UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Refinement in a Concurrent, Object-Based Language

por
Paulo Blauth Menezes, A.Sernadas and Félix Costa

RP 263 Maio/1996

UFRGS-II-CPGCC
Caixa Postal 15064 - CEP 91501-970
Porto Alegre RS BRASIL
Telefone: (051)316-6155
Fax: (051) 336-5576
Email: pgcc@inf.ufrgs

Refinement in a Concurrent, Object-Based Language *

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

† Departamento de Informática Teórica, Instituto de Informática, Universidade Federal do Rio Grande do Sul
Caixa Postal 15064, 91501-970, Porto Alegre, Brazil - blauth@inf.ufrgs.br

†† Departamento de Matemática, Instituto Superior Técnico, Universidade Técnica de Lisboa
Av. Rovisco Pais, 1096 Lisboa Codex, Portugal - 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 restriction and relabeling are functorial and a class of morphisms stands for refinement. It is, for our knowledge, the first model for concurrency which satisfies the diagonal compositionality requirement, i.e., refinements 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 refinement. The diagonal compositionality is an essential property to give semantics in this context.

1 Introduction

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

Taking into consideration the developments in Petri net theory (mainly with seminal papers like [Winskel 87], [Meseguer & Montanari 90] and [Sassone *et al* 93]) it was clear that nets might be good candidates. However, most of net-based models such as Petri nets in the sense of [Reisig 85] and labeled transition systems (see [Milner 89]) lack composition operations (modularity) and abstraction mechanisms in their original definitions. This motivate the use of the category theory: the approach in [Winskel 87] provides the former, where categorical constructions such as product and coproduct stand for system composition, and the approach in [Meseguer & Montanari 90] provides the later for Petri nets where a special kind of net morphism corresponds to the notion of implementation. Also, category theory provides powerful techniques to unify different categories of models (i.e., classes of models categorically structured) through adjunctions (usually reflections and coreflections) expressing the relation of their semantics as in [Sassone *et al* 93], where a formal framework for classification of models for concurrency is set.

A nonsequential automaton (first introduced in [Menezes & Costa 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. Restriction and relabeling are functorial operations. A restriction restricts the transitions of an automaton according to some table of restrictions (at label level). A relabeling relabels the transitions of an automaton according to some relabeling morphism (at label level). A refinement 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 refinement morphisms is inspired by Kleisli categories. With respect to nonsequential automata and comparing with [Menezes *et al* 96], in this paper we revise the refinement morphisms and introduce the restriction and relabeling for refinements.

* This work was partially supported by: 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.

In [Menezes & Costa 95] and [Menezes & Costa 96] 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].

To experiment with the proposed semantic domain, a semantics for a concurrent object-based specification language (using the terminology of [Wegner 90]) 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 refinement and aggregation are introduced. A refinement implements an object over sequential or concurrent computations of another. 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. Following the proposed approach, the diagonal compositionality requirement is an essential property to give semantics for Nautilus.

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. Comparing the graphical representations of nonsequential automata and asynchronous transition systems (first introduced in [Bednarczyk 88]), the independence relation of a nonsequential automaton is explicit.

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 *CMon* denotes the category of commutative monoids and suppose that i is in I where I is a set and k is in {0, 1} (for simplicity, we omit that $i \in I$ and $k \in \{0, 1\}$). Also, for the proof or details omitted, see [Menezes *et al* 96] and [Menezes & Costa 95b].

2.1 Nonsequential Automaton

Definition 1. Nonsequential Automaton. A nonsequential automaton $N = \langle V, T, \partial_0, \partial_1, \iota, L, \mathrm{lab}\rangle$ is such that $T = \langle T, \|, \tau\rangle$, $V = \langle V, \oplus, e\rangle$, $L = \langle L, \|, \tau\rangle$ are *CMon*-objects of transitions, states and labels respectively, ∂_0, ∂_1: $T \to V$ are *CMon*-morphisms called source and target respectively, ι: $V \to T$ is a *CMon*-morphism such that $\partial_k \circ \iota = id_V$ and lab: $T \to L$ is a *CMon*-morphism such that $\mathrm{lab}(t) = \tau$ whenever there is v in V where $\iota(v) = t$. ❑

We may refer to a nonsequential automaton $N = \langle V, T, \partial_0, \partial_1, \iota, L, \mathrm{lab}\rangle$ by $N = \langle G, L, \mathrm{lab}\rangle$ where $G = \langle V, T, \partial_0, \partial_1, \iota\rangle$ is a reflexive graph internal to *CMon* (i.e., *V*, *T* are *CMon*-objects and $\partial_0, \partial_1, \iota$ are *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 refinement in Nautilus). For simplicity, 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...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 ι_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. Let $\langle \{A, B, X, Y\}^{\oplus}, \{t_1, t_2, t_3, A, B, C, X, Y\}^{\otimes}, \partial_0, \partial_1, \iota, \{x, y\}^{\otimes}, \mathrm{lab}\rangle$ 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⊕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 (see [Menezes & Costa 95] and [Menezes & Costa 95b]). ❑

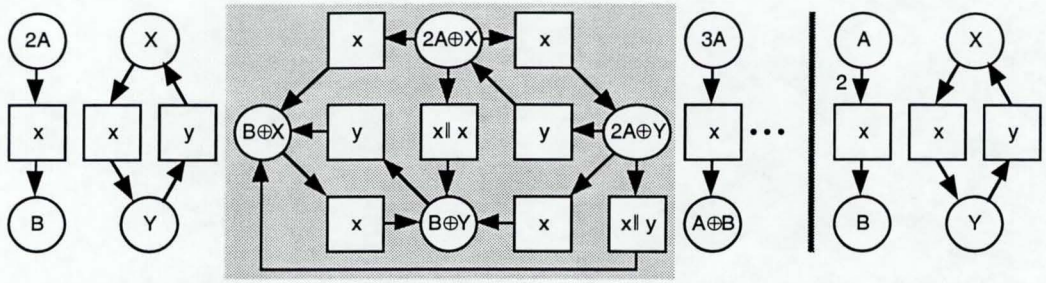

Figure 1. A nonsequential automaton (left) and the corresponding labeled Petri net (right)

Definition 3. Nonsequential Automaton Morphism. A nonsequential automaton morphism $h: N_1 \to N_2$ where $N_1 = \langle V_1, T_1, \partial_{01}, \partial_{11}, \iota_1, L_1, lab_1 \rangle$ and $N_2 = \langle V_2, T_2, \partial_{02}, \partial_{12}, \iota_2, L_2, lab_2 \rangle$ 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 $h_L \circ lab_1 = lab_2 \circ h_T$. ❑

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

Proposition 4. The category $\mathcal{NA}ut$ is complete and cocomplete with products isomorphic to coproducts. ❑

A categorical product (or coproduct) of two automata $N_1 = \langle V_1, T_1, \partial_{01}, \partial_{11}, \iota_1, L_1, lab_1 \rangle$, $N_2 = \langle V_2, T_2, \partial_{02}, \partial_{12}, \iota_2, L_2, lab_2 \rangle$ is $N_1 \times_{\mathcal{NA}ut} N_2 = \langle V_1 \times_{CMon} V_2, T_1 \times_{CMon} T_2, \partial_{01} \times \partial_{02}, \partial_{11} \times \partial_{12}, \iota_1 \times \iota_2, L_1 \times_{CMon} L_2, lab_1 \times lab_2 \rangle$ where $\partial_{k1} \times \partial_{k2}$, $\iota_1 \times \iota_2$ and $lab_1 \times lab_2$ are uniquely induced by the product construction.

Example 5. Consider the nonsequential automata Consumer and Producer (with free monoids) determined by the labeled transitions prod: A → B, send: B → A for the Producer and rec: X → Y, cons: Y → X for the Consumer. Then, the resulting objet of the parallel composition (categorial product) Consumer × Producer is illustrated in the Figure 2 (for simplicity, prod, send, rec and cons are abbreviated by p, s, r and c, respectively). ❑

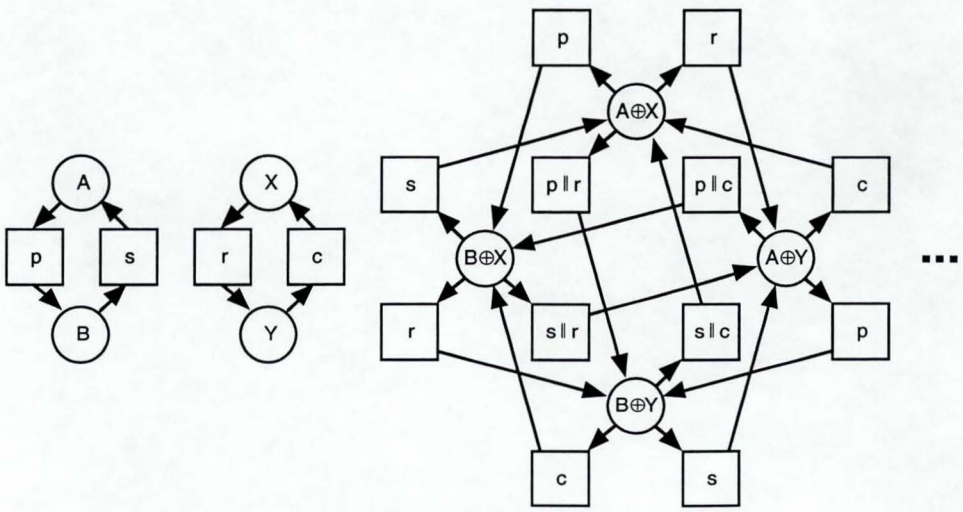

Figure 2. Parallel composition of nonsequential automata

2.2 Restriction and Relabeling

Restriction and relabeling of transitions are functorial operations defined using the fibration and cofibration techniques inspired by [Winskel 87]. Both functors are induced by a morphism at the label level. The restriction operation restricts an automaton "erasing" all those transitions which do not reflect some given table of restrictions:

a) let N be a $\mathcal{NA}ut$-object with L as the *CMon*-object of labels, *Table* be a *CMon*-object, called table of restrictions, and restr: *Table* $\rightarrow L$ be a restriction morphism. Let u: $\mathcal{NA}ut \rightarrow$ *CMon* be the obvious forgetful functor taking each automaton into its labels;
b) the functor u is a fibration and the fibers u^{-1}*Table*, $u^{-1}L$ are subcategories of $\mathcal{NA}ut$. The fibration u and the morphism restr induce a functor *restr*: $u^{-1}L \rightarrow u^{-1}$*Table*. The functor *restr* applied to N provides the automaton reflecting the desired restrictions.

The steps for relabeling are as follows:

a) let N be a $\mathcal{NA}ut$-object with L_1 as the *CMon*-object of labels, relab: $L_1 \rightarrow L_2$ be a relabeling morphism. Let u be the same forgetful functor used for synchronization purpose;
b) the functor u is a cofibration (and therefore, a bifibration) and the fibers $u^{-1}L_1$, $u^{-1}L_2$ are subcategories of $\mathcal{NA}ut$. The cofibration u and the morphism relab induce a functor *relab*: $u^{-1}L_1 \rightarrow u^{-1}L_2$. The functor *relab* applied to N provides the automaton reflecting the desired relabeling.

Restriction. In what follows, we show that the forgetful functor which takes each nonsequential automaton onto its labels is a fibration and then we introduce the restriction functor.

Proposition 6. The forgetful functor u: $\mathcal{NA}ut \rightarrow$ *CMon* that takes each nonsequential automaton onto its underlying commutative monoid of labels is a fibration.

Proof: Let $\mathcal{RG}r$(*CMon*) be the category of reflexive graphs internal to *CMon* and let *id*: $\mathcal{RG}r$(*CMon*) $\rightarrow$ $\mathcal{RG}r$(*CMon*), *emb*: *CMon* $\rightarrow$ $\mathcal{RG}r$(*CMon*) be functors. Then, $\mathcal{NA}ut$ 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 $G_2 = \langle V_2, T_2, \partial_{02}, \partial_{12}, \iota_2\rangle$ is a $\mathcal{RG}r$(*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 7. Functor restr. Consider the fibration u: $\mathcal{NA}ut \rightarrow$ *CMon*, the automata $N = \langle V, T, \partial_0, \partial_1, \iota_i, L, \text{lab}\rangle$ and the restriction morphism restr: *Table* $\rightarrow L$. The restriction of N is given by the functor *restr*: $u^{-1}(L) \rightarrow u^{-1}$(*Table*) induced by u and restr applied to N. ❑

In the following example, we show that restriction operation can be used for synchronization. For further details on synchronization, see [Menezes & Costa 93] and [Menezes *et al* 96].

Example 8. Restriction × Synchronization. Since the product (or coproduct) construction stands for parallel composition, it is possible to define a synchronization operation using the restriction operation. For instance, consider the nonsequential automata Consumer and Producer of the previous example. Suppose a joint behavior sharing the transitions send and rec (a communication without buffer such as in CSP [Hoare 85] or CCS [Milner 89]), represented by send|rec. Then, *Table* = {prod, cons, send|rec}$^{\|}$ and the restriction morphism is such that prod $\mapsto$ prod, cons $\mapsto$ cons and send|rec $\mapsto$ send$\|$rec. The synchronized automaton is given by *restr*(Consumer × Producer) as illustrated in the Figure 3. Note that the transitions send, rec are erased and send|rec is included. ❑

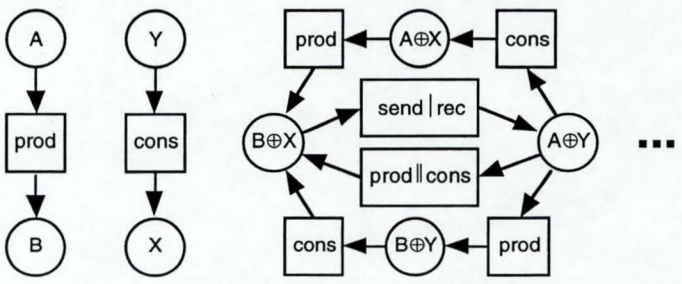

Figure 3. Synchronized automaton

Relabeling. In what follows, we show that the forgetful functor which takes each nonsequential automaton into its labels is a fibration and then we introduce the restriction functor.

Proposition 9. The forgetful functor u: $\mathcal{NAut} \to \mathcal{CMon}$ that maps each automaton onto its underlying commutative monoid of labels is a cofibration.

Proof: Let $f: L_1 \to L_2$ be a $\mathcal{CMon}$-morphism and $N_1 = \langle V_1, T_1, \partial_{01}, \partial_{11}, \iota_1, L_1, lab_1 \rangle$ be an automaton. Define $N_2 = \langle V_1, T_1, \partial_{01}, \partial_{11}, \iota_1, L_2, f \circ lab_1 \rangle$. 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 10. Functor relab. Consider the fibration u: $\mathcal{NAut} \to \mathcal{CMon}$, the nonsequential automata $N = \langle V, T, \partial_0, \partial_1, \iota, L_1, lab \rangle$ and the relabeling morphism relab: $L_1 \to L_2$. The relabeling of N satisfying relab is given by the functor *relab*: $u^{-1}L_1 \to u^{-1}L_2$ induced by u and relab applied to N. ❑

2.3 Refinement

A refinement 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 $\mathcal{CMon}$ is denoted by $Cat(\mathcal{CMon})$. We introduce the category $\mathcal{L}Cat(\mathcal{CMon})$ which can be viewed as a generalization of labeling on $Cat(\mathcal{CMon})$. There is a forgetful functor from $\mathcal{L}Cat(\mathcal{CMon})$ into $\mathcal{NAut}$. This functor has a left adjoint which freely generates a nonsequential automaton into a labeled internal category. The composition of both functors from $\mathcal{NAut}$ into $\mathcal{L}Cat(\mathcal{CMon})$ leads to an endofunctor, called transitive closure. The composition of refinements 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 refinement distributes over the parallel composition and therefore, the resulting category of automata and refinements *satisfies the diagonal compositionality*.

Definition 11. Category $\mathcal{L}Cat(\mathcal{CMon})$. Consider the category $Cat(\mathcal{CMon})$. The category $\mathcal{L}Cat(\mathcal{CMon})$ is the comma category $id_{Cat(\mathcal{CMon})} \downarrow id_{Cat(\mathcal{CMon})}$ where $id_{Cat(\mathcal{CMon})}$ is the identity functor in $Cat(\mathcal{CMon})$. ❑

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

Proposition 12. The category $\mathcal{L}Cat(\mathcal{CMon})$ has all (small) products and coproducts. Moreover, products and coproducts are isomorphic.

Definition 13. Functor cn. Let $\mathcal{N} = \langle \mathcal{G}, \mathcal{L}, \mathcal{lab} \rangle$ be a $\mathcal{L}Cat(\mathcal{CMon})$-object and $\mathcal{h} = \langle \mathcal{h}_G, \mathcal{h}_L \rangle: \mathcal{N}_1 \to \mathcal{N}_2$ be a $\mathcal{L}Cat(\mathcal{CMon})$-morphism. The functor *cn*: $\mathcal{L}Cat(\mathcal{CMon}) \to \mathcal{NAut}$ is such that:

a) the $Cat(\mathcal{CMon})$-object $\mathcal{G} = \langle V, T, \partial_0, \partial_1, \iota, ; \rangle$ is taken into the $\mathcal{RGr}(\mathcal{CMon})$-object $G = \langle V, T', \partial_0', \partial_1', \iota' \rangle$, where T' is T subject to the equational rule below and ∂_0', ∂_1', ι' are induced by ∂_0, ∂_1, ι considering the monoid T'; the $Cat(\mathcal{CMon})$-object $\mathcal{L} = \langle V, L, \partial_0, \partial_1, \iota, ; \rangle$ is taken into the $\mathcal{CMon}$-object L', where L' is L subject to the same equational rule; the $\mathcal{L}Cat(\mathcal{CMon})$-object $\mathcal{N} = \langle \mathcal{G}, \mathcal{L}, \mathcal{lab} \rangle$ is taken into the $\mathcal{NAut}$-object $N = \langle G, L', lab \rangle$ where lab is the $\mathcal{RGr}(\mathcal{CMon})$-morphism canonically induced by the $Cat(\mathcal{CMon})$-morphism $\mathcal{lab}$;

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

b) the $\mathcal{L}Cat(\mathcal{CMon})$-morphism $\mathcal{h} = \langle \mathcal{h}_G, \mathcal{h}_L \rangle: \mathcal{N}_1 \to \mathcal{N}_2$ with $\mathcal{h}_G = \langle h_{NV}, h_{NT} \rangle$, $\mathcal{h}_L = \langle h_{LV}, h_{LT} \rangle$ is taken into the $\mathcal{NAut}$-morphism $h = \langle h_{NV}, h_{NT'}, h_{LT'} \rangle: N_1 \to N_2$ where $h_{NT'}$ and $h_{LT'}$ are the monoid morphisms induced by h_{NT} and h_{LT}, respectively. ❑

The functor *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 14. Functor nc. Let $A = \langle G, L, lab \rangle$ be a $\mathcal{NAut}$-object and $h = \langle h_G, h_L \rangle: A_1 \to A_2$ be a $\mathcal{NAut}$-morphism. The functor *nc*: $\mathcal{NAut} \to \mathcal{L}Cat(\mathcal{CMon})$ is such that:

a) the $\mathcal{RGr}(\mathcal{CMon})$-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 $Cat(\mathcal{CMon})$-object $\mathcal{G} = \langle V, T^c, \partial_0^\vee, \partial_1^\vee, \iota, ; \rangle$ with $T^c = \langle T^c, \otimes, \tau \rangle$, $\partial_0^\vee$, $\partial_1^\vee$, $\_;\_: T^c \times T^c \to T^c$ inductively defined as follows:

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

subject to the following equational rules:

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

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

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

d) the $\mathcal{NA}ut$-morphism $h = \langle h_V, h_T, h_L \rangle: A_1 \to A_2$ is taken into the *Cat(CMon)*-morphism $\mathcal{h} = \langle \mathcal{h}_G, \mathcal{h}_L \rangle: \mathcal{A}_1 \to \mathcal{A}_2$ where $\mathcal{h}_G = \langle h_V, h_{Tc} \rangle$, $\mathcal{h}_L = \langle !, h_{Lc} \rangle$ and h_{Tc}, h_{Lc} are the monoid morphisms generated by the monoid morphisms h_T and h_{T_L}, respectively. ❑

Proposition 15. The functor $nc: \mathcal{NA}ut \to LCat(CMon)$ is left adjoint to $cn: LCat(CMon) \to \mathcal{NA}ut$.

Definition 16. Transitive Closure Functor. The transitive closure functor is $tc = cn \circ nc: \mathcal{NA}ut \to \mathcal{NA}ut$. ❑

Example 17. Consider the nonsequential automaton with free monoids on states and transitions, determined by the transitions a: A → B and b: B → C. Then, for instance, a;2b: A⊕B → B⊕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) = (τ[B]‖a);(b‖b) = (τ[B];b)‖(a;b) = b‖(a;b) = (b;τ[C])‖(τ[A];(a;b)) = (b‖τ[A]);(τ[C]‖(a;b)) = b;a;b = ... ❑

Let $\langle nc, cn, \eta, \varepsilon \rangle: \mathcal{NA}ut \to LCat(CMon)$ be the adjunction above. Then, $T = \langle tc, \eta, \mu \rangle$ is a monad on $\mathcal{NA}ut$ such that $\mu = cn\varepsilon nc: tc^2 \to tc$ where $cn: cn \to cn$ and $nc: nc \to nc$ 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, $\eta_N: N \to tcN$ includes N into its computations and $\mu_N: tc^2N \to tcN$ flattens computations of computations into computations.

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

Definition 18. Refinement. 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 $\langle nc, cn, \eta, \varepsilon \rangle: \mathcal{NA}ut \to LCat(CMon)$. The category of nonsequential automata and refinements, denoted by $\mathcal{RefNA}ut$, 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) $\mathcal{RefNA}ut$-objects are the $\mathcal{NA}ut$-objects;
b) $\varphi = \varphi_G: N_1 \to N_2$ is a $\mathcal{RefNA}ut$-morphism where $\varphi_G: G_1 \to tcG_2$ is a $\mathcal{RG}r(CMon)$-morphism and for each $\mathcal{NA}ut$-object N, $\varphi = \eta_G: N \to N$ is the identity morphism of N in $\mathcal{RefNA}ut$;
c) let $\varphi: N_1 \to N_2$, $\psi: N_2 \to N_3$ be $\mathcal{RefNA}ut$-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 4. ❑

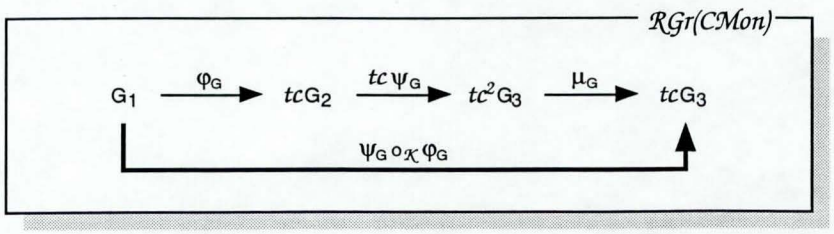

Figure 4. Composition of refinements is the composition in the Kleisli category forgetting about the labeling

In what follows, an automaton $\langle G, L, lab\rangle$ is viewed as a morphism $lab: G \to emb\,L$ (see the proposition about fibration). For simplicity, in diagrams, it is abbreviated just by $lab: G \to L$.

Definition 19. Refinement with Induced Labeling. 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 $\langle nc, cn, \eta, \varepsilon\rangle$. The category of nonsequential automata and refinements with induced labeling, denoted by $\mathcal{RefNAut}_{\mathcal{L}}$, 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) $\mathcal{RefNAut}_{\mathcal{L}}$-objects are the $\mathcal{NAut}$-objects;

b) let $\varphi_G: G_1 \to tcG_2$ be a $\mathcal{RGr(CMon)}$-morphism. Then $\varphi = \langle \varphi_G, \varphi_L\rangle: N_1 \to N_2$ is a $\mathcal{RefNAut}_{\mathcal{L}}$-morphism where φ_L is given by the pushout illustrated in the Figure 5 (left). For each $\mathcal{NAut}$-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 $\mathcal{RefNAut}_{\mathcal{L}}$ where φ_L is as above;

c) let $\varphi: N_1 \to N_2$, $\psi: N_2 \to N_3$ be $\mathcal{RefNAut}_{\mathcal{L}}$-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 5 (right). ❑

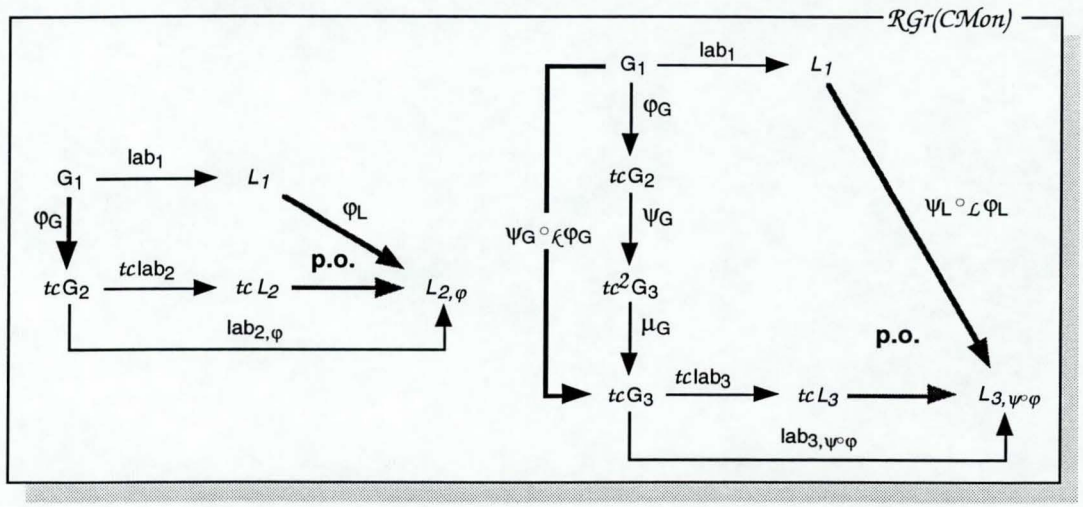

Figure 5. Refinement with induced labeling

It is easy to prove that $\mathcal{RefNAut}$ and $\mathcal{RefNAut}_{\mathcal{L}}$ are isomorphic (and we identify both categories by $\mathcal{RefNAut}$). Therefore, every refinement morphism can be viewed as a $\mathcal{NAut}$-morphism. For a $\mathcal{RefNAut}$-morphism $\varphi: A \to B$, the corresponding $\mathcal{NAut}$-morphism is denoted by $\varphi: A \to tcB$.

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

Proposition 20. Let $\{\varphi_i: N_{1i} \to tcN_{2i}\}$ be an indexed family of refinements. Then $\times_{i\in I}\varphi_i: \times_{i\in I} N_{1i} \to \times_{i\in I}\, tcN_{2i}$ is a refinement.

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

2.4 Restriction and Relabeling of Refinements

The restriction of a refinement is the restriction of the source automaton. The restriction of a community of refinements (i.e., the parallel composition of refined automata) is the restriction of the parallel composition of the source automata whose refinement is induced by the component refinements. Note that, in the following construction, we assume that the horizontal compositionality requirement is satisfied. Remember that *tc* preserves products and that every restriction morphism has a cartesian lifting at the automata level.

Definition 21. Restriction of a Refinement. Let $\varphi: N_1 \to tcN_2$ be a refinement and $restr_L$: Table $\to L_1$ be a restriction morphism and $restr_N$: $restrN_1 \to N_1$ be its cartesian lifting. The refinement of the restricted automaton $restrN_1$ is $restr\varphi$: $restrN_1 \to tcN_2$ such that $restr\varphi = \varphi \circ restr_N$. ❑

Proposition 22. Let $\{\varphi_i: N_{1_i} \to tcN_{2_i}\}$ be an indexed family of refinements where $N_{ki} = \langle G_{ki}, L_{ki}, lab_{ki}\rangle$. Let $restr_L$: Table $\to \times_i L_{1_i}$ be a restriction morphism and $restr_N$: $restrN_{1_i} \to \times_i N_{1_i}$ be its cartesian lifting. The restriction of the parallel composition of component refinements is $restr\varphi_i$: $restrN_{1_i} \to tc(\times_i N_{2_i})$ such that $restr\varphi_i = \times_i \varphi_i \circ restr_N$ where $\times_i \varphi_i$ is uniquely induced by the product construction.

Proof: Consider the Figure 6 (left). Since the horizontal compositionality requirement is satisfied, the proof is straightforward. ❑

The relabeling of a refinement is induced by the relabeling of the source automaton.

Definition 23. Relabeling of a Refinement. Consider the Figure 6 (right). Let φ: $N_1 \to tcN_2$ be a reification where $N_k = \langle G_k, L_k, lab_k\rangle$ and $\varphi = \langle \varphi_G, \varphi_L\rangle$. Let lab: $L_1 \to L_1'$ be a relabeling morphism and $relabN_1 = \langle G_1, L_1', relab \circ lab_1\rangle$ the relabeled automaton. Then, the relabeling of the reification morphism is $relab\varphi = \langle \varphi_G, relab\varphi_L\rangle$. ❑

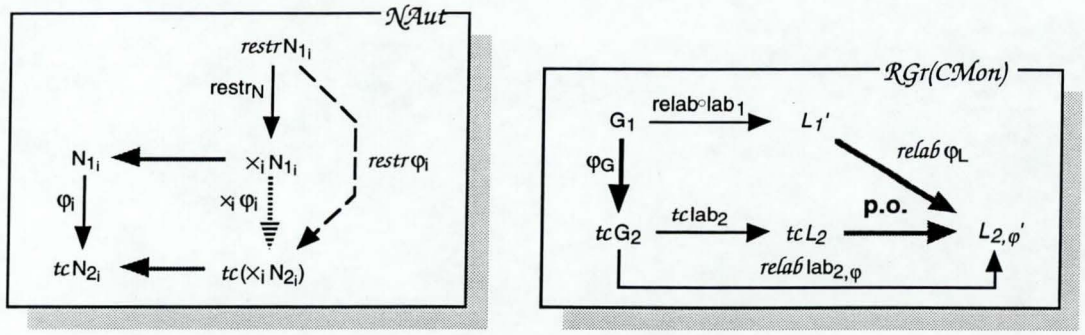

Figure 6. Restriction and relabeling of refinements

3 Language Nautilus and its Semantics

In this brief introduction to the language Nautilus we introduce some key words in order to help the understanding of the examples below. Remember that, in this paper, we do not deal with synchronization of objects. The specification of an object in Nautilus depends on if it is a simple object or a structured object such as a refinement (`over`) or a parallel composition. In any case, a specification has two main parts: interface and body. The interface declares the category (`category`) of some actions (`birth, death`). The body (`body`) declares the attributes (`slot` - only for the simple object) and the methods of all actions. 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. An action may be a sequential (`seq/end seq`) or multiple (`cps/end cps`) composition of clauses. A multiple composition is a special composition of concurrent clauses based on Dijkstra's guarded commands [Dijkstra 76] 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, $\mathbf{o}_a$ denotes its initial (birth) value. For instance, the set of values of a boolean attribute a is $\{\mathbf{o}_a, F_a, T_a\}$.

Example 24. Consider object `Obj` below (in this example, 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
category
  birth Start
  death Finish
body
  slot a: boolean
  slot b: boolean
  act Start
    alt S1                                          t0: ⊙✓ → ⊙a⊕⊙b⊕✓
      seq
        val a << false                                  t1: ⊙a → Fa
        val b << false                                  t2: ⊙b → Fb
      end seq
    alt S2                                          t0: ⊙✓ → ⊙a⊕⊙b⊕✓
      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, t6: Tb → Tb
    end cps
  act Finish
    enb a = true and b = true                       t7: Ta⊕Tb → ↑
end Obj
```

□

Since an action may be a sequential or multiple composition of clauses executed in an atomic way, the semantics of an independent object in Nautilus is given by a refinement as follows:

- 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. 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 automata is a relabeled restriction of the base.

Example 25. Consider object `Obj` of the example above. Its semantics is given by the refinement morphism Obj: $N_1 \to tcN_2$ (partially) illustrated in the Figure 7 (the part of tcN_2 used to construct N_1 is drawn using a different line). Consider the additional attribute ✓ with $\{⊙_✓, ✓\}$ as its set of values, used to control the birth of an object (in graphical representation, the value ✓ is omitted in the sums). Note that the labeling of the automata N_1 is not extensional. The semantics is defined as follows:

a) N_2 has $V_2 = \{⊙_✓, ✓, ⊙_a, F_a, T_a, ⊙_b, F_b, T_b, ↑\}^⊕$ as states and $T_2 = \{a(A_1, A_2), b(B_1, B_2), birth(⊙_✓, ⊙_a⊕⊙_b⊕✓), death(A_1⊕B_1⊕✓, ↑)\}^‖$ as transitions (free *CMon*-objects) with source and target given by $a(A_1, A_2)$: $A_1 \to A_2$, $b(B_1, B_2)$: $B_1 \to B_2$, $birth(⊙_✓, ⊙_a⊕⊙_b⊕✓)$: $⊙_✓ \to ⊙_a⊕⊙_b⊕✓$ and $death(A_1⊕B_1⊕✓, ↑)$: $A_1⊕B_1⊕✓ \to ↑$ where A_k and B_k are values of a and b, respectively. For simplicity, consider the following labeling which has correspondence in `Obj` (the rightmost column):

$birth(⊙_✓, ⊙_a⊕⊙_b⊕✓) \mapsto t_0$	$a(⊙_a, F_a) \mapsto t_1$	$b(⊙_b, F_b) \mapsto t_2$
$b(⊙_b, T_b) \mapsto t_3$	$a(F_a, T_a) \mapsto t_4$	$b(F_b, T_b) \mapsto t_5$
$b(T_b, T_b) \mapsto t_6$	$death(T_a⊕T_b) \mapsto t_7$	

b) 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 $restr_N$: $restr(tcN_2) \to tcN_2$.

$t_0;t_1;t_2 \mapsto t_0;t_1;t_2$	$t_0;(t_1 \mid t_3) \mapsto t_0;(t_1 ‖ t_3)$	$t_4 \mid t_3 \mapsto t_4 ‖ t_3$
$t_4 \mid t_5 \mapsto t_4 ‖ t_5$	$t_4 \mid t_6 \mapsto t_4 ‖ t_6$	$t_7 \mapsto t_7$

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 action. The morphism $relab_L$ has a cocartesian lifting $relab_N$: $N_1 \rightarrow restr(tcN_2)$.

$t_0;t_1;t_2 \mapsto$ Start $t_0;(t_1 \mid t_3) \mapsto$ Start $t_4 \mid t_3 \mapsto$ Proc
$t_4 \mid t_5 \mapsto$ Proc $t_4 \mid t_6 \mapsto$ Proc $t_7 \mapsto$ Finish

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

Start: $\bigcirc\!\!\!\!\checkmark \rightarrow F_a \oplus F_b \oplus \checkmark$ Start: $\bigcirc\!\!\!\!\checkmark \rightarrow F_a \oplus T_b \oplus \checkmark$ Proc: $F_a \oplus \bigcirc\!\!\!\!_b \oplus \checkmark \rightarrow T_a \oplus T_b \oplus \checkmark$
Proc: $F_a \oplus F_b \oplus \checkmark \rightarrow T_a \oplus T_b \oplus \checkmark$ Proc: $F_a \oplus T_b \oplus \checkmark \rightarrow T_a \oplus T_b \oplus \checkmark$ Finish: $T_a \oplus T_b \oplus \checkmark \rightarrow \Uparrow$

c) Obj: $N_1 \rightarrow tcN_2$ where Obj = $restr_N \circ relab_N$ is determined as below (only the labels are represented). The state $\bigcirc\!\!\!\!\checkmark$ is chosen as the initial one.

Start $\mapsto t_0;t_1;t_2$ Start $\mapsto t_0;(t_1 \| t_3)$ Proc $\mapsto t_4 \| t_3$
Proc $\mapsto t_4 \| t_5$ Proc $\mapsto t_4 \| t_6$ Finish $\mapsto t_7$

❑

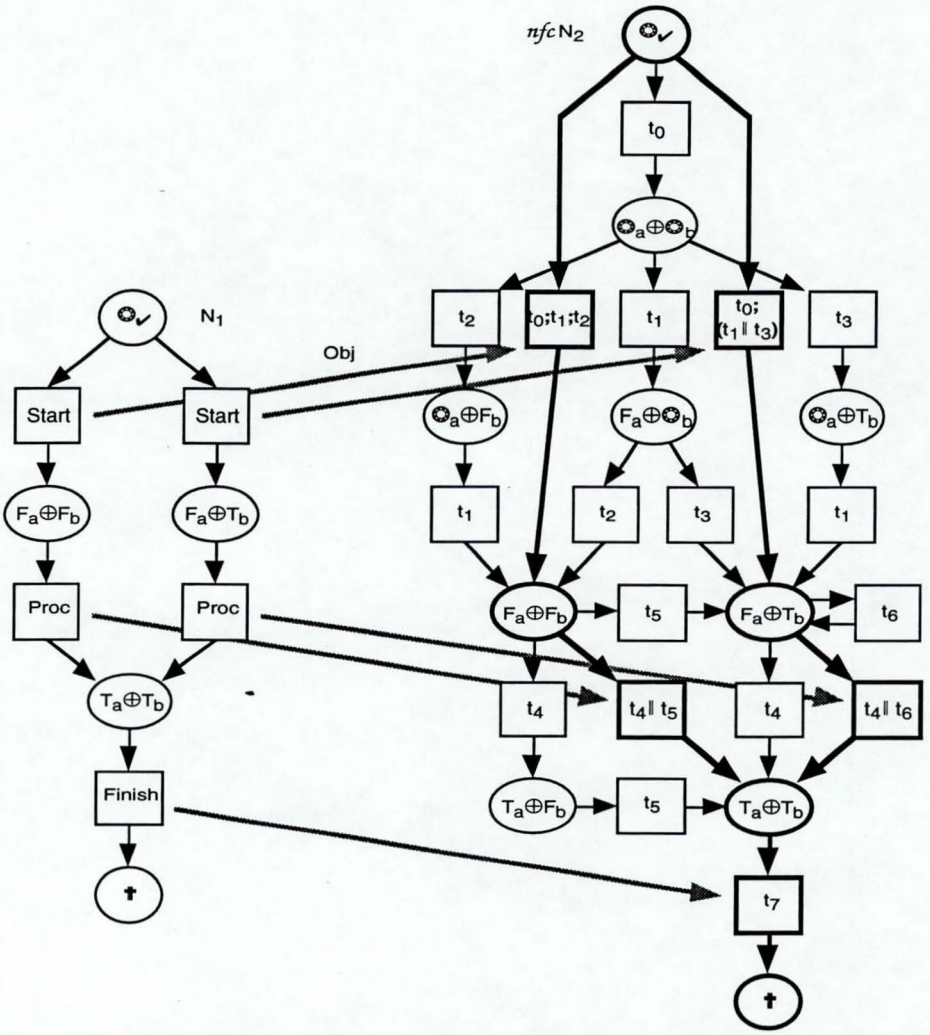

Figure 7 Semantics of an object in Nautilus as a refinement morphism

3.2 Refinement

The refinement of an object is specified over an existing object. An action may be refined into a complex action (a sequential or multiple composition of clauses) of the target object. Also, an action

may be refined according to several alternatives, that is, a refinement may be state dependent. Note that refinements are compositional and therefore, the target object of a refinement may be the source of another refinement.

Example 26. 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

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

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 refinement is a composition of refinements, i.e., the refinement of the source automata over the target composed with the refinement of the target over its base. An action of the source object may have more then one implementation which may be explicit (alternatives are explicit in the source object) or implicit (actions in the target object used in a refinement have alternatives). In both cases, there exist more than one transition with the same label and they have different implementations.

Example 27. Consider the refinement of the previous example. Its semantics is given by the refinement (partially) illustrated in the Figure 8 (the parentheses in a transition relate the alternative with its corresponding transition). Again the labeling is not extensional. The morphism illustrated in the Figure 8 composed with the refinement morphism that implements `Concr` over its base automata is the semantics of `Abstr` over `Concr`. ❑

3.3 Community of Concurrent Objects

In this context, the semantics of a community of concurrent objects in Nautilus is easily defined. It is the parallel composition of the semantics of component objects. Therefore, it is the parallel composition of refinements and again, the diagonal compositionality is an essential property. In what follows, we omit that $i \in I$ for some set $I = \{1,..., n\}$:

- a terminal object is Nautilus is an object which is not used to construct more complex objects such as the target object in a refinement;
- a unity in Nautilus is a community of (concurrent) terminal objects;
- let $\{Ob_i\}$ be the terminal objects of a unity and $\{Ob_i: N_{1i} \rightarrow tcN_{2i}\}$ be their semantics;
- the semantics of the unity is the resulting object of the categorial product of $\times Ob_i: \times N_{1i} \rightarrow tc \times N_{2i}$.

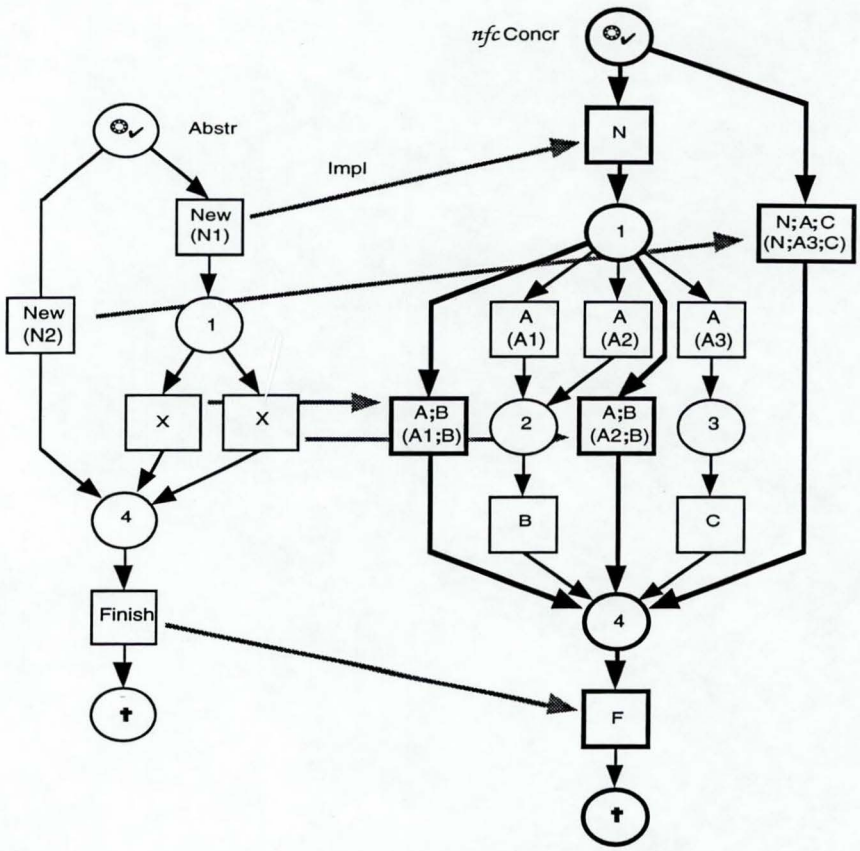

Figure 8. Semantics of a refinement in Nautilus

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., refinement compose (vertically) and distributes through the parallel composition (horizontally). It is based on structured labeled transition systems. Restriction of automata is categorically explained, by fibration technique. The relabeling of transitions is also dealt with, by cofibration technique. Refinement is explained using Kleisli categories. Restriction and relabeling are extended for refinements.

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 refinement (implementation of an object over computations of another) are introduced.

Considering that an action of an object in Nautilus may be a sequential or multiple (concurrent) composition of clauses, executed in an atomic way, the semantics of an object in Nautilus is given by a refinement 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 refinement is the refinement of the source automata over the target composed with the refinement of the target over its base. The semantics of a community of concurrent objects is given by the parallel composition of refinements of nonsequential automata. In this context, the diagonal compositionality is essential.

With respect to further works, the next step is to extend the semantics for encapsulation, aggregation and synchronization defined in Nautilus (using the restriction and relabeling

operations as sketched in this paper) and to reintroduce 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].

5 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.

[Dijkstra 76] E. W. Dijkstra, *A Discipline of Programming*, Prentice Hall, 1976.

[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, *Restriction 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 Refinement 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* 96] P. B. Menezes, J. F. Costa & A. Sernadas, *Refinement Mapping for (Discrete Event) System Theory*, Proceedings of the Fifth International Conference on Computer Aided System Technology, EUROCAST 95, pp. 103-116, LNCS 1030, Springer-Verlag, 1996.

[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 Refinement 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.

[Wegner 90] P. Wegner, *Concepts and Paradigms of Object-Oriented Programming*, OOPS Messenger, Vol. 1, No. 1, ACM Press, 1990.

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