Nonsequential Automata Semantics for a Concurrent Object-Based Language

MENEZES, P.B.; SERNADAS, A.; COSTA, J.F.

Preprint 21/95 October 1995

Nonsequential Automata Semantics for a Concurrent Object-Based Language *

P. Blauth Menezes†, A. Sernadas† and J. Félix Costa††

† Departamento de Matemática, Instituto Superior Técnico
Av. Rovisco Pais, 1096 Lisboa Codex, Portugal - {blauth, acs}@math.ist.utl.pt

†† Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa
Campo Grande, 1700 Lisboa, Portugal - fgc@di.fc.ul.pt

Abstract. Nonsequential automata constitute a categorial semantic domain based on labeled transition system with full concurrency, where synchronization and hiding are functorial and a class of morphisms stands for reification. It is, for our knowledge, the first model for concurrency which satisfies the diagonal compositionality requirement, i. e., reifications compose (vertical) and distribute over combinators (horizontal). To experiment with the proposed semantic domain, a semantics for a concurrent, object-based language is given. It is a simplified and revised version of the object-oriented specification language GNOME, introducing some special features inspired by the semantic domain such as reification and aggregation. The diagonal compositionality is an essential property to give semantics in this context.

1 Introduction

We construct a semantic domain with full concurrency for interacting systems which is, for our knowledge, the first model for concurrency satisfying the diagonal compositionality requirement, i.e., reifications compose (vertically), reflecting the stepwise description of systems, involving several levels of abstraction, and distributes through parallel composition (horizontally), meaning that the reification of a composite system is the composition of the reification of its parts, even in the presence of synchronization.

A nonsequential automaton (first introduced in [Menezes *et al* 95]) is a kind of automaton with monoidal structure on states and transitions, inspired by [Meseguer & Montanari 90]. Structured states are "bags" of local states like tokens in Petri nets (as in [Reisig 85]) and structured transitions specify a concurrency relationship between component transitions in the sense of [Bednarczyk 88] and [Mazurkiewicz 88]. The resulting category is bicomplete where the categorial product stands for parallel composition. Synchronization and hiding are functorial operations. A synchronization restricts a parallel composition according to some table of synchronizations (at label level). A view of an automaton is obtained through hiding of transitions introducing an internal nondeterminism. A hidden transition cannot be used for interaction. A reification maps transitions into transactions reflecting an implementation of an automaton on top of another. It is defined as an automaton morphism where the target object is enriched with all conceivable sequential and nonsequential computations. Computations are induced by an endofunctor and composition of reification morphisms is defined using Kleisli categories. Comparing with [Menezes *et al* 95], in this paper we revise the reification morphisms, introduce the synchronization and hiding for reifications (extending the approach for automata) and generalize the categorical definition for table of synchronizations.

In [Menezes & Costa 95] and [Menezes & Costa 95b] we show that nonsequential automata are more concrete then Petri nets (in fact, categories of Petri nets are isomorphic to subcategories of nonsequential automata) extending the approach in [Sassone *et al* 93], where a formal framework for classification of models for concurrency is set.

To experiment with the proposed semantic domain, a semantics for a concurrent object-based language is given. The language named Nautilus is based on the object-oriented language GNOME [Sernadas & Ramos 94] which is a simplified and revised version of OBLOG [SernadasC *et al* 92], [SernadasC *et al* 92b], [SernadasC *et al* 91]. Some features inspired by the semantic domain (and not present on GNOME) such as reification and aggregation are introduced. A reification implements an object over sequential or concurrent computations of another. The main difference between interaction and aggregation is that, in the former, the relationship between objects is defined within each object while in the later, the relationship is defined externally to the component objects. Also, the state-dependent calling of GNOME is extended for interaction, aggregation and reification in Nautilus. For simplicity and in order to keep the paper short, we do not deal with some feature of GNOME such as classes of objects and inheritance. The diagonal compositionality requirement is essential to give semantics for Nautilus.

* This work was partially supported by: UFRGS - Universidade Federal do Rio Grande do Sul and CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico in Brazil; CEC under ESPRIT-III BRA WG 6071 IS-CORE, WG 6112 COMPASS, HCM Scientific Network MEDICIS, JNICT (PBIC/C/TIT/1227/92) in Portugal.

2 Nonsequential Automata

A nonsequential automaton is a reflexive graph labeled on arcs such that nodes, arcs and labels are elements of commutative monoids. A reflexive graph represents the *shape* of an automaton where nodes and arcs stand for states and transitions, respectively, with identity arcs interpreted as *idle* transitions. A structured transition specify a concurrency relation between component transitions. Comparing with asynchronous transition systems (first introduced in [Bednarczyk 88]), the independence relation of a nonsequential automaton is explicit in the graphical representation. A structured state can be viewed as a "bag" of local states where each local state can be viewed as a resource to be consumed or produced, like a token in Petri nets.

Nonsequential automata and its morphisms constitute a category which is complete and cocomplete with products isomorphic to coproducts. A product (or coproduct) can be viewed as a parallel composition. In what follows $\mathcal{CMon}$ denotes the category of commutative monoids and suppose that k is in $\{0, 1\}$. Also, for the proof or details omitted, see [Menezes *et al* 95] and [Menezes & Costa 95b].

2.1 Nonsequential Automaton

Definition 2.1 Nonsequential Automaton. A nonsequential automaton $N = \langle V, T, \partial_0, \partial_1, \iota, L, lab\rangle$ is such that $T = \langle T, \|, \tau\rangle$, $V = \langle V, \oplus, e\rangle$, $L = \langle L, \|, \tau\rangle$ are $\mathcal{CMon}$-objects of transitions, states and labels respectively, $\partial_0, \partial_1: T \to V$ are $\mathcal{CMon}$-morphisms called source and target respectively, $\iota: V \to T$ is a $\mathcal{CMon}$-morphism such that $\partial_k \circ \iota = id_V$ and $lab: T \to L$ is a $\mathcal{CMon}$-morphism such that $lab(t) = \tau$ whenever there is v in V where $\iota(v) = t$. ❑

We may refer to a nonsequential automaton $N = \langle V, T, \partial_0, \partial_1, \iota, L, lab\rangle$ by $N = \langle G, L, lab\rangle$ where $G = \langle V, T, \partial_0, \partial_1, \iota\rangle$ is a reflexive graph internal to $\mathcal{CMon}$ (i.e., V, T are $\mathcal{CMon}$-objects and $\partial_0, \partial_1, \iota$ are $\mathcal{CMon}$-morphisms). In an automaton, a transition labeled by τ represents a hidden transition (and therefore, can not be triggered from the outside). Note that, all idle transitions are hidden. The labeling procedure is not extensional in the sense that two distinct transitions with the same label may have the same source and target states (as we will se later, it is essential to give semantics for an object reification in Nautilus). In this paper we are not concerned with initial states.

A transition t such that $\partial_0(t) = X$, $\partial_1(t) = Y$ is denoted by $t: X \to Y$. Since a state is an element of a monoid, it may be denoted as a formal sum $n_1A_1 \oplus \ldots \oplus n_mA_m$, with the order of the terms being immaterial, where A_i is in V and n_i indicate the multiplicity of the corresponding (local) state, for $i = 1\ldots m$. The denotation of a transition is analogous. We also refer to a structured transition as the *parallel composition* of component transitions. When no confusion is possible, a structured transition $x \| \tau: X \oplus A \to Y \oplus A$ where $t: X \to Y$ and $\iota_A: A \to A$ are labeled by x and τ, respectively, is denoted by $x: X \oplus A \to Y \oplus A$. For simplicity, in graphical representation, we omit the identity transitions. States and labeled transitions are graphically represented as circles and boxes, respectively.

Example 2.2 Let $\langle\{A, B, X, Y\}^{\oplus}, \{t_1, t_2, t_3, A, B, C, X, Y\}^{\otimes}, \partial_0, \partial_1, \iota, \{x, y\}^{\otimes}, lab\rangle$ be a nonsequential automaton with ∂_0, ∂_1 determined by the local arcs $t_1: 2A \to B$, $t_2: X \to Y$, $t_3: Y \to X$ and lab determined by $t_1 \mapsto x$, $t_2 \mapsto x$, $t_3 \mapsto y$. The distributed and infinite schema in Figure 1 (left) represents the automaton. Since in this framework we do not deal with initial states, the graphical representation makes explicit all possible states that can be reached by all possible independent combination of component transitions. For instance, if we consider the initial state $A \oplus 2X$, only the corresponding part of the schema of the automata in the figure has to be considered. In Figure 1 (right), we illustrate a labeled Petri net which simulates the behavior of the automaton. Comparing both schema, we realize that, while the concurrence and possible reachable markings are implicit in a net, they are explicit in an automaton. Categories of Petri nets and categories of nonsequential automata can be unified through adjunctions. For details, see [Menezes & Costa 95] and [Menezes & Costa 95b]. ❑

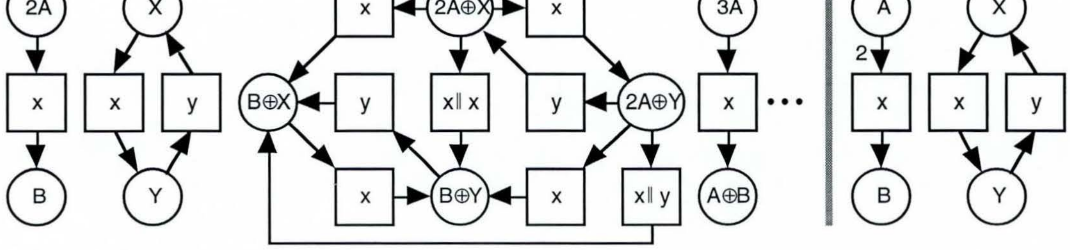


Figure 1. A nonsequential automaton (left) and the corresponding labeled Petri net (right)

Definition 2.3 Nonsequential Automaton Morphism. A nonsequential automaton morphism $h: N_1 \to N_2$ where $N_1 = \langle V_1, T_1, \partial_{01}, \partial_{11}, \iota_1, L_1, \mathsf{lab}_1\rangle$ and $N_2 = \langle V_2, T_2, \partial_{02}, \partial_{12}, \iota_2, L_2, \mathsf{lab}_2\rangle$ is a triple $h = \langle h_V, h_T, h_L\rangle$ such that $h_V: V_1 \to V_2$, $h_T: T_1 \to T_2$, $h_L: L_1 \to L_2$ are *CMon*-morphisms, $h_V \circ \partial_{k1} = \partial_{k2} \circ h_T$, $h_T \circ \iota_1 = \iota_2 \circ h_V$ and $h_L \circ \mathsf{lab}_1 = \mathsf{lab}_2 \circ h_T$. ❑

Nonsequential automata and their morphisms constitute the category $\mathcal{NAut}$.

Proposition 2.4 The category $\mathcal{NAut}$ is complete and cocomplete with products isomorphic to coproducts. ❑

A categorical product (or coproduct) of two automata $N_1 = \langle V_1, T_1, \partial_{01}, \partial_{11}, \iota_1, L_1, \mathsf{lab}_1\rangle$, $N_2 = \langle V_2, T_2, \partial_{02}, \partial_{12}, \iota_2, L_2, \mathsf{lab}_2\rangle$ is $N_1 \times_{\mathcal{NAut}} N_2 = \langle V_1 \times_{\mathcal{CMon}} V_2, T_1 \times_{\mathcal{CMon}} T_2, \partial_{01} \times \partial_{02}, \partial_{11} \times \partial_{12}, \iota_1 \times \iota_2, L_1 \times_{\mathcal{CMon}} L_2, \mathsf{lab}_1 \times \mathsf{lab}_2\rangle$ where $\partial_{k1} \times \partial_{k2}$, $\iota_1 \times \iota_2$ and $\mathsf{lab}_1 \times \mathsf{lab}_2$ are uniquely induced by the product construction.

2.2 Synchronization and Hiding

Synchronization and hiding of transitions are functorial operations defined using fibration and cofibration techniques inspired by [Winskel 87]. Both functors are induced by morphisms at the label level.

The synchronization operation restricts the product "erasing" all those transitions which do not reflect some given table of synchronizations (suppose that i is in I):

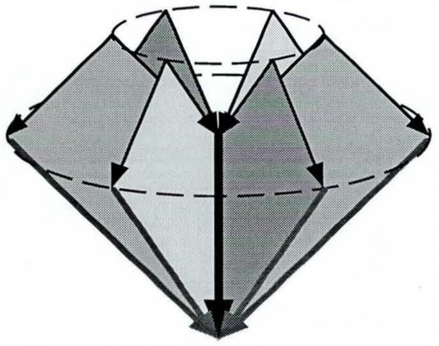
a) let $\{N_i\}$ be a set of nonsequential automata with $\{L_i\}$ as the corresponding *CMon*-objects of labels, *Table* be a commutative monoid, called table of synchronizations, determined by the tuples of labels to be synchronized and sync: $\mathit{Table} \to \times L_i$ be the synchronization morphism which maps the table into the labels of a given automaton;
b) let $u: \mathcal{NAut} \to \mathcal{CMon}$ be the obvious forgetful functor taking each automaton into its commutative monoid of labels. The functor u is a fibration and the fibers $u^{-1}\,\mathit{Table}$, $u^{-1} \times L_i$ are subcategories of $\mathcal{NAut}$;
c) the fibration u and the morphism sync induce a functor $\mathit{sync}: u^{-1} \times L_i \to u^{-1}\,\mathit{Table}$. The functor sync applied to $\times N_i$ provides the automaton reflecting the desired synchronizations.

Traditionally, in concurrency theory, the concealment of transitions is achieved by resorting to labeling and using the special label τ (cf. [Winskel 87]). Such hidden transitions cannot be used for synchronization since they are *encapsulated.* The steps for hiding are the following:

a) let N be a nonsequential automaton with L_1 as its commutative monoid of labels, let hide: $L_1 \to L_2$ be a morphism taking the transitions to be hidden into τ;
b) let $u: \mathcal{NAut} \to \mathcal{CMon}$ be the same forgetful functor used for synchronization purpose. The functor u is a cofibration (and therefore, a bifibration) and the fibers $u^{-1} L_1$, $u^{-1} L_2$ are subcategories of $\mathcal{NAut}$;
c) the cofibration u and the morphism hide induce a functor $\mathit{hide}: u^{-1} L_1 \to u^{-1} L_2$. The functor hide applied to N provides the automaton reflecting the desired encapsulation.

Table of Synchronizations. In what follows, we show a categorial way to construct tables of synchronizations for calling and sharing and the corresponding synchronization morphism. The following construction generalizes the approaches in [Menezes *et al* 95] and [Menezes & Costa 93] for more than two systems.

The table of synchronizations for interaction is given by a colimit whose resulting diagram has a shape illustrated in the Figure 2 (left) where the central arrow has as source an object named channel and as target the table of synchronizations. We say that a shares x if and only if a calls x and x calls a. In what follows, we denote by a | x a pair of synchronized transitions.



$$\begin{tikzcd} & L_i' \arrow[dl, "\mathsf{inc}_i"'] \arrow[dr, "\mathsf{call}_i"] & \\ L_i \arrow[dr, "\mathsf{p}_i"'] \arrow[rr, phantom, "\textbf{colimit}"] & & \mathit{Channel} \arrow[dl, "\mathsf{q}"] \\ & \mathit{Table} & \end{tikzcd}$$

CMon

Figure 2. Table of synchronizations

Definition 2.5 Table of Synchronizations. Let $\{N_i\}$ be a set of nonsequential automata with $\{L_i\}$ as the corresponding commutative monoids of labels, *Channel* be the least commutative monoid determined by all tuples of transitions to be synchronized, L_i' be the least commutative submonoid of L_i containing all transitions of N_i which call other transitions, call_i: $L_i' \rightarrow Channel$ be the morphisms such that, for a in L_i', if a calls $x_1,\ldots, x_n$ then $\text{call}_i(a) = a \mid x_1 \mid \ldots \mid x_n$ and D be the diagram represented in the Figure 2 (right) where inc_i: $L_i' \rightarrow L_i$ are inclusion morphisms. The table of synchronizations *Table* is given by the colimit of D. ❑

Example 2.6 Consider the free commutative monoids of labels $L_1 = \{a, b, c\}^{\parallel}$, $L_2 = \{x, y\}^{\parallel}$. Suppose that a calls x, b calls y and y calls b (i.e., b shares y). Then, $Channel = \{a \mid x, b \mid y\}^{\parallel}$, $L_1' = \{a, b\}^{\parallel}$, $L_2' = \{y\}^{\parallel}$ and $Table = \{c, x, a \mid x, b \mid y\}^{\parallel}$. ❑

Let D be a diagram whose colimit determines *Table* and p_i: $L_i \rightarrow Table$. Then there are retractions for p_i denoted by p_i^R such that, for every b in *Table*, if there is a in L_i such that p(a) = b then $p_i^R(b) = a$ else $p_i^R(b) = \tau$.

Definition 2.7 Synchronization Morphism. The synchronization morphism sync: $Table \rightarrow \times L_i$ is uniquely induced by the product construction as illustrated in the Figure 3. ❑

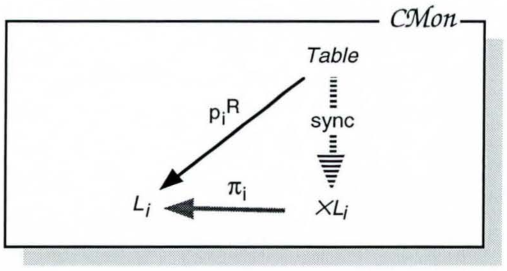

$$\begin{tikzcd} & Table \arrow[dl, "p_i^R"'] \arrow[d, dashed, "\text{sync}"] \\ L_i & \times L_i \arrow[l, "\pi_i"] \end{tikzcd}$$

Figure 3. Synchronization morphism

Synchronization Functor. First we show that the forgetful functor which takes each nonsequential automaton into its commutative monoids of labels is a fibration and then we introduce the synchronization functor.

Proposition 2.8 The forgetful functor u: *NAut* $\rightarrow$ *CMon* that takes each nonsequential automaton onto its underlying commutative monoid of labels is a fibration. ❑

Proof: Let *RGr(CMon)* be the category of reflexive graphs internal to *CMon* and let *id*: *RGr(CMon)* $\rightarrow$ *RGr(CMon)*, *emb*: *CMon* $\rightarrow$ *RGr(CMon)* be functors. Then, *NAut* can be defined as the comma category $id{\downarrow}emb$. Let f: $L_1 \rightarrow L_2$ be a *CMon*-morphism and $N_2 = \langle G_2, L_2, \text{lab}_2\rangle$ be a nonsequential automaton where $G_2 = \langle V_2, T_2, \partial 0_2, \partial 1_2, \iota_2\rangle$ is a *RGr(CMon)*-object. Let the object G_1 together with lab_1: $G_1 \rightarrow emb L_1$ and u_G: $G_1 \rightarrow G_2$ be the pullback of f: $emb L_1 \rightarrow emb L_2$ and lab_2: $G_2 \rightarrow emb L_2$. Define $N_1 = \langle G_1, L_1, \text{lab}_1\rangle$ which is an automaton by construction. Then $u = \langle u_G, f\rangle$: $N_1 \rightarrow N_2$ is cartesian with respect to f and N_2. ❑

Definition 2.9 Functor sync. Consider the fibration u: *NAut* $\rightarrow$ *CMon*, the automata $N_i = \langle V_i, T_i, \partial 0_i, \partial 1_i, \iota_i, L_i, \text{lab}_i\rangle$ and the synchronization morphism sync: $Table \rightarrow \times L_i$. The synchronization of N_i represented by $\parallel_{\text{sync}} N_i$ is given by the functor *sync*: $u^{-1}(\times L_2) \rightarrow u^{-1}(Table)$ induced by u and sync applied to $\times N_i$, i.e., $\parallel_{\text{sync}} N_i = sync(\times N_i)$. ❑

Example 2.10 Consider the nonsequential automata Consumer and Producer (with free monoids) determined by the labeled transitions prod: A $\rightarrow$ B, send: B $\rightarrow$ A for the Producer and rec: X $\rightarrow$ Y, cons: Y $\rightarrow$ X for the Consumer. Suppose that we want a joint behavior sharing the transitions send and rec (a communication without buffer such as in CSP [Hoare 85] or CCS [Milner 89]). Then, $Channel = \{\text{send} \mid \text{rec}\}^{\parallel}$ and $Table = \{\text{prod}, \text{cons}, \text{send} \mid \text{rec}\}^{\parallel}$. The resulting automaton is illustrated in the Figure 4. Note that the transitions send, rec are

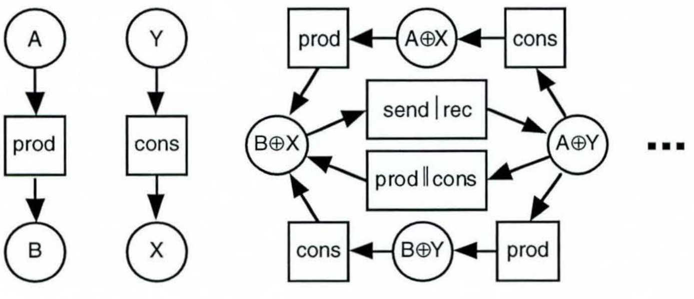


Figure 4. Synchronized automaton

erased and send | rec is included. ❑

Hiding. For encapsulation purposes, we work with *hiding morphisms*. A hiding morphism is an injective morphism except for those labels we want to hide (i.e., to relabel by τ). In what follows e, denotes a zero object in *CMon* (any monoid with only one element) and ! denotes the unique morphism with e as source or target.

Definition 2.11 Hiding Morphism. Let L_1 be the commutative monoid of labels of the automata to be encapsulated, L be least commutative submonoid of L_1 containing all labels to be hidden and inc: $L \to L_1$ be the inclusion. Let L_2 together with hide: $L_1 \to L_2$ and q: $e \to L_2$ be the pushout of !: $L \to e$ and inc: $L \to L_1$. Then, the hiding morphism is the morphism hide. ❑

Proposition 2.12 The forgetful functor *u: NAut → CMon* that maps each automaton onto its underlying commutative monoid of labels is a cofibration.

Proof: Let f: $L_1 \to L_2$ be a *CMon*-morphism and $N_1 = \langle V_1, T_1, \partial_{0_1}, \partial_{1_1}, \iota_1, L_1, lab_1\rangle$ be an automaton. Define $N_2 = \langle V_1, T_1, \partial_{0_1}, \partial_{1_1}, \iota_1, L_2, f \circ lab_1\rangle$. Then $u = \langle id_{V_1}, id_{T_1}, f\rangle$: $N_1 \to N_2$ is cocartesian with respect to f and N_1. ❑

Definition 2.13 Functor hide. Consider the fibration *u: NAut → CMon*, the nonsequential automata $N = \langle V, T, \partial_0, \partial_1, \iota, L_1, lab\rangle$ and the hiding morphism hide: $L_1 \to L_2$. The hiding of N satisfying hide denoted by N\hide is given by the functor *hide*: $u^{-1}L_1 \to u^{-1}L_2$ induced by u and hide applied to N, i.e., N\hide = *hide*N. ❑

Example 2.14 Consider the resulting automata of the previous example. Suppose that we want to hide the synchronized transition send | rec. Then, the hiding morphism is induced by send | rec $\mapsto \tau$ and the encapsulated automaton is as illustrated in the Figure 4 except that the transition send | rec has its label replaced by τ. ❑

2.3 Reification

A reification is defined as a special automaton morphism where the target object is closed under computations, i.e., the target (more concrete) automaton is enriched with all the conceivable sequential and nonsequential computations that can be split into permutations of original transitions, respecting source and target states.

The category of categories internal to *CMon* is denoted by *Cat(CMon)*. We introduce the category *LCat(CMon)* which can be viewed as a generalization of labeling on *Cat(CMon)*. There is a forgetful functor from *LCat(CMon)* into *NAut*. This functor has a left adjoint which freely generates a nonsequential automaton into a labeled internal category. The composition of both functors from *NAut* into *LCat(CMon)* leads to an endofunctor, called transitive closure. The composition of reifications of nonsequential automata is defined using Kleisli categories (see [Asperti & Longo 91]). In fact, the adjunction above induces a monad which defines a Kleisli category. Then we show that reification distributes over the parallel composition and therefore, the resulting category of automata and reifications *satisfies the diagonal compositionality*.

Definition 2.15 Category LCat(CMon). Consider the category *Cat(CMon)*. The category *LCat(CMon)* is the comma category $id_{Cat(CMon)} \downarrow id_{Cat(CMon)}$ where $id_{Cat(CMon)}$ is the identity functor in *Cat(CMon)*. ❑

Therefore, a *LCat(CMon)*-object is triple $\mathcal{N} = \langle \mathcal{G}, \mathcal{L}, lab\rangle$ where $\mathcal{G}$, $\mathcal{L}$ are *Cat(CMon)*-objects and *lab* is a *Cat(CMon)*-morphism.

Proposition 2.16 The category *LCat(CMon)* has all (small) products and coproducts. Moreover, products and coproducts are isomorphic.

Definition 2.17 Functor cn. Let $\mathcal{N} = \langle \mathcal{G}, \mathcal{L}, lab\rangle$ be a *LCat(CMon)*-object and $h = \langle h_{\mathcal{G}}, h_{\mathcal{L}}\rangle$: $\mathcal{N}_1 \to \mathcal{N}_2$ be a *LCat(CMon)*-morphism. The functor *cn: LCat(CMon) → NAut* is such that:

a) the *Cat(CMon)*-object $\mathcal{G} = \langle V, T, \partial_0, \partial_1, \iota, ;\rangle$ is taken into the *RGraph(CMon)*-object $G = \langle V, T', \partial_0', \partial_1', \iota'\rangle$, where T' is T subject to the equational rule below and $\partial_0', \partial_1', \iota'$ are induced by $\partial_0, \partial_1, \iota$ considering the monoid T'; the *Cat(CMon)*-object $\mathcal{L} = \langle V, L, \partial_0, \partial_1, \iota, ;\rangle$ is taken into the *CMon*-object L', where L' is L subject to the same equational rule; the *LCat(CMon)*-object $\mathcal{N} = \langle \mathcal{G}, \mathcal{L}, lab\rangle$ is taken into the *NAut*-object $N = \langle G, L', lab\rangle$ where lab is the *RGraph(CMon)*-morphism canonically induced by the *Cat(CMon)*-morphism *lab*;

$$\frac{t: A \to B \in T \quad u: B \to C \in T \quad t': A' \to B' \in T \quad u': B' \to C' \in T}{(t;u) \| (t';u')' = (t \| t');(u \| u') \text{ in } T'}$$

b) the *LCat(CMon)*-morphism $h = \langle h_{\mathcal{G}}, h_{\mathcal{L}}\rangle$: $\mathcal{N}_1 \to \mathcal{N}_2$ with $h_{\mathcal{G}} = \langle h_{N_V}, h_{N_T}\rangle$, $h_{\mathcal{L}} = \langle h_{L_V}, h_{L_T}\rangle$ is taken into the *NAut*-morphism $h = \langle h_{N_V}, h_{N_{T'}}, h_{L_{T'}}\rangle$: $N_1 \to N_2$ where $h_{N_{T'}}$ and $h_{L_{T'}}$ are the monoid morphisms induced by h_{N_T} and h_{L_T}, respectively. ❑

The functor $\mathcal{cn}$ has a requirement about concurrency which is $(t;u)\|(t';u') = (t\|t');(u\|u')$. That is, the computation determined by two independent composed transitions $t;u$ and $t';u'$ is equivalent to the computation whose steps are the independent transitions $t\|t'$ and $u\|u'$.

Definition 2.18 Functor nc. Let $A = \langle G, L, lab\rangle$ be a $\mathcal{NAut}$-object and $h = \langle h_G, h_L\rangle: A_1 \rightarrow A_2$ be a $\mathcal{NAut}$-morphism. The functor $nc: \mathcal{NAut} \rightarrow \mathcal{LCat(CMon)}$ is such that:

a) the $\mathcal{RGraph(CMon)}$-object $G = \langle V, T, \partial_0, \partial_1, \iota\rangle$ with $V = \langle V, \oplus, e\rangle$, $T = \langle T, \|, \tau\rangle$ is taken into the $\mathcal{Cat(CMon)}$-object $\mathcal{G} = \langle V, T^c, \partial_0^c, \partial_1^c, \iota, ;\rangle$ with $T^c = \langle T^c, \otimes, \tau\rangle$, ∂_0^c, ∂_1^c, $\_;\_: T^c \times T^c \rightarrow T^c$ inductively defined as follows:

$$\frac{t: A \rightarrow B \in T}{t: A \rightarrow B \in T^c} \qquad \frac{t: A \rightarrow B \in T^c \quad u: C \rightarrow D \in T^c}{t \otimes u: A \oplus C \rightarrow B \oplus D \in T^c} \qquad \frac{t: A \rightarrow B \in T^c \quad u: B \rightarrow C \in T^c}{t;u: A \rightarrow C \in T^c}$$

subject to the following equational rules:

$$\frac{t \in T \quad u \in T}{t \otimes u = t\|u} \qquad \frac{t \in T^c \quad u \in T^c}{t \otimes u = u \otimes t} \qquad \frac{t \in T^c}{t \otimes \tau = t} \qquad \frac{t \in T^c \quad u \in T^c \quad v \in T^c}{t \otimes (u \otimes v) = (t \otimes u) \otimes v}$$

$$\frac{t: A \rightarrow B \in T^c}{\iota_A;t = t \;\&\; t;\iota_B = t} \qquad \frac{t: A \rightarrow B \in T^c \quad u: B \rightarrow C \in T^c \quad v: C \rightarrow D \in T^c}{t;(u;v) = (t;u);v}$$

the $\mathcal{CMon}$-object L is taken into the $\mathcal{Cat(CMon)}$-object $\mathcal{L} = \langle 1, L^c, !, !, !, ;\rangle$ as above; the $\mathcal{NAut}$-object $A = \langle G, L, lab\rangle$ is taken into the $\mathcal{LCat(CMon)}$-object $\mathcal{A} = \langle \mathcal{G}, \mathcal{L}, \mathcal{lab}\rangle$ where $\mathcal{lab}$ is the morphism induced by lab;

d) the $\mathcal{NAut}$-morphism $h = \langle h_V, h_T, h_L\rangle: A_1 \rightarrow A_2$ is taken into the $\mathcal{Cat(CMon)}$-morphism $\mathcal{h} = \langle \mathcal{h}_{\mathcal{G}}, \mathcal{h}_{\mathcal{L}}\rangle: \mathcal{A}_1 \rightarrow \mathcal{A}_2$ where $\mathcal{h}_{\mathcal{G}} = \langle h_V, h_{T^c}\rangle$, $\mathcal{h}_{\mathcal{L}} = \langle !, h_{L^c}\rangle$ and h_{T^c}, h_{L^c} are the monoid morphisms generated by the monoid morphisms h_T and h_{T_L}, respectively. ❑

Proposition 2.19 The functor $nc: \mathcal{NAut} \rightarrow \mathcal{LCat(CMon)}$ is left adjoint to $cn: \mathcal{LCat(CMon)} \rightarrow \mathcal{NAut}$.

Definition 2.20 Transitive Closure Functor. The transitive closure functor is $tc = cn \circ nc: \mathcal{NAut} \rightarrow \mathcal{NAut}$. ❑

Example 2.21 Consider the nonsequential automaton with free monoids on states and transitions, determined by the transitions $a: A \rightarrow B$ and $b: B \rightarrow C$. Then, for instance, $a;2b: A\oplus B \rightarrow B\oplus C$ is a transition in the transitive closure. Note that, $a;2b$ represents a class of transitions. In fact, from the equations we can infer that $a;2b = a;(b\|b) = (\tau[B]\|a);(b\|b) = (\tau[B];b)\|(a;b) = b\|(a;b) = (b;\tau[C])\|(\tau[A];(a;b)) = (b\|\tau[A]);(\tau[C]\|(a;b)) = b;a;b = \ldots$ ❑

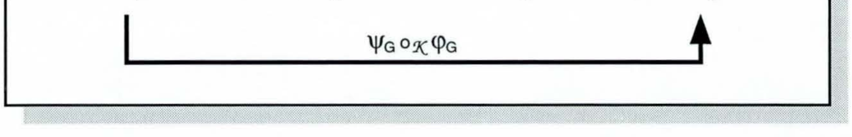
Let $\langle nc, cn, \eta, \varepsilon\rangle$ be the adjunction from $\mathcal{NAut}$ into $\mathcal{LCat(CMon)}$ as above. Then, $T = \langle tc, \eta, \mu\rangle$ is a monad on $\mathcal{NAut}$ such that $\mu = cn\varepsilon\, nc: tc^2 \rightarrow tc$ where $cn: cn \rightarrow cn$ and $nc: nc \rightarrow nc$ are the identity natural transformations and $cn\varepsilon\, nc$ is the horizontal composition of natural transformations. For some given automaton N, tcN is N enriched with its computations, $\eta_N: N \rightarrow tcN$ includes N into its computations and $\mu_N: tc^2N \rightarrow tcN$ flattens computations of computations of N into the computations of N.

In previous works we define a reification morphism φ from A into the computations of B as an $\mathcal{NAut}$-morphism $\varphi: A \rightarrow tcB$ and the composition of reifications as in Kleisli categories (each monad defines a Kleisli category). In this work, we modify the definition, since reifications should to not preserve labeling (and thus, they are not $\mathcal{NAut}$-morphisms). However, as we show below, each reification induces a $\mathcal{NAut}$-morphism. Therefore, we may define a category whose morphisms are $\mathcal{NAut}$-morphisms induced by reifications. Both categories are isomorphic.

Definition 2.22 Reification. Let $T = \langle tc, \eta, \mu\rangle$ where $\eta = \langle \eta_G, \eta_L\rangle$, $\mu = \langle \mu_G, \mu_L\rangle$ be the monad induced by the adjunction $\langle nc, cn, \eta, \varepsilon\rangle: \mathcal{NAut} \rightarrow \mathcal{LCat(CMon)}$. The category of nonsequential automata and reifications, denoted by $\mathcal{ReifNAut}$, is such that (suppose the $\mathcal{NAut}$-objects $N_k = \langle G_k, L_k, lab_k\rangle$, for k in {1, 2, 3}):

a) $\mathcal{ReifNAut}$-objects are the $\mathcal{NAut}$-objects;

b) $\varphi = \varphi_G: N_1 \rightarrow N_2$ is a $\mathcal{ReifNAut}$-morphism where $\varphi_G: G_1 \rightarrow tcG_2$ is a $\mathcal{RGr(CMon)}$-morphism and for each



$$\begin{tikzcd} G_1 \arrow[r, "\varphi_G"] \arrow[rrr, bend right, "\psi_G \circ_{\mathcal{K}} \varphi_G"'] & tcG_2 \arrow[r, "tc\,\psi_G"] & tc^2G_3 \arrow[r, "\mu_G"] & tcG_3 \end{tikzcd}$$

Figure 5. Composition of reifications is the composition in the Kleisli category forgetting about the labeling

$\mathcal{NA}ut$-object N, $\varphi = \eta_G$: N $\to$ N is the identity morphism of N in $\mathcal{R}eif\mathcal{NA}ut$;

c) let φ: $N_1 \to N_2$, ψ: $N_2 \to N_3$ be $\mathcal{R}eif\mathcal{NA}ut$-morphisms. The composition $\psi \circ \varphi$ is a morphism $\psi_G \circ_{\mathcal{K}} \varphi_G$: $N_1 \to N_3$ where $\psi_G \circ_{\mathcal{K}} \varphi_G$ is as illustrated in the Figure 5. ❑

In what follows, a nonsequential automaton ⟨G, *L*, lab⟩ is viewed as a morphism lab: G $\to$ *emb* *L* as in the Proposition 2.8. For simplicity, in diagrams, lab: G $\to$ *emb* *L* is abbreviated just by lab: G $\to$ *L*.

Definition 2.23 Reification with Induced Labeling. Let T = ⟨*tc*, η, μ⟩ where $\eta = \langle \eta_G, \eta_L \rangle$, $\mu = \langle \mu_G, \mu_L \rangle$ be the monad induced by the adjunction ⟨*nc*, *cn*, η, ε⟩. The category of nonsequential automata and reifications with induced labeling, denoted by $\mathcal{R}eif\mathcal{NA}ut_{\mathcal{L}}$, is such that (suppose the $\mathcal{NA}ut$-objects $N_k = \langle G_k, L_k, lab_k \rangle$, for k in {1, 2, 3}):

a) $\mathcal{R}eif\mathcal{NA}ut_{\mathcal{L}}$-objects are the $\mathcal{NA}ut$-objects;

b) let φ_G: $G_1 \to tcG_2$ be a $\mathcal{RG}r(\mathcal{CM}on)$-morphism. Then $\varphi = \langle \varphi_G, \varphi_L \rangle$: $N_1 \to N_2$ is a $\mathcal{R}eif\mathcal{NA}ut_{\mathcal{L}}$-morphism where φ_L is given by the pushout illustrated in the Figure 6 (left). For each $\mathcal{NA}ut$-object N, $\varphi = \langle \eta_G: G \to tcG, \varphi_L: L \to L_\eta \rangle$: N $\to$ N is the identity morphism of N in $\mathcal{R}eif\mathcal{NA}ut_{\mathcal{L}}$ where φ_L is as above;

c) let φ: $N_1 \to N_2$, ψ: $N_2 \to N_3$ be $\mathcal{R}eif\mathcal{NA}ut_{\mathcal{L}}$-morphisms. The composition $\psi \circ \varphi$ is a morphism $\langle \psi_G \circ_{\mathcal{K}} \varphi_G, \psi_L \circ_{\mathcal{L}} \varphi_L \rangle$: $N_1 \to N_3$ where $\psi_G \circ_{\mathcal{K}} \varphi_G$ e $\psi_L \circ_{\mathcal{L}} \varphi_L$ is as illustrated in the Figure 6 (right). ❑

$\mathcal{RG}r(\mathcal{CM}on)$

$$\begin{tikzcd}
G_1 \arrow[r, "lab_1"] \arrow[d, "\varphi_G"'] & L_1 \arrow[dr, "\varphi_L"] & \\
tcG_2 \arrow[r, "tc\,lab_2"] \arrow[rr, bend right, "lab_{2,\varphi}"'] & tc\,L_2 \arrow[r] \arrow[ur, phantom, "p.o."] & L_{2,\varphi}
\end{tikzcd}$$

$$\begin{tikzcd}
G_1 \arrow[r, "lab_1"] \arrow[d, "\varphi_G"] \arrow[dddd, bend right=60, "\psi_G \circ_{\mathcal{K}} \varphi_G"'] & L_1 \arrow[ddddr, "\psi_L \circ_{\mathcal{L}} \varphi_L"] & \\
tcG_2 \arrow[d, "\psi_G"] & & \\
tc^2G_3 \arrow[d, "\mu_G"] & & \\
 & & \\
tcG_3 \arrow[r, "tc\,lab_3"] \arrow[rr, bend right, "lab_{3,\psi\circ\varphi}"'] & tc\,L_3 \arrow[r] \arrow[ur, phantom, "p.o."] & L_{3,\psi\circ\varphi}
\end{tikzcd}$$

Figure 6. Reification with induced labeling

It is easy to prove that $\mathcal{R}eif\mathcal{NA}ut$ and $\mathcal{R}eif\mathcal{NA}ut_{\mathcal{L}}$ are isomorphic (and we identify both categories by $\mathcal{R}eif\mathcal{NA}ut$). Therefore, every reification morphism can be viewed as a $\mathcal{NA}ut$-morphism. For a $\mathcal{R}eif\mathcal{NA}ut$-morphism φ: A $\to$ B, the corresponding $\mathcal{NA}ut$-morphism is denoted by φ: A $\to$ *tc*B.

Since reifications constitute a category, the *vertical compositionality is achieved.* In the following proposition, we show that, for some given reification morphisms, the morphism (uniquely) induced by the parallel composition is also a refinement morphism and thus, the *horizontal compositionality is (also) achieved.*

Proposition 2.24 Let $\{\varphi_i: N_{1_i} \to tcN_{2_i}\}$ be an indexed family of reifications. Then $\times_{i\in I}\varphi_i$: $\times_{i\in I} N_{1_i} \to \times_{i\in I}\, tcN_{2_i}$ is a reification.

Proof: Remember that *tc* = *cn* ∘ *nc*. Since *nc* is left adjoint to *cn* then *nc* preserves colimits and *cn* preserves limits. Since products and coproducts are isomorphic in $\mathcal{LC}at(\mathcal{CM}on)$, *tc* preserves products. Following this approach, it is easy to prove that $\times_i \varphi_i$ is a reification morphism. ❑

2.4 Synchronization and Hiding of Reifications

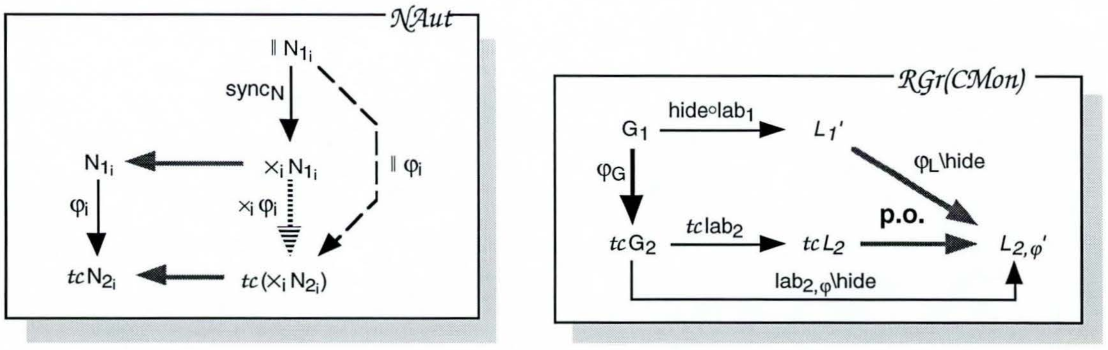
The synchronization of reified automata is the synchronization of the source automata whose reification is induced by the component reifications. Note that, in the following construction, we assume that the horizontal compositionality requirement is satisfied. In what follows, suppose that k is in {1, 2} and that i is in I where I is a set (for simplicity, we omit that i ∈ I). Remember that *tc* preserves products (previous proposition) and that every synchronization morphism has a cartesian lifting at the automata level.

Definition 2.25 Synchronization of Reifications. Consider the Figure 7 (left). Let $\{\varphi_i: N_{1_i} \to tcN_{2_i}\}$ be an indexed family of reifications where $N_{k_i} = \langle G_{k_i}, L_{k_i}, lab_{k_i} \rangle$. Let $sync_L$: Table $\to \times_i L_{1_i}$ be a synchronization morphism and

$\mathsf{sync_N}\colon \parallel \mathsf{N_{1_i}} \to \times_i \mathsf{N_{1_i}}$ be its cartesian lifting. The reification of the synchronized automaton $\parallel \mathsf{N_{1_i}}$ is $\parallel \varphi_i\colon \parallel \mathsf{N_{1_i}} \to tc(\times_i \mathsf{N_{2_i}})$ such that $\parallel \varphi_i = \times_i \varphi_i \circ \mathsf{sync_N}$ where $\times_i \varphi_i$ is uniquely induced by the product construction. ❑

The hiding of a reification is induced by the hiding of the source automaton.

Definition 2.26 Hiding of a Reification. Consider the Figure 7 (right). Let $\varphi\colon \mathsf{N_1} \to tc\mathsf{N_2}$ be a reification where $\mathsf{N_k} = \langle \mathsf{G_k}, L_k, \mathsf{lab_k}\rangle$ e $\varphi = \langle \varphi_\mathsf{G}, \varphi_\mathsf{L}\rangle$. Let $\mathsf{lab}\colon L_1 \to L_1'$ be a hiding morphism and $\mathit{hide}\mathsf{N_1} = \langle \mathsf{G_1}, L_1', \mathsf{hide}\circ\mathsf{lab_1}\rangle$ the hidden automaton. Then, the hiding of the reification morphism is $\mathit{hide}\varphi = \langle \varphi_\mathsf{G}, \varphi_\mathsf{L}\backslash\mathsf{hide}\rangle$. ❑



$$\begin{tikzcd}
 & \parallel \mathsf{N_{1_i}} \arrow[d, "\mathsf{sync_N}"'] \arrow[dd, bend left, dashed, "\parallel \varphi_i"] \\
\mathsf{N_{1_i}} \arrow[d, "\varphi_i"'] & \times_i \mathsf{N_{1_i}} \arrow[l] \arrow[d, dotted, "\times_i \varphi_i"'] \\
tc\mathsf{N_{2_i}} & tc(\times_i \mathsf{N_{2_i}}) \arrow[l]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{G_1} \arrow[r, "\mathsf{hide}\circ\mathsf{lab_1}"] \arrow[d, "\varphi_\mathsf{G}"'] & L_1' \arrow[dr, "\varphi_\mathsf{L}\backslash\mathsf{hide}"] & \\
tc\mathsf{G_2} \arrow[r, "tc\mathsf{lab_2}"] \arrow[rr, bend right, "\mathsf{lab}_{2,\varphi}\backslash\mathsf{hide}"'] & tcL_2 \arrow[r] \arrow[u, phantom, "\mathbf{p.o.}"] & L_{2,\varphi}'
\end{tikzcd}$$

Figure 7 Synchronization and hiding of reifications

3 Language Nautilus and its Semantics

In this brief introduction to the language Nautilus we present between parentheses, some key words in order to help identifying each features in the following examples. The specification of an object in Nautilus depends on if it is a simple object or an object resulting of an encapsulation (`view`), aggregation (`aggregation`) or reification (`over`). In any case, a specification has two main parts: interface and body. The interface declares imported (`import` - only for the simple object) and exported (`export`) actions and the category (`category`) of some actions (`birth, death, request`). The body (`body`) declares the attributes (`slot` - only for the simple object) and the methods of all actions. An action (`act`) may occur spontaneously, under request or both depending on if its specification and use. A birth or death action may occur at most one time (and determines the birth or the death of the object). An action may occur if its enabling (`enb`) condition holds. An action with alternatives (`alt`) is enabled if at least one alternative is enabled. In this case, only one enabled alternative may occur where the choice is an internal nondeterminism. The evaluation of an action (or an alternative within an action) is atomic. The clauses of an action may be composed in a sequential (`seq/end seq`) or multiple (`cps/end cps`) ways. A multiple composition is a special composition of concurrent clauses generalizing the notion of a multiple assignment where the valuation (`val`) clauses are evaluated before the results are assigned to the corresponding slots. Due to space restrictions, we introduce some details of the language Nautilus through examples and, at the same time, we give its semantics using nonsequential automata.

3.1 Simple Object

The first example introduces a simple object in Nautilus. In what follows, for an attribute a, $✪_\mathsf{a}$ denotes its initial (birth) value. For instance, the set of all possible values of an attribute a of type boolean is $\{✪_\mathsf{a}, \mathsf{F_a}, \mathsf{T_a}\}$.

Example 3.1 Consider object `Obj` below (at this moment, do not consider the rightmost column). Note that the birth action `Start` has two alternatives. Both alternatives are always enabled, since they do not have enabling conditions. However, since it is a birth action, it occurs only once. Due to the enabling conditions, each action occurs once and in the following order: `Start`, `Proc` and `Finish`.

```
object Obj
export
  Start  Proc  Finish
category
  birth Start
  death Finish
```

```
body
  slot a: boolean
  slot b: boolean
  act Start
    alt S1
      seq
        val a << false                                          t1: ⊛a → Fa
        val b << a                                   t2: ⊛b → Fb, t3: ⊛b → Tb
      end seq
    alt S2
      cps
        val a << false                                          t1: ⊛a → Fa
        val b << true                                           t3: ⊛b → Tb
      end cps
  act Proc
    enb a = false
    cps
      val a << true                                             t4: Fa → Ta
      val b << true                      t3: ⊛b → Tb, t5: Fb → Tb or t6: Tb → Tb
    end cps
  act Finish
    enb a = true and b = true                                   t7: Ta⊕Tb → ✝
end Obj
```

❑

Considering that an action may be a (possible complex) composition of clauses in a sequential or multiple ways, the semantics of an independent object in Nautilus is given by a reification morphism as follows:

- the target or base object reflects all possible computations over the attributes involved and therefore, it is able to implement any object specified over this attributes. It is defined as the computations of an automaton whose *CMon*-object of states is freely generated by the set of all possible values of all slots and the *CMon*-object of transitions is freely generated by the set of all possible transitions between values of component attributes;
- the source object is a relabeled restriction of the target. It is induced by a restriction followed by a relabeling using the fibration and cofibration techniques.

Example 3.2 Consider object `Obj` of the previous example. Its semantics is given by the reification morphism Obj: $N_1 \to tcN_2$ (partially) illustrated in the Figure 8 where the restriction of tcN_2 that induces N_1 is represented using a different line. Note that the labeling of the automata N_1 is not extensional. The semantics is defined as follows:

a) the *CMon*-object N_2 has $V_2 = \{\circledast_a, F_a, T_a, \circledast_b, F_b, T_b, \dagger\}^{\oplus}$ as states and $T_2 = \{a(A_1, A_2), b(B_1, B_2), death(A_1 \oplus B_1)\}^{\parallel}$ as transitions (free *CMon*-objects) with source and target given by $a(A_1, A_2)$: $A_1 \to A_2$, $b(B_1, B_2)$: $B_1 \to B_2$ and $death(A_1 \oplus B_1)$: $A_1 \oplus B_1 \to \dagger$ where A_k and B_k are values of a and b, respectively. Consider the following labeling which has correspondence in `Obj` (see the rightmost column in `Obj`):

   $a(\circledast_a, F_a) \mapsto t_1 \quad b(\circledast_b, F_b) \mapsto t_2 \quad b(\circledast_b, T_b) \mapsto t_3 \quad a(F_a, T_a) \mapsto t_4$
   $b(F_b, T_b) \mapsto t_5 \quad b(T_b, T_b) \mapsto t_6 \quad death(T_a \oplus T_b) \mapsto t_7$

b) the *CMon*-object N_1 is a relabeled restriction of tcN_2. Consider the restriction *restr*(tcN_2) where the functor *restr* is induced by the morphism $restr_L$ on labels determined as below according to the clauses of each action. The morphism $restr_L$ has a cartesian lifting $restr_N$: *restr*(tcN_2) $\to tcN_2$ at the automata level.

   $t_1;t_2 \mapsto t_1;t_2 \quad t_1|t_3 \mapsto t_1 \parallel t_3 \quad t_4|t_3 \mapsto t_4 \parallel t_3 \quad t_4|t_5 \mapsto t_4 \parallel t_5 \quad t_4|t_6 \mapsto t_4 \parallel t_6 \quad t_7 \mapsto t_7$

   The automaton N_1 is the resulting object of the relabeling N_1 = *relab*(*restr*(tcN_2)) where *relab* is induced by the morphism of labels $relab_L$ determined as below according to the identifications of each exported action (if not exported, the identification consider is τ, the unity of the monoid). The morphism $relab_L$ has a cocartesian lifting $relab_N$: $N_1 \to$ *restr*(tcN_2) at the automata level.

   $t_1;t_2 \mapsto$ Start $\quad t_1|t_3 \mapsto$ Start $\quad t_4|t_3 \mapsto$ Proc $\quad t_4|t_5 \mapsto$ Proc $\quad t_4|t_6 \mapsto$ Proc $\quad t_7 \mapsto$ Finish

   Therefore, the labeled transitions of N_1 are determined as follows:

   Start: $\circledast_a \oplus \circledast_b \to F_a \oplus F_b$ $\quad$ Start: $\circledast_a \oplus \circledast_b \to F_a \oplus T_b$ $\quad$ Proc: $F_a \oplus \circledast_b \to T_a \oplus T_b$
   Proc: $F_a \oplus F_b \to T_a \oplus T_b$ $\quad$ Proc: $F_a \oplus T_b \to T_a \oplus T_b$ $\quad$ Finish: $T_a \oplus T_b \to \dagger$

c) Obj: $N_1 \to tcN_2$ where Obj = $restr_N \circ relab_N$ is determined as follows (only the labels are represented):

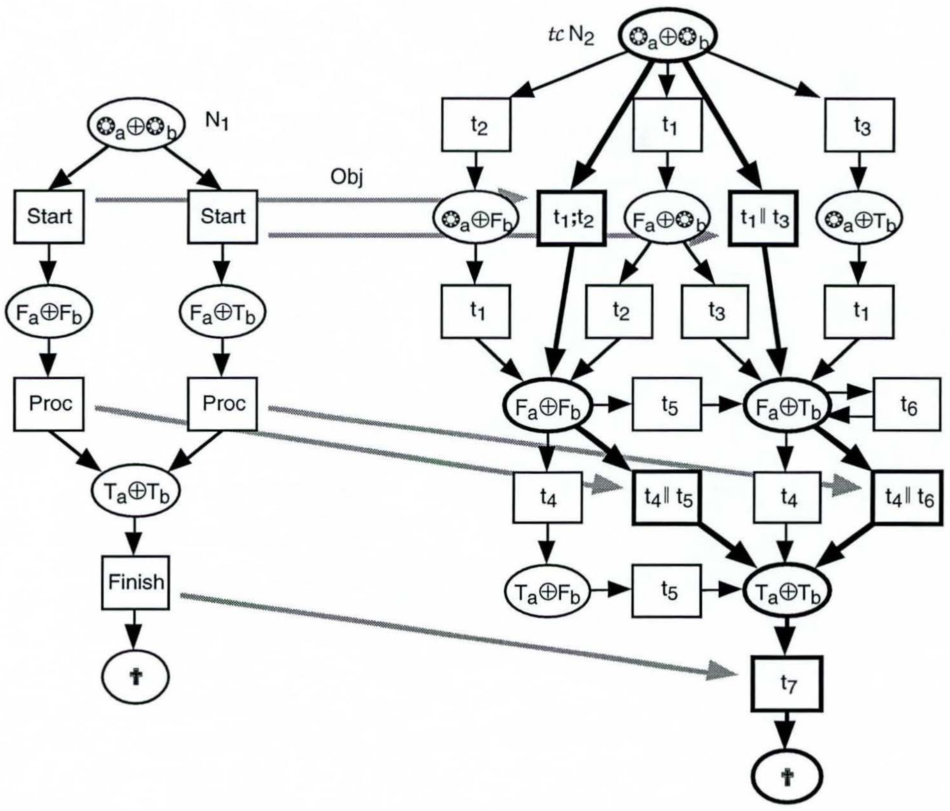


Figure 8. Semantics of an object in Nautilus as a reification morphism

$\text{Start} \mapsto t_1;t_2 \quad \text{Start} \mapsto t_1 \| t_3 \quad \text{Proc} \mapsto t_4 \| t_3 \quad \text{Proc} \mapsto t_4 \| t_5 \quad \text{Proc} \mapsto t_4 \| t_6 \quad \text{Finish} \mapsto t_7$

The state corresponding to the sum of initial values of all slots is chosen as the initial one. In this example, the initial state is $✪_a \oplus ✪_b$. ❑

3.2 Reification

An object reification is defined over previously specified object. An action may be reified into sequential or multiple composition of actions of the target object. The same action may be reified according to several alternatives, that is, a reification may be state dependent. Note that all object constructions (reification, interaction, aggregation and encapsulation) are compositional and therefore, the target object of a reification may be the resulting object of a (possible complex) construction.

Example 3.3 The object `Abstr` is implemented over the object `Concr`. Note that `Abstr` specifies alternative implementations for the action `New`. Also, `Concr` has alternatives for the action `A`.

```
object Abstr over Concr
export
  New X
category
  birth New
  death Finish
body
  act New
    alt N1
      N
```

```
    alt N2
      seq
        N
        A
        C
      end seq
  act X
    seq
      A
      B
    end seq
  act Finish
    F
end Abstr
```

```
object Concr
export
  A B C
  N F
category
  birth N
  death F
body
  slot state: 1..4
  act N
    val state << 1
  act A
    alt A1
      enb state = 1
      val state << 2
    alt A2
      enb state = 1
      val state << 2
    alt A3
      enb state = 1
      val state << 3
  act B
    enb state = 2
    val state << 4
  act C
    enb state = 3
    val state << 4
  act F
    enb state = 4
end Concr
```

❑

The semantics of a reification is a composition of reifications, i.e., the reification of the source automata over the target composed with the reification of the target over its base automata. The semantics of a reification is as in the previous section: determined by a relabeled restriction of computations of the target automata. An action of the source object may have more then one implementation which may be specified explicitly (alternatives are explicit in the source object) or implicitly (actions in the target object used for reification have alternatives). In both cases, there exist more than one transition with the same label and they have different implementations.

Example 3.4 Consider the reification of the previous example. Its semantics is given by the reification (partially) illustrated in the Figure 9 (the parentheses in the transitions help to relate the alternatives with its corresponding transition). Again the labeling is not extensional. Since the action `Finish` is not exported, the corresponding transition in the source automata is labeled by τ. The morphism illustrated in the Figure 9 composed with the reification morphism that implements `Concr` over its base automata is the semantics of `Abstr` over `Concr`. ❑

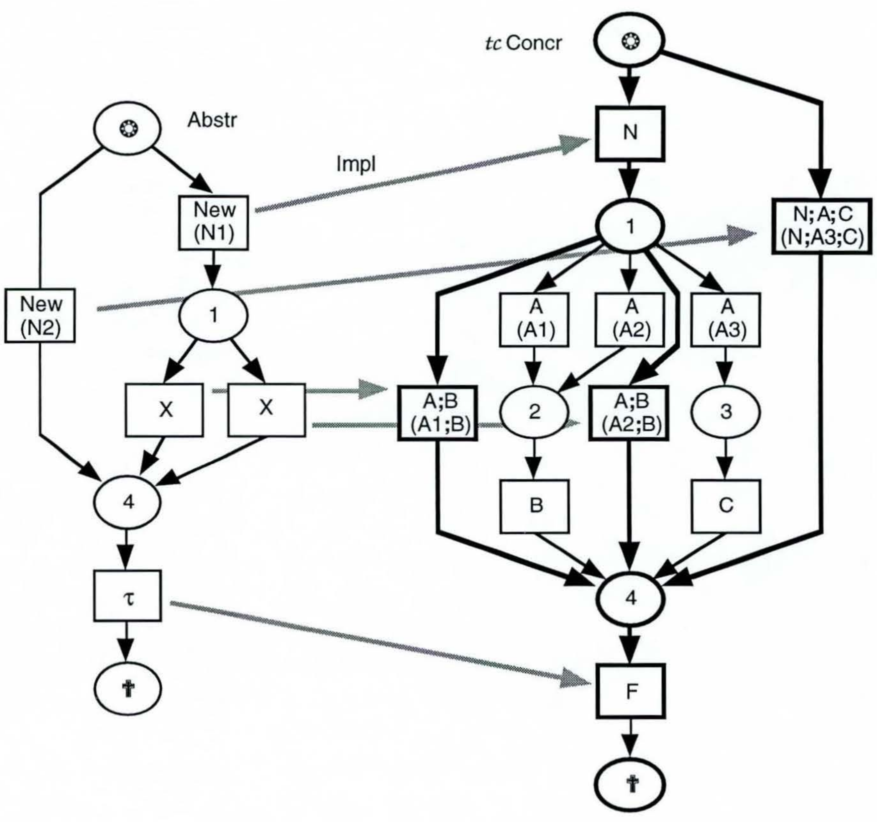


Figure 9. Semantics of a reification in Nautilus

3.3 Interaction, Aggregation and Encapsulation

Interacting (`call`) objects can be thought as a unique object with a distributed specification. In Nautilus, a reference to interacting objects (such as in a reification) is through its component objects (`interaction/end interaction`) and, following the same idea, a reference to a composed action is through its component actions (`int/end int`). In an aggregation, we can specify a relationship between component actions (`composed by`) and a relabeling of the resulting actions. For interaction/aggregation, an action of the category request may occur only if it is enabled and it is called/aggregated. The occurrence of an action may be under request (for a request action), spontaneous (for an nonrequest action which is not called/aggregated) or both (for a nonrequest action which is called/aggregated - it may occur independently of the other action). Actions may have input/output arguments (`in, out`) used for interaction or parameters (`par`) used for aggregation (where the sharing is defined by a `mach`). The arguments or parameters are declared at the interface. A view of an object can be thought as a generic relabeling where an action exported in the original object may not be exported in the resulting one, i.e., it may be encapsulated.

Example 3.5 Assume that we want to compose two objects, the `Producer` and the `Consumer`, sharing a message, in order to build a more complex one. The `Producer` is a view of the interacting objects `Prod` and `Part_Number`. The object `Part_Number` returns a random value between 1 and 2. In the following specification, note that the parameter `Pmsg` of the action `Send` in the object `Prod` is derived (`der par`) to the object `Producer`.

```
object Producer_Consumer
aggregation of
  Producer
  Consumer
export
  Prod_Random
category
  birth request Prod_Random
  death Finish
body
  act Prod_Random composed by
    Prod_Random of Producer
  act Communicate composed by
    Send of Producer
    Receive of Consumer
    match
      Send.Pmsg of Producer
      Receive.Cmsg of Consumer
  act Finish composed by
    Finish of Producer
    Finish of Consumer
end Producer_Consumer


object Producer view of
  interaction
    Prod
    Part_Number
  end interaction
export
  Prod_Random Finish
  Send der par Pmsg: natural
    by Send.Pmsg of Prod
category
  birth request Prod_Random
  death Finish
body
  act Prod_Random composed by
    int
      Produce of Prod
      Random of Part_Number
    end int
  act Send composed by
    Send of Prod
  act Finish composed by
    int
      Finish of Prod
      Close of Part_Number
    end int
end Producer


object Prod
import
  Random out PN: natural
  Close of Part_Number
export
  Produce Finish
  Send par Pmsg: natural
category
  birth Produce
  request Send
  death request Finish
body
  slot pr: 1..2
  slot num: natural
  act Produce
    call Random of Part_Number
    cps
      val num << Random.PN
      val pr << 1
    end cps
  act Send
    enb pr = 1
    cps
      val Send.Pmsg << num
      val pr << 2
    end cps
  act Finish
    enb pr = 2
    call Close of Part_Number
end Prod


object Part_Number
export
  Random out PN: natural
  Close
category
  birth request Random
  death request Close
body
  act Random
    alt
      ret Random.PN = 1
    alt
      ret Random.PN = 2
  act Close
end Part_Number
```

```
object Consumer
export
  Consume Finish
  Receive par Cmsg: natural
category
  birth request Receive
  death request Finish
body
  slot cs: 1..2
  slot inf: natural
  act Receive
    cps
      val inf << Receive.Cmsg
      val cs << 1
    end cps
  act Consume
    enb cs = 1
    val cs = 2
  act Finish
    enb cs = 2
end Consumer
```

□

The semantics of an interaction, aggregation or encapsulation in Nautilus is straightforward since it is given by a synchronization or an encapsulation of reification morphisms of nonsequential automata (an aggregation also defines a relabeling). An action with parameters, inputs or outputs is associated to a family of transitions indexed by the corresponding values.

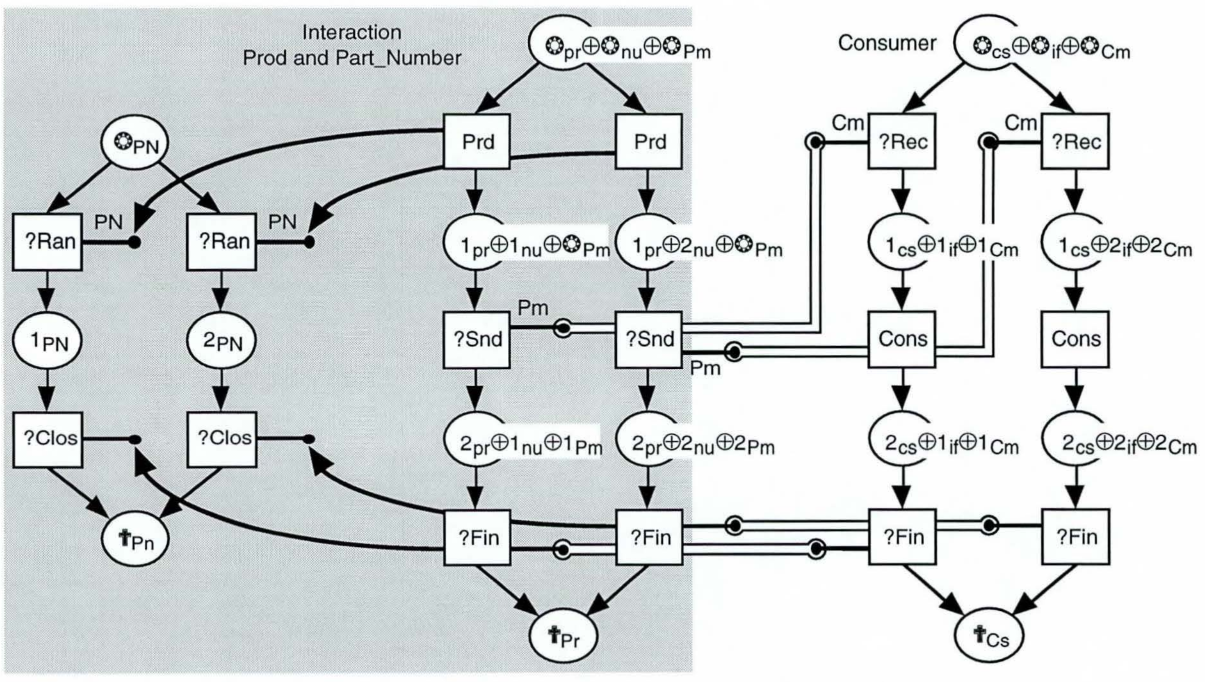
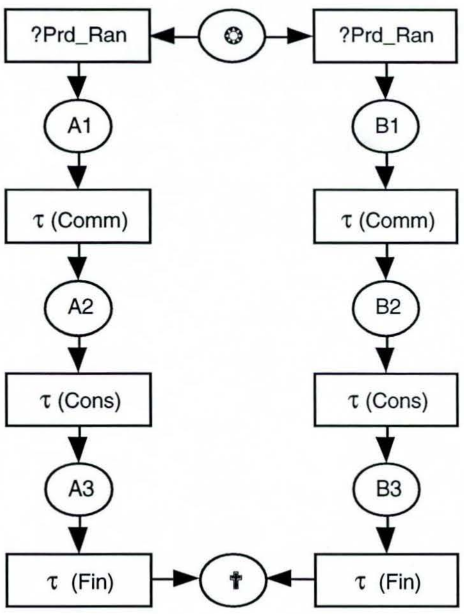


Figure 10 Relationship between component automata of an interaction and an aggregation



$$\circledast = \circledast_{PN} \oplus (\circledast_{pr} \oplus \circledast_{nu} \oplus \circledast_{Pm}) \oplus (\circledast_{cs} \oplus \circledast_{if} \oplus \circledast_{Cm})$$

$$A1 = 1_{PN} \oplus (1_{pr} \oplus 1_{nu} \oplus \circledast_{Pm}) \oplus (\circledast_{cs} \oplus \circledast_{if} \oplus \circledast_{Cm})$$

$$A2 = 1_{PN} \oplus (2_{pr} \oplus 1_{nu} \oplus 1_{Pm}) \oplus (1_{cs} \oplus 1_{if} \oplus 1_{Cm})$$

$$A3 = 1_{PN} \oplus (2_{pr} \oplus 1_{nu} \oplus 1_{Pm}) \oplus (2_{cs} \oplus 1_{if} \oplus 1_{Cm})$$

$$B1 = 2_{PN} \oplus (1_{pr} \oplus 2_{nu} \oplus \circledast_{Pm}) \oplus (\circledast_{cs} \oplus \circledast_{if} \oplus \circledast_{Cm})$$

$$B2 = 2_{PN} \oplus (2_{pr} \oplus 2_{nu} \oplus 2_{Pm}) \oplus (1_{cs} \oplus 2_{if} \oplus 2_{Cm})$$

$$B3 = 2_{PN} \oplus (2_{pr} \oplus 2_{nu} \oplus 2_{Pm}) \oplus (2_{cs} \oplus 2_{if} \oplus 2_{Cm})$$

$$\dagger = \dagger_{Pn} \oplus \dagger_{Pr} \oplus \dagger_{Cs}$$

Figure 11 Resulting (source) automaton

Example 3.6 Consider the specification in the previous example. Its semantics is given by composing a synchronization (for the interaction) followed by an encapsulation (for the view) and by another synchronization (for the aggregation). The relationship between component automata is (partially) illustrated in the Figure 10 where interactions are represented by arrows and aggregations by "tracks" and the resulting automata is (partially) illustrated in the Figure 11. In the figures, a transition whose (abbreviated) label has a question mark corresponds to a request action. ❑

4 Concluding Remarks

Nonsequential automata constitute a categorial semantic domain with full concurrency which is, for our knowledge, the first model for concurrency which satisfies the diagonal compositionality requirement, i.e., reification compose (vertically) and distributes through the parallel composition (horizontally). It is based on structured labeled transition systems. Synchronization of automata is categorically explained, by fibration techniques. Tables for synchronization are categorically defined. The hiding of transitions is also dealt with, by cofibration techniques, introducing the essential ingredient of internal non-determinism. Reification is explained using Kleisli categories. Synchronization and hiding are extended for reifications.

To experiment with the proposed semantic domain, a semantics for a concurrent, object-based language is given. The language named Nautilus is based on the object-oriented language GNOME, which is a simplified and revised version of OBLOG. Some features not present on GNOME such as reification (implementation of an object over computations of another) and aggregation (composition of objects in order to build a more complex one) are introduced. While in an interaction (also present in Nautilus) the relationship between objects is defined within the component objects, in an aggregation it is defined externally. Interaction, aggregation and reification may be state dependent. Also, in Nautilus, it is possible to extract a view from an existing object.

Considering that an action of an object in Nautilus may be a sequential or concurrent composition of clauses, executed in an atomic way, the semantics of an object in Nautilus is given by a reification morphism where the target automata called base is determined by the computations of a freely generated automata able to simulate any object specified over the involved attributes and the source automata is a relabeled restriction of the base. The semantics of a reification is the reification of the source automata over the target composed with the reification of the target over its base. The semantics of an interaction, aggregation or encapsulation is given by a synchronization or hiding of reifications of nonsequential automata. In this context, the diagonal compositionality is essential.

With respect to further works, the next step is to reintroduce in the language Nautilus some of the forgotten features of GNOME such as classes and inheritance. Also interesting is the clarification of the relationship of the nonsequential automata with logics, following the work in [Fiadeiro & Costa 94] and extending the work in [Menezes & Costa 95].

References

[Asperti & Longo 91] A. Asperti & G. Longo, *Categories, Types and Structures - An Introduction to the Working Computer Science*, Foundations of Computing (M. Garey, A. Meyer, Eds.), MIT Press, 1991.

[Bednarczyk 88] M. A. Bednarczyk, *Categories of Asynchronous Systems*, Ph.D. thesis, technical report 1/88, University of Sussex, 1988.

[Costa *et al* 92] J. F. Costa, A. Sernadas, C. Sernadas & H. D. Ehrich, *Object Interaction*, Mathematical Foundations of Computer Science '92 (I. Havel, V. Koubek, Eds.), pp. 200-208, LNCS 629, Springer-Verlag, 1992.

[Costa *et al* 93] J. F. Costa, A. Sernadas & C. Sernadas, *Data Encapsulation and Modularity: Tree Views of Inheritance*, Mathematical Foundation of Computer Science '93, (A. Borzyszkowski, S. Sokolowski, Eds.), pp. 382-391, LNCS 711, Springer-Verlag, 1993.

[Costa *et al* 94] J. F. Costa, A. Sernadas & C. Sernadas, *Object Inheritance Beyond Subtyping*, Acta Informatica 31, pp. 5-26, Springer-Verlag, 1994.

[Ehrich & Sernadas 90] H. D. Ehrich & A. Sernadas, *Algebraic Implementation of Objects over Objects*, Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (J. W. de Bakker, W. -P. de Roever, G. Rozenberg, Eds.), pp. 239-266, Springer-Verlag, 1990.

[Fiadeiro & Costa 94] J. Fiadeiro & J. F. Costa, *Mirror, Mirror in My Hand... A Duality Between Specifications and Models of Process Behavior*, accepted for publication in Mathematical Structures in Computer Science.

[Gorrieri 90] R. Gorrieri, *Refinement, Atomicity and Transactions for Process Description Language*, Ph.D. thesis, Università di Pisa, 1990.

[Hoare 85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[Mac Lane 71] S. Mac Lane, *Categories for the Working Mathematician*, Springer-Verlag, 1971.

[Mazurkiewicz 88] A. Mazurkiewicz, *Basic Notion of Trace Theory*, REX 88: Linear Time, Branching Time and Partial Orders in Logic and Models for Concurrency (J. W. de Bakker, W. -P. de Roever, G. Rozenberg, Eds.), pp. 285-363, LNCS 354, Springer-Verlag, 1988.

[Menezes & Costa 93] P. B. Menezes & J. F. Costa, *Synchronization in Petri Nets*, accepted for publication in Fundamenta Informaticae, Annales Societatis Mathematicae Polonae, IOS Press.

[Menezes & Costa 95] P. B. Menezes & J. F. Costa, *Compositional Reification of Concurrent Systems*, Journal of the Brazilian Computer Society - Special Issue on Parallel Computation, No. 1, Vol. 2, pp. 50-67, 1995.

[Menezes & Costa 95b] P. B. Menezes & J. F. Costa, *Systems for System Implementation*, in Proceedings of the 14th International Congress on Cybernetics, accepted for publication in the Journal of Cybernetics.

[Menezes *et al* 95] P. B. Menezes, J. F. Costa & A. Sernadas, *Refinement Mapping for (Discrete Event) System Theory*, to appear in the Proceedings of the Fifth International Conference on Computer Aided System Technology, EUROCAST 95, LNCS, Springer-Verlag.

[Meseguer & Montanari 90] J. Meseguer & U. Montanari, *Petri Nets are Monoids*, Information and Computation 88, pp. 105-155, Academic Press, 1990.

[Milner 89] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[Reisig 85] W. Reisig, *Petri Nets: An Introduction*, EATCS Monographs on Theoretical Computer Science 4, Springer-Verlag, 1985.

[Sassone *et al* 93] V. Sassone, M. Nielsen & G. Winskel, *A Classification of Models for Concurrency*, CONCUR 93: 4th International Conference of Concurrency (E. Best, Ed.), pp. 82-96, LNCS 715, Springer-Verlag, 1993.

[Sernadas & Ehrich 90] A. Sernadas & H. D. Ehrich, *What is an Object, After All*, Object-oriented Databases: Analysis, Design and Construction (R. Meersman, W. Kent, S. Khosla, Eds.), pp. 39-69, North-Holland, 1991.

[Sernadas & Ramos 94] A. Sernadas & J. Ramos, *A Linguagem GNOME: Sintaxe, Semântica e Cálculo*, technical report, Universidade Técnica de Lisboa, Instituto Superior Técnico, Lisbon, 1994.

[Sernadas *et al* 92] A. Sernadas, J. F. Costa & C. Sernadas, *Especificação de Objetos com Diagramas: Abordagem OBLOG*, technical report, Universidade Técnica de Lisboa, Instituto Superior Técnico, Lisbon, 1992. Prêmio Descartes 1992.

[SernadasC *et al* 91] C. Sernadas, P. Resende, P. Gouveia & A. Sernadas, *In-the-Large Object-Oriented Design of Information Systems*, The Object-Oriented Approach in Information Systems (F. van Assche, B. Moulin, C. Rolland, Eds.), pp. 209-232, North-Holland, 1991.

[SernadasC *et al* 92] C. Sernadas, P. Gouveia & A. Sernadas, *OBLOG: Object-Oriented, Logic-Based Conceptual Modeling*, technical report, Universidade Técnica de Lisboa, Instituto Superior Técnico, Lisbon, 1992.

[SernadasC *et al* 92b] C. Sernadas, P. Gouveia, J. Gouveia & P. Resende, *The Reification Dimension in Object-Oriented Database Design*, Specification of Data Base Systems (D. Harper, M. Norrie, Eds.), pp. 275-299, Springer-Verlag, 1992.

[Szabo 78] M. E. Szabo, *Algebra of Proofs*, Studies in Logic and the Foundations of Mathematics, vol. 88, North-Holland, 1978.

[Winskel 87] G. Winskel, *Petri Nets, Algebras, Morphisms and Compositionality*, Information and Computation 72, pp. 197-238, Academic Press, 1987.