and monoidal functors (as arrows), or as an internal monoid in **DCat** (see [15]).

A *monoidal double functor* is a double functor that (strictly) preserves tensor product and unit. The category of monoidal double categories (as objects) and monoidal double functors (as arrows) is called **DMCat**.

B Categories and Constructions

For the reader's convenience, in this Appendix we summarize in two tables the relevant categories and constructions between them that are discussed in the paper.

We recall that the notion of adjunction is an elegant categorical tool for establishing a correspondence between categories. There are several equivalent definitions of adjunction. Probably, the more "constructive" presentation consists of the scenario with two categories $\mathbf{A}$ and $\mathbf{B}$ and a functor $\mathscr{F} : \mathbf{A} \to \mathbf{B}$. Then, given an object $b \in \mathbf{B}$ we would like to find the object $a \in \mathbf{A}$ that "better approximates" b via $\mathscr{F}$, where:

- *approximation* means the existence of a morphism f from $\mathscr{F}(a)$ to b in $\mathbf{B}$;
- *best approximation* means that any other approximation $f' : \mathscr{F}(a') \to b$ via an object $a' \in \mathbf{A}$ can be expressed in terms of f and (the image of) a uniquely determined morphism from a' to a (the so-called *universal property*, as formalized below).

$$\begin{tikzcd}
\mathbf{A} & G_b & & \mathscr{F}(G_b) \arrow[r, "\varepsilon_b"] & b & \mathbf{B} \\
& a \arrow[u, dotted, "\widehat{g}"] & & \mathscr{F}(a) \arrow[u, "\mathscr{F}(\widehat{g})"] \arrow[ur, "g"'] & &
\end{tikzcd}$$

Figure B.1. The left adjoint $\mathscr{F}$.

When best approximations exist for all objects of $\mathbf{B}$, then they can be used to represent the relevant structure of $\mathbf{B}$ inside $\mathbf{A}$ itself (from the point of view of $\mathscr{F}$).

Definition B.1 Let $\mathbf{A}$ and $\mathbf{B}$ be two categories and let $\mathscr{F}\colon\mathbf{A} \to \mathbf{B}$ be a functor. We say that $\mathscr{F}$ is a *left adjoint* if for each object $b \in \mathbf{B}$ there exist an object $G_b \in \mathbf{A}$ and an arrow $\varepsilon_b\colon \mathscr{F}(G_b) \to b \in \mathbf{B}$, such that for any object $a \in \mathbf{A}$ and for any arrow $g\colon \mathscr{F}(a) \to b \in \mathbf{B}$, there is a unique arrow $\widehat{g}\colon a \to G_b \in \mathbf{A}$, such that $g = \mathscr{F}(\widehat{g});\varepsilon_b$ (see Figure B.1).

A consequence of this fact is the existence of a backward functor $\mathscr{G}\colon\mathbf{B} \to \mathbf{A}$ that maps each object b into its best approximation G_b. To see this point, note that given an arrow $h\colon b \to b' \in \mathbf{B}$, then the composite arrow $\varepsilon_b;h\colon \mathscr{F}(G_b) \to b'$ factorizes through $\varepsilon_{b'}$ via the image of a unique arrow $f\colon G_b \to G_{b'} \in \mathbf{A}$ (by definition of adjoint $f = \widehat{\varepsilon_b;h}$). Hence the functor $\mathscr{G}$ can be defined by letting $\mathscr{G}(h) = f$.

The functor $\mathscr{G}$ is called the *right adjoint* of $\mathscr{F}$, and we write $\mathscr{F} \dashv \mathscr{G}$. The collection $\varepsilon = \{\varepsilon_b\}_{b\in\mathbf{B}}$ is called the *counit* of the adjunction and defines a natural transformation from $\mathscr{G};\mathscr{F}$ to $1_{\mathbf{B}}$. Dually, it is possible to define a collection of "least upper" approximations $\eta = \{\eta_a\colon a \to \mathscr{G}(\mathscr{F}(a))\}_{a\in\mathbf{A}}$, where $\eta_a = \widehat{id_{\mathscr{F}(a)}}$, which defines a natural transformation from $1_{\mathbf{A}}$ to $\mathscr{F};\mathscr{G}$ (called *unit*).

An important property of adjunctions is the preservation of universal constructions: left adjoints preserve colimits, and right adjoints preserve limits. Since (co)limits can be seen as the categorical way of expressing operations, adjunctions guarantee to some extent a "compositional" interpretation for such operations.

The typical situation involves a category $\mathbf{B}$ that has more structure than $\mathbf{A}$, and a forgetful functor $\mathscr{G}$ that projects $\mathbf{B}$ to $\mathbf{A}$, deleting the extra structure. If $\mathscr{G}$ has left adjoint $\mathscr{F}$, then $\mathscr{F}$ defines the best way of adding that extra structure to $\mathbf{A}$.

Reflection and *coreflection* are two particularly kinds of adjunction, where, respectively, the counit and the unit define natural isomorphisms, yielding optimal approximations. When the unit is a natural isomorphism, then $\mathbf{A}$ can be seen just as subcategory of $\mathbf{B}$, with the left adjoint $\mathscr{F}$ being the inclusion functor. Thus, coreflection is the ideal situation from the semantics point of view. In fact, the typical situation involves a category of operational models $\mathbf{B}$ that contains a subcategory of abstract models $\mathbf{A}$, with $\mathscr{G}(b)$ being the abstraction of b. Then, the universal property of coreflections means that there is a natural isomorphisms between the observations of any concrete model b and of its abstract counterpart $\mathscr{G}(b)$, i.e. that b is the same as $\mathscr{G}(b)$ when observed from the abstract point of view defined by $\mathbf{A}$.

Category	Objects	Arrows
Cat	categories	functors
MCat	strict monoidal categories	strict monoidal functors
2MCat	strict monoidal 2-categories	strict monoidal 2-functors
DCat	double categories	double functors
DMCat	strict monoidal double categories	strict monoidal double functors
Petri	P/T Petri nets	Petri net morphisms
ZPetri	zero-safe nets	(disjoint) zero-safe net morphisms
HCatZPetri	zero-safe nets with enriched transitions $(*, \otimes, id)$	(disjoint) net homomorphisms
ZSN	zero-safe nets	refinement morphisms
2Comp, $\mathbf{2Comp}^{\mathrm{c}}$	2-computads	2-computad morphisms
DComp, $\mathbf{DComp}^{\mathrm{c}}$	D-computads	D-computad morphisms
ZComp, $\mathbf{ZComp}^{\mathrm{c}}$	D-computads	disjoint D-computad morphisms
HCatZComp, $\mathbf{HCatZComp}^{\mathrm{c}}$	D-computads with enriched tiles $(*, \otimes, id)$	disjoint D-computad homomorphisms
RComp, $\mathbf{RComp}^{\mathrm{c}}$	D-computads	computad refinement morphisms

Construction	Description	Original to this contribution?
$\mathbf{2Comp} \underset{\mathscr{U}_2}{\overset{\mathscr{F}_2}{\underset{\perp}{\rightleftarrows}}} \mathbf{2MCat}$	Adjunction	No (see [15])
$\mathbf{DComp} \underset{\mathscr{U}_{\mathrm{d}}}{\overset{\mathscr{F}_{\mathrm{d}}}{\underset{\perp}{\rightleftarrows}}} \mathbf{DMCat}$	Adjunction	No (see [15])
$\mathbf{ZPetri} \overset{\mathscr{Z}}{\underset{\perp}{\rightleftarrows}} \mathbf{HCatZPetri}$	Adjunction	No (see [5])
$\mathbf{Petri} \underset{\mathscr{A}_{\mathrm{z}}}{\underset{\perp}{\rightleftarrows}} \mathbf{ZSN}$	Coreflection	No (see [5])
$\mathbf{ZPetri} \hookrightarrow \mathbf{ZComp}^{\mathrm{c}} \hookrightarrow \mathbf{ZComp}$	Full and faithful inclusions	Yes (Proposition 4.2)
$\mathbf{HCatZPetri} \hookrightarrow \mathbf{HCatZComp}^{\mathrm{c}}$	Full and faithful inclusion	Yes (Proposition 4.3)
$\mathbf{ZComp} \overset{\mathscr{D}}{\underset{\perp}{\rightleftarrows}} \mathbf{HCatZComp}$	Adjunction	Yes (Proposition 4.4)
$\mathbf{ZComp}^{\mathrm{c}} \overset{\mathscr{D}^{\mathrm{c}}}{\underset{\perp}{\rightleftarrows}} \mathbf{HCatZComp}^{\mathrm{c}}$	Adjunction	Yes (Proposition 4.4)
$\mathbf{ZSN} \hookrightarrow \mathbf{RComp}^{\mathrm{c}} \hookrightarrow \mathbf{RComp}$	Full and faithful inclusions	Yes (Proposition 4.11)
$\mathbf{Petri} \hookrightarrow \mathbf{2Comp}^{\mathrm{c}} \hookrightarrow \mathbf{2Comp}$	Full and faithful inclusions	No (see [14])
$\mathbf{2Comp} \underset{\mathscr{A}_{\mathrm{d}}}{\overset{\mathscr{I}}{\underset{\perp}{\rightleftarrows}}} \mathbf{RComp}$	Coreflection	Yes (Theorem 4.10)
$\mathbf{2Comp}^{\mathrm{c}} \underset{\mathscr{A}_{\mathrm{d}}^{\mathrm{c}}}{\overset{\mathscr{I}^{\mathrm{c}}}{\underset{\perp}{\rightleftarrows}}} \mathbf{RComp}^{\mathrm{c}}$	Coreflection	Yes (Proposition 4.11)