ON THE RELATIONSHIP BETWEEN OPERATIONAL AND DENOTATIONAL SEMANTICS OF TEMPORAL LOGIC SPECIFICATION OF OBJECT BEHAVIOUR[†]

Carlos Caleiro[§]
Departamento de Matemática, Instituto Superior Técnico
Av. Rovisco Pais, 1000 Lisboa, Portugal

ABSTRACT

A simple linear propositional temporal logic is adopted for specifying object behaviour. It explicitly distinguishes action occurrence from action enabling. An institution for this logic is built and shown to provide a denotational categorial semantics to aggregation and hiding constructs. An operational domain of labelled transition systems with stuttering, extended with an acceptance condition based on the fulfilment of pending goals is proposed. Its relation to the semantic domain of the institution is exploited towards the achievement of an exact operational semantics, by summing up on the properties of the previously obtained denotational semantics.

1. Introduction

The use of temporal logics[8,10] has been widely explored both on the fields of specification[12,15] and certification of properties[7,11,13] of reactive systems, namely of object-oriented systems[18,19,21]. The advantages are known to lie on the clear declarative formalization of the systems at hand and on the use of temporal verification techniques to prove properties of the specified systems. Temporal specification has also played an important role in the search for seemly compositional specification frameworks[2]. In this paper we consider the problem of setting up a suitable operational semantics of object behaviour specifications which agrees with the denotational semantics put forth by A. Sernadas and C. Sernadas[20]. Institutions[9] and category theory[1,3] are used throughout.

Naively, we may say that an object behaves as a reactive system, in accordance with an internal state reflected in the value of its slots[17]. For the sake of simplicity, only boolean slots (observations) are considered. Special emphasis is given to one important class of observable properties of a state: its action menu, the set of actions whose occurrence is possible at that state. Hence, an object behaviour specification should consist of a description of its actions and observations, of their relationships, of the enabling conditions of actions, of the effects of action occurrences on observations and of other behaviour constraints such as fairness or liveness[16]. Capitalizing on the general method[20], we set up an object behaviour institution over a temporal logic designed for object specification, whose interpretation frames are sets of life-cycles

[†]This work was partly supported by CEC under ESPRIT-III BRA WG 6071 IS-CORE (Information Systems - COrrectness and REusability) and BRA WG 6112 COMPASS.
[§]Email: ccal@inesc.pt

and which yields a final denotational semantics to object aggregation and hiding. Some changes are made, however, in order to obtain a natural categorial linkup to the envisaged operational semantics. This counterpart is played by G-automata[6], labelled transition systems with a goal-driven acceptance condition for computations. They were first introduced for the purpose of dealing with fairness and liveness specification, since usual automata were not at all adequate[5].

2. Temporal logic specification

The specification logic we use is a simplified propositional version of the one proposed by A. Sernadas, C. Sernadas and J. F. Costa[22]. In this logic, if a is some action (or prime action, as we call it), $\triangledown a$ stands for "a occurs" and $\diamond a$ for "a is enabled to occur". We start by categorially defining its syntax.

Proposition/Definition 1 (A category of signatures)
A *signature* is a pair $\Sigma = \langle \Sigma_{obs}, \Sigma_{act} \rangle$ of sets, whose elements are called, respectively, observation and prime action symbols. A *signature morphism* $\sigma : \Sigma \to \Sigma'$ is a pair $\langle \sigma_{obs} : \Sigma_{obs} \to \Sigma'_{obs}, \sigma_{act} : \Sigma_{act} \to \Sigma'_{act} \rangle$ of set maps. Signatures and their morphisms constitute a category Sig. ∎

Obviously, Sig is $Set \otimes Set$ and hence cocomplete.

Definition 2 (Atoms and formulae)
The set $Atom_\Sigma$ of *atoms* over Σ is $\{b : b \in \Sigma_{obs}\} \cup \{\triangledown a : a \in \Sigma_{act}\} \cup \{\diamond a : a \in \Sigma_{act}\}$. The set $Form_\Sigma$ of *formulae* over Σ is the least set such that:

- $Atom_\Sigma \subseteq Form_\Sigma$;
- $(\neg\varphi), (\mathbf{X}\,\varphi), (\varphi \Rightarrow \psi), (\varphi\,\mathbf{U}\,\psi) \in Form_\Sigma$ provided that $\varphi, \psi \in Form_\Sigma$. ∎

Besides the well known logical connectives, two temporal operators were introduced. The "next" temporal operator ($\mathbf{X}$) refers, as usual, to the following instant. The "until" operator ($\mathbf{U}$) we adopt is weak and includes the evaluating point i.e., $(\varphi\,\mathbf{U}\,\psi)$ shall mean that φ holds from now on, at least until ψ first holds.

Proposition/Definition 3 (Syntax functor)
The *syntax functor* is $Syn : Sig \to Set$ such that $Syn(\Sigma) = Form_\Sigma$ for each $\Sigma \in |Sig|$, and $Syn(\sigma : \Sigma \to \Sigma') = \hat{\sigma} : Form_\Sigma \to Form_{\Sigma'}$ is the usual extension of each signature morphism σ to formulae. ∎

3. An institution of object behaviour

As was said before, this section is the result of applying the general technique for setting up an object behaviour institution[20] to the temporal logic framework at hand. The underlying idea is that we can abstract away an object from a set of life-cycles describing its allowed behaviour. As a subproduct of the construction we obtain a denotational semantics of object specifications which is the main concern of section 6.

3.1. A category of objects

We start by defining an object as a set of life-cycles over its signature. The set of life-cycles expresses exactly the possible behaviour of the object. However, we restrict attention to sets of life-cycles satisfying three conditions. The first one meaning that an object is an open system, able to interleave (and interact) with other objects. The second condition imposing that enabling of an action really corresponds to its possible occurrence. The third just requiring that suffixes of life-cycles are life-cycles of the object (no initial constraints). For the purpose, let Σ be a signature.

Definition 4 (Snapshots and life-cycles)
A *snapshot* over Σ is a set $snp \subseteq Atom_\Sigma$ such that if for some $a \in \Sigma_{act}$, $\nabla a \in snp$ then also $\diamond a \in snp$. Denote by Snp_Σ the set of all snapshots over Σ. A *life-cycle* over Σ is a map $\lambda : \mathbb{N} \to Snp_\Sigma$. Denote by Λ_Σ the set of all life-cycles over Σ. ■

Let us first state the openness condition[20]. The idea is that every action occurrence in a life-cycle can be finitely postponed in order to allow the evolution of the surrounding environment.

In the following definition, ω stands for the first infinite ordinal.

Proposition/Definition 5 (One-step delay)
Let $\lambda \in \Lambda_\Sigma$ and $i \in \mathbb{N}$. The *one-step delay* of λ at i is the life-cycle $d^i(\lambda)$ such that:

- $d^i(\lambda)_n = \lambda_n$ for $n < i$; $d^i(\lambda)_i = \lambda_i \setminus \{\nabla a : a \in \Sigma_{act}\}$; $d^i(\lambda)_n = \lambda_{n-1}$ for $n > i$.

We also define $d^\omega(\lambda) = \lambda$. ■

Definition 6 (Delay-point sequence)
A *delay-point sequence* μ is a map $\mu : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ such that:

- *monotonicity:* for every $i, j \in \mathbb{N}$, $i < j$ implies $\mu_i < \mu_j$ or $\mu_j = \omega$;
- *diligence:* for each $i \in \mathbb{N}$ there exists $j \in \mathbb{N}$ such that either $\mu_j - \mu_i > j - i$ and $\mu_j \in \mathbb{N}$ or $\mu_j = \omega$. ■

The monotonicity condition imposes that the sequence of ordinals is strictly increasing until it reaches ω. The diligence condition excludes sequences containing every natural number greater than any fixed one (nothing can be delayed forever).

Proposition/Definition 7 (Delay life-cycles and delay closure)
Let $\lambda \in \Lambda_\Sigma$ and μ be a delay-point sequence. Define the sequence of μ *delay life-cycles* of λ, $d^\mu(\lambda)$, inductively as follows:

- $d^\mu(\lambda)_0 = \lambda$;
- $d^\mu(\lambda)_{n+1} = d^{\mu_n}(d^\mu(\lambda)_n)$.

The delay relation on life-cycles $\infty \subseteq \Lambda_\Sigma \otimes \Lambda_\Sigma$ is defined by:

- $\lambda \infty \lambda'$ iff there exists a delay-point sequence μ such that for each $n \in \mathbb{N}$ there exists $i \in \mathbb{N}$ with $\lambda'_k = (d^\mu(\lambda)_i)_k$ for each $k \leq n$.

The *delay closure* of $\Lambda \subseteq \Lambda_\Sigma$ is the set $\Lambda^\infty = \{\lambda' \in \Lambda_\Sigma : \lambda\infty\lambda' \text{ for some } \lambda \in \Lambda\}$. We say that Λ is delay closed iff $\Lambda = \Lambda^\infty$. ■

The other two conditions are:

Definition 8 (Occurrence closure)
Let $\Lambda \subseteq \Lambda_\Sigma$. We say that Λ is *occurrence closed* iff for every $\lambda \in \Lambda$, $n \in I\!N$ and $\diamond a \in \lambda_n$ there exists $\lambda' \in \Lambda$ such that for all $i < n$, $\lambda_i = \lambda'_i$, $\lambda_n \setminus \{\nabla x : x \in \Sigma_{act}\} = \lambda'_n \setminus \{\nabla x : x \in \Sigma_{act}\}$ and $\nabla a \in \lambda'_n$. ■

Definition 9 (Suffix closure)
Let $\Lambda \subseteq \Lambda_\Sigma$. Define the *suffix closure* of Λ to be the set $\Lambda^{suf} = \{\lambda \in \Lambda_\Sigma : \text{there exist } \lambda' \in \Lambda \text{ and } n \in I\!N \text{ such that for all } i \in I\!N, \lambda_i = \lambda'_{i+n}\}$. Furthermore, Λ is suffix closed iff $\Lambda = \Lambda^{suf}$. ■

Our notion of object can now de introduced.

Definition 10 (Object)
An *object* is a pair $ob = \langle \Sigma, \Lambda \rangle$ where Σ is a signature and Λ is a delay, occurrence and suffix closed subset of Λ_Σ. ■

We now proceed to object morphisms.

Proposition/Definition 11 (Back image of life-cycle by signature morphism)
Let $\sigma : \Sigma' \to \Sigma$ be a signature morphism and $\lambda \in \Lambda_\Sigma$. The *back image* of λ by σ is the life-cycle $\sigma^{-1}(\lambda) \in \Lambda_{\Sigma'}$ such that $\sigma^{-1}(\lambda)_n = \{\varphi \in Form_{\Sigma'} : \hat{\sigma}(\varphi) \in \lambda_n\}$ for each $n \in I\!N$. Given $\Lambda \subseteq \Lambda_\Sigma$, its back image is the set $\sigma^{-1}(\Lambda) = \{\sigma^{-1}(\lambda) : \lambda \in \Lambda\}$. ■

Definition 12 (Object morphism)
An *object morphism* $\sigma : \langle \Sigma, \Lambda \rangle \to \langle \Sigma', \Lambda' \rangle$ is a signature morphism $\sigma : \Sigma' \to \Sigma$ such that $\sigma^{-1}(\Lambda) \subseteq \Lambda'$. ■

Note that an object morphism just asserts that its domain object simulates its codomain object.

Proposition/Definition 13 (A category of objects)
Objects and their morphisms constitute a category Ob concrete over Sig^{op}. Moreover, Ob is cartesian. ■

Products in this category denote parallel composition[20].

Proposition 14 (Cofibration)
The forgetful functor $U : Ob \to Sig^{op}$ is a cofibration.

Proof: the cocartesian morphism for $\sigma : \Sigma' \to \Sigma$ on object $ob = \langle \Sigma, \Lambda \rangle$ for U is the object morphism $\sigma : ob \to ob'$, where $ob' = \langle \Sigma', \sigma^{-1}(\Lambda) \rangle$. ■

Denote by Ob_Σ the $U^{-1}(\Sigma)$ fiber subcategory of Ob under U.

3.2. The institution

We now define the foreseen institution. Signatures and syntax were defined in section 2. The model part is based on the category of objects above defined and, by indexing[23], on its cofibration over the (op)category of signatures.

Definition 15 (Satisfaction by life-cycle)
Let Σ be a signature and $\lambda \in \Lambda_\Sigma$. For each $n \in I\!N$, the *satisfaction* of formulae in $Form_\Sigma$ by λ at n ($\models_{\Sigma,n}$) is inductively defined as follows:

- $\lambda \models_{\Sigma,n} \alpha$ iff $\alpha \in \lambda_n$, provided that $\alpha \in Atom_\Sigma$;
- $\lambda \models_{\Sigma,n} (\neg\varphi)$ iff not $\lambda \models_{\Sigma,n} \varphi$, provided that $\varphi \in Form_\Sigma$;
- $\lambda \models_{\Sigma,n} (\varphi \Rightarrow \psi)$ iff $\lambda \models_{\Sigma,n} (\neg\varphi)$ or $\lambda \models_{\Sigma,n} \psi$, provided that $\varphi, \psi \in Form_\Sigma$;
- $\lambda \models_{\Sigma,n} (\mathbf{X}\,\varphi)$ iff $\lambda \models_{\Sigma,n+1} \varphi$, provided that $\varphi \in Form_\Sigma$;
- $\lambda \models_{\Sigma,n} (\varphi\,\mathbf{U}\,\psi)$ iff for some $j \geq n$, $\lambda \models_{\Sigma,j} \psi$ and for all $n \leq k < j$, $\lambda \models_{\Sigma,k} \varphi$

  or else

  for every $j \geq n$, $\lambda \models_{\Sigma,j} \varphi$, provided that $\varphi, \psi \in Form_\Sigma$.

Let $\varphi \in Form_\Sigma$. We say that λ *satisfies* φ, and write $\lambda \models_\Sigma \varphi$, iff $\lambda \models_{\Sigma,n} \varphi$ for every $n \in I\!N$. ∎

Proposition/Definition 16 (Object behaviour institution)
The *object behaviour institution* is $Inst = \langle Sig, Syn, Mod, \models^{ob}\rangle$.

The functor $Mod : Sig \to Cat^{op}$ is defined by:

- $Mod(\Sigma) = Ob_\Sigma$, for each $\Sigma \in |Sig|$;
- $Mod(\sigma : \Sigma' \to \Sigma) : Ob_\Sigma \to Ob_{\Sigma'}$, for each $\sigma \in Mor(Sig)$, is such that:

  $Mod(\sigma)(\langle\Sigma, \Lambda\rangle) = \langle\Sigma', \sigma^{-1}(\Lambda)\rangle$;

  $Mod(\sigma)(id_\Sigma : \langle\Sigma, \Lambda\rangle \to \langle\Sigma, \Lambda'\rangle) = id_{\Sigma'} : \langle\Sigma', \sigma^{-1}(\Lambda)\rangle \to \langle\Sigma', \sigma^{-1}(\Lambda')\rangle$.

For each signature Σ, the satisfaction relation $\models^{ob}_\Sigma \subseteq |Mod(\Sigma)| \otimes Syn(\Sigma)$ is defined by:

- for each $\varphi \in Syn(\Sigma)$, $\langle\Sigma, \Lambda\rangle \models^{ob}_\Sigma \varphi$ iff $\lambda \models_\Sigma \varphi$ for every $\lambda \in \Lambda$. ∎

4. G-automata

We present G-automata[6], labelled transition systems with a mandatory stuttering transition, additionally equipped with a structure of goals whose intention is to restrict acceptance. This goal structure is closely related to the "fairness" assumptions on the behaviour of systems proposed by Z. Manna and A. Pnueli[12] and somehow similar to the acceptance structures of classical automata on infinite words such as Büchi and Muller automata[24]. However, G-automata arise, in our opinion, much more naturally, while still emulating their expressible power.

4.1. The structure of G-automata

The idea behind G-automata is that computations (taken as usually) are required to satisfy certain conditions related with attaching pending goals to its states and expecting that they are fulfilled by some future transition.

Definition 17 (G-automaton)
A *G-automaton* is a tuple $aut = \langle \Sigma, S, \Delta, v, G, p, f \rangle$ consisting of:

- a *signature* Σ;
- a set S of *states*;
- a set $\Delta \subseteq S \otimes 2^{\Sigma_{act}} \otimes S$ of *transitions*, such that for each $e \in 2^{\Sigma_{act}}$ and $s, r \in S$, $\delta = \langle s, e, r \rangle \in \Delta$ means that there is a transition labelled by $e \equiv act(\delta)$ leading from state $s \equiv dom(\delta)$ to state $r \equiv cod(\delta)$;
- a *state valuation map* $v : S \to 2^{\Sigma_{obs}}$, which assigns to each state its set of satisfied observations;
- a set G of *goals*;
- a *pending goals map* $p : S \to 2^G$, which defines the set of pending goals at each state;
- a *fulfilled goals map* $f : \Delta \to 2^G$, which maps each transition onto the set of goals it actually fulfils

such that the following conditions hold:

- *open-nature:* $\langle s, \emptyset, s \rangle \in \Delta$ for each $s \in S$;
- *fulfilment vs. pending goals:* $f(\delta) \subseteq p(dom(\delta))$ for each $\delta \in \Delta$;
- *goal persistence:* $p(dom(\delta)) \setminus p(cod(\delta)) \subseteq f(\delta)$ for each $\delta \in \Delta$;
- *goal consistency:* for each $s \in S$ and $g \in p(s)$ there exists a finite sequence $\delta^1 \dots \delta^n$ of transitions in Δ such that $dom(\delta^1) = s$, $dom(\delta^{i+1}) = cod(\delta^i)$ for each $i \in \{1, \dots, n-1\}$ and $g \in f(\delta^n)$. ∎

If Δ is the set of transitions of a G-automaton and s is one of its states, we assume that $\Delta(s)$ stands for $\{\delta \in \Delta : dom(\delta) = s\}$.

As must be clear, transitions are labelled with sets of prime action symbols instead of being labelled with one prime action symbol. This is due to our interest in having a semantic domain which is rich enough to express, for instance, parallel composition. In that case, and for synchronization purposes, the reasons for our choice turn to be quite natural. Note also that, in our definition of life-cycle, simultaneous prime action occurrences were not excluded.

To explain the four conditions imposed on G-automata we have to assume a pragmatic point of view. In fact, since we consider an action to be a set of prime action symbols (meaning, somehow, that all those prime actions occur from an instant in time to the following by performing a concurrent step), we have to give a meaning to the $\emptyset$ action. We claim that it should be seen as an "invisible" change of state. In case there is no change of state, it can also be seen as a stuttering transition. This is the key to the achievement of suitable support for hiding constructs and, together with the open-nature condition, for non-synchronized aggregation. Clearly, it is closely related to the delay closure of objects. Besides, the information introduced by the maps p and f concerning goals should be such that only pending goals may be fulfilled, and that every pending goal must propagate to subsequent states until some transition indeed fulfils it. The fourth condition just requires possible fulfilment of every pending goal.

Having defined G-automata, we now face the problem of choosing an adequate notion of homomorphism, which may be seen as a guarantee of behaviour simulation[6,16,25] and preservation of the goal structure.

Definition 18 (Menu of state)
Let $aut = \langle \Sigma, S, \Delta, v, G, p, f \rangle$ be a G-automaton. The *menu of state* $s \in S$ is the set $Menu_{aut}(s) = \bigcup_{\delta \in \Delta(s)} act(\delta)$. ∎

The menu of a state is the set of all prime actions whose occurrence is enabled at that state.

Definition 19 (G-automata homomorphism)
Let $aut = \langle \Sigma, S, \Delta, v, G, p, f \rangle$ and $aut' = \langle \Sigma', S', \Delta', v', G', p', f' \rangle$ be two G-automata. A *G-automata homomorphism* $h : aut \to aut'$ is a tuple $h = \langle h_{obs}, h_{act}, h_S, h_G \rangle$ such that $\langle h_{obs}, h_{act} \rangle : \Sigma' \to \Sigma$ is a signature morphism, $h_S : S \to S'$ and $h_G : G' \to G$ are set maps, and the following structure preservation conditions are satisfied:

- *simulation:* $\langle h_S(s), h_{act}^{-1}(e), h_S(r) \rangle \equiv h(\delta) \in \Delta'$ for each $\delta = \langle s, e, r \rangle \in \Delta$;
- *enabling correspondence:* $Menu_{aut'}(h_S(s)) \subseteq h_{act}^{-1}(Menu_{aut}(s))$ for each $s \in S$;
- *observational correspondence:* $h_{obs}^{-1}(v(s)) \subseteq v'(h_S(s))$ for each $s \in S$;
- *pending goals correspondence:* $h_G(g') \in p(s)$ iff $g' \in p'(h_S(s))$ for each $g' \in G'$ and $s \in S$;
- *goal fulfilment correspondence:* $h_G(g') \in f(\delta)$ iff $g \in f'(h(\delta))$ for each $g' \in G'$ and $\delta \in \Delta$. ∎

In the presence of a G-automata homomorphism as defined above, given $\delta' \in \Delta'$, we assume that $h^{-1}(\Delta, \delta')$ stands for the set $\{\delta \in \Delta : h(\delta) = \delta'\}$.

Intuitively, the *simulation* condition establishes a behavioural relationship. In fact, if we regard the source of the homomorphism as a whole system and the target as one of its components, the condition states that every transition of the whole system has a counterpart in terms of a transition of the component. The *enabling* *correspondence* condition similarly requires that the enabling of corresponding prime actions must propagate from the component to corresponding states of the whole. The same intuition applies to the *observational correspondence* condition, meaning that any observation in a state of the whole system must be supported by the relevant observations in the corresponding state of the component. The *pending goals* and *goal fulfilment correspondence* conditions force pending goals and goal fulfilment performance of a component to be carried away to the whole system.

Proposition/Definition 20 (A category of G-automata)
G-automata and their homomorphisms constitute a category $GAut$ concrete over $Sig^{op} \otimes Set \otimes Set^{op}$. Moreover, $GAut$ is cartesian. ■

Later on, it shall be clear that products in $GAut$ also denote parallel composition.

Following the strategy previously used with the category Ob, it seems reasonable to study what happens to G-automata in the presence of a signature change. Namely, if we take a look at its goal structure, it turns out that some goals, due to their fulfilment, may not be accordingly explained over the new signature and hence cannot be properly dealt with by relabelled transitions. To these goals we shall call *local*.

Definition 21 (Local goals with respect to signature morphism)
Let $aut = \langle \Sigma, S, \Delta, v, G, p, f \rangle$ be a G-automaton and $\sigma : \Sigma' \to \Sigma$ a signature morphism. We say that a goal $g \in G$ is *local* to *aut with respect to* σ if there exist $\delta^1, \delta^2 \in \Delta$ such that $dom(\delta^1) = dom(\delta^2)$, $cod(\delta^1) = cod(\delta^2)$, $\sigma_{act}^{-1}(act(\delta^1)) = \sigma_{act}^{-1}(act(\delta^2))$, $g \in f(\delta^1)$ and $g \notin f(\delta^2)$. Denote by Loc_{aut}^{σ} the set of all goals of aut which are local with respect to σ. ■

As before, the functor from G-automata which just keeps the signature is a cofibration.

Proposition 22 (Another cofibration)
The forgetful functor $V : GAut \to Sig^{op}$ is a cofibration.

Proof: the cocartesian morphism for $\sigma : \Sigma' \to \Sigma$ on $aut = \langle \Sigma, S, \Delta, v, G, p, f \rangle$ for V is the G-automata homomorphism $\langle \sigma_{obs}, \sigma_{act}, id_S, inj_G \rangle : aut \to \sigma^{-1}(aut)$, where $\sigma^{-1}(aut) = \langle \Sigma', S, \Delta', v', G', p', f' \rangle$ is such that:

- $\Delta' = \{\langle s, \sigma_{act}^{-1}(e), r \rangle : \langle s, e, r \rangle \in \Delta\}$;
- $v'(s) = \sigma_{obs}^{-1}(v(s))$ for each $s \in S$;
- $G' = G \setminus Loc_{aut}^{\sigma}$;
- $p'(s) = p(s) \cap G'$ for each $s \in S$;
- $f'(\delta') = (\bigcap_{\delta \in h^{-1}(\Delta, \delta')} f(\delta)) \cap G'$ for each $\delta' \in \Delta'$.

The map $inj_G : G' \to G$ is the injection of G' onto G. ■

Call $\sigma^{-1}(aut)$ the reduct for σ on aut and say that it is non-local if $Loc_{aut}^{\sigma} = \emptyset$.

Denote by $GAut_{\Sigma}$ the $V^{-1}(\Sigma)$ fiber subcategory of $GAut$ under V.

5. Properties of objects and G-automata

Computations of a G-automaton are, as usual, meant to be sequences of consecutive transitions. However, it is possible that some of these sequences do not agree with our principle of goal fulfilment. Hence, we have to filter undesired computations by imposing that pending goals are eventually fulfilled. And it is clear that each computation has an associated life-cycle. Let us consider fixed a G-automaton $aut = \langle \Sigma, S, \Delta, v, G, p, f \rangle$.

Proposition/Definition 23 (Computation and induced life-cycle)
A *computation* of aut is a map $\eta : I\!N \to \Delta$ such that

- $dom(\eta_{i+1}) = cod(\eta_i)$ for every $i \in I\!N$;
- for each $i \in I\!N$, if $g \in p(dom(\eta_i))$ then there exists $j \geq i$ such that $g \in f(\eta_j)$.

Denote by $Comp_{aut}$ the set of all computations of aut.

Moreover, define $Lfc_{aut}(\eta)$, the *life-cycle induced by computation* η, to be the sequence of snapshots such that:

- $Lfc_{aut}(\eta)_i = v(dom(\eta_i)) \cup \{\diamond a : a \in Menu_{aut}(dom(\eta_i))\} \cup \{\nabla a : a \in act(\eta_i)\}$ for each $i \in I\!N$. ■

Thereby, formulae satisfaction by G-automata follows naturally.

Definition 24 (Satisfaction by G-automaton)
We say that aut satisfies $\varphi \in Form_\Sigma$, and write $aut \models^{aut}_{\Sigma} \varphi$, iff $Lfc_{aut}(\eta) \models_\Sigma \varphi$ for each $\eta \in Comp_{aut}$. ■

Proposition/Definition 25 (From objects to G-automata)
There is a correspondence between G-automata and objects, given by the functor $F : GAut \to Ob$ such that $F(aut) = \langle \Sigma, Lfc_{aut}(Comp_{aut}) \rangle$ for each $aut \in |GAut|$ and $F(h : aut^1 \to aut^2) = \langle h_{obs}, h_{act} \rangle$ for each $h \in Mor(GAut)$. Moreover, F preserves products and non-local reducts. ■

We denote by ob_{aut} the object $F(aut)$ induced by aut.

Note that, since F preserves products and the product of objects denotes parallel composition, also the product of G-automata denotes parallel composition.

Regrettably, we do not get an institution over G-automata by using the cofibration technique. The problem arises exactly because of the set of goals of reducts.

Proposition 26 *Let $\sigma : \Sigma' \to \Sigma \in Mor(Sig)$, $aut \in GAut_\Sigma$ and $\varphi \in Syn(\Sigma')$. Then:*

- *$\sigma^{-1}(aut) \models^{aut}_{\Sigma'} \varphi$ implies that $aut \models^{aut}_{\Sigma} Syn(\sigma)(\varphi)$;*
- *$aut \models^{aut}_{\Sigma} Syn(\sigma)(\varphi)$ and $Loc^{\sigma}_{aut} = \emptyset$ implies that $\sigma^{-1}(aut) \models^{aut}_{\Sigma'} \varphi$.* ■

Thus, only a relativized version of the institution satisfaction condition holds. Yet, we can associate suitable G-automata to objects. It is done by computing equivalence classes of instants of life-cycles under a behavioural relation. The idea behind the construction is that a same state of the system can be associated to any two life-cycle instants with the same enablings and observations and whose possible futures are exactly the same. By doing this, transitions between states come out in a natural way.

The only problem is that undesirable behaviour may have been added. The obtained transition system may now have computations inducing unwanted life-cycles. It happens to those life-cycles such that all its finite prefixes are prefixes of some life-cycle of the starting object (e.g., one could loose fairness). Thus, to ensure that these life-cycles are not accepted, we introduce a goal corresponding to each such sequence of prefixes and fulfill it whenever the performed transition does not agree with the corresponding sequence, thus ensuring that the life-cycle is excluded. As was mentioned before, this capability was the main motivation for introducing G-automata in the first place.

Definition 27 (Concatenation of a life-cycle with a prefix)
Let Σ be a signature, $\lambda, \lambda' \in \Lambda_\Sigma$ and $n \in I\!\!N$. The *concatenation* of λ' at instant n of λ is the life-cycle $([\lambda[_n{}^\wedge\lambda')$ over Σ such that:

- $([\lambda[_n{}^\wedge\lambda')_i = \lambda_i$ for $i < n$; $([\lambda[_n{}^\wedge\lambda')_i = \lambda'_{i-n}$ for $i \geq n$. ■

Proposition/Definition 28 (Instants and behavioural equivalence)
Let $ob = \langle \Sigma, \Lambda \rangle$ be an object. Define $Inst_{ob}$, the set of life-instants of ob, to be $\{\langle \lambda, n \rangle : \lambda \in \Lambda \text{ and } n \in I\!\!N\}$. Two life-instants $\langle \lambda, i \rangle$ and $\langle \lambda', j \rangle$ of ob are behaviourally related, $\langle \lambda, i \rangle \approx \langle \lambda', j \rangle$, iff for each $\lambda'' \in \Lambda_\Sigma$ s.t. $\lambda_i \setminus \{\nabla a : a \in \Sigma_{act}\} = \lambda'_j \setminus \{\nabla a : a \in \Sigma_{act}\} = \lambda''_0 \setminus \{\nabla a : a \in \Sigma_{act}\}$, $([\lambda[_i{}^\wedge\lambda'') \in \Lambda$ iff $([\lambda'[_j{}^\wedge\lambda'') \in \Lambda$. The behavioural relation $\approx \subseteq Inst_{ob} \otimes Inst_{ob}$ defined above is an equivalence relation. ■

Denote by $[\lambda, n]$ the equivalence class of $\langle \lambda, n \rangle$ under $\approx$.

Proposition/Definition 29 (Canonical G-automaton of an object)
The canonical G-automaton of an object $ob = \langle \Sigma, \Lambda \rangle$ is $aut_{ob} = \langle \Sigma, S, \Delta, v, G, p, f \rangle$ such that:

- $S = (Inst_{ob}/_\approx)$ is the quotient set of $Inst_{ob}$ under $\approx$;
- $\Delta = \{\langle x, \emptyset, x \rangle : x \in S\} \cup \{\langle [\lambda, n], \{a : \nabla a \in \lambda_n\}, [\lambda, n+1] \rangle : \lambda \in \Lambda \text{ and } n \in I\!\!N\}$;
- $v([\lambda, n]) = \{b \in \Sigma_{obs} : b \in \lambda_n\}$ for each $[\lambda, n] \in S$;
- $G = \{(g : I\!\!N \to \Lambda) : \text{for all } i \in I\!\!N \text{ and } n \geq i, (g_n)_i = \lambda_i \text{ and } \lambda \notin \Lambda\}$;
- for each $g \in G$, $g \in p([\lambda, k])$ iff for some $i \in I\!\!N$ and $n \geq i$ $\langle g_n, i \rangle \approx \langle \lambda, k \rangle$;
- for each $g \in G$, $g \in f(\delta)$ iff $g \in p(dom(\delta))$ and $g \notin p(cod(\delta))$. ■

Unfortunately, this construction is not free due to the fact that states and goals of a G-automaton are unrelated to its signature. But still, it has some good properties. First of all, the sets of life-cycles induced by the computations of the canonical G-automaton of an object and of the object itself are exactly the same.

Proposition 30 (Object induced by canonical G-automaton)
Let $ob \in |Ob|$. Then $F(aut_{ob}) = ob$. ■

Moreover, by construction, canonical G-automata are always free of local goals.

Proposition 31 (Non-locality of canonical G-automata)
Let $ob \in |Ob_\Sigma|$ and aut_{ob} be its canonical G-automaton. Then, for every signature morphism $\sigma : \Sigma' \to \Sigma$, $Loc^{\sigma}_{aut_{ob}} = \emptyset$. ■

6. Dealing with theories

This is where we make use of the properties of the defined institution, not only to obtain the claimed denotational semantics of object specifications[20] but also to provide them an adequate operational semantics by extrapolation of properties via the functor $F : GAut \to Ob$ above defined. For now, we shall only be concerned with aggregation and hiding constructs.

We start by noting that there is a reciprocity between objects and G-automata with respect to formulae satisfaction.

Proposition 32 (Satisfaction)
Let $aut \in |GAut_\Sigma|$, $ob \in |Ob_\Sigma|$ and $\varphi \in Form_\Sigma$. Then:

- $aut \models^{aut}_{\Sigma} \varphi$ iff $F(aut) \models^{ob}_{\Sigma} \varphi$;
- $ob \models^{ob}_{\Sigma} \varphi$ iff $aut_{ob} \models^{aut}_{\Sigma} \varphi$; ■

This result allows us to refer to theories over the institution and their operational semantics without worrying about any possible inconsistency. This would not be the case, however, if the interpretation structures of the institution were simply life-cycles, or unstructured sets of life-cycles instead of objects.

So, from now on, we consider theories over $Inst$. Clearly, each theory can be given a loose semantics by associating to it all its models, either objects or G-automata.

Definition 33 (Loose semantics of theories)
Let $th = \langle \Sigma, \Gamma \rangle$ be a theory.

The *loose denotational semantics* of th is the full subcategory $Den(th)$ of Ob_Σ consisting only of those objects ob such that $ob \models^{ob}_{\Sigma} \Gamma$.

The *loose operational semantics* of th is the full subcategory $Oper(th)$ of $GAut_\Sigma$ consisting only of those G-automata aut such that $aut \models^{aut}_{\Sigma} \Gamma$. ■

Due to previous results it is easy to see that $|Den(th)| = F(|Oper(th)|)$. However, a much more accurate semantics can be given to theories. The idea is to associate to it the maximal behaviour object or G-automaton satisfying it. In the case of the denotational semantics, this is done by electing the final object of its loose semantics.

Proposition 34 (Final object model)
Let $th = \langle \Sigma, \Gamma \rangle$ be a theory. The object $[th] = \langle \Sigma, \Lambda_{th} \rangle$ with $\Lambda_{th} = (\bigcup_{\langle \Sigma, \Lambda \rangle \in |Den(th)|} \Lambda)$ is final in $Den(th)$. ■

For operational semantics, we just pick up the canonical G-automaton of its denotational semantics object.

Definition 35 (Canonical semantics of theories)
Let $th = \langle \Sigma, \Gamma \rangle$ be a theory.

The *final denotational semantics* of th is the object $[th]$.

The *final operational semantics* of th is the G-automaton $aut_{[th]}$. ■

As expected, $F(aut_{[th]}) = [th]$.

We end up by taking a closer look at two of the most used object constructs. We define their denotational and operational semantics and prove exactness of the latter with the help of the exactness results of the former[20].

Proposition 36 (Exact semantics of aggregation constructs)
Let th_1, th_2 be theories. Then, $[th_1 \oplus th_2] = [th_1] \otimes [th_2]$. Thus, $F(aut_{[th_1 \oplus th_2]}) = F(aut_{[th_1]}) \otimes F(aut_{[th_2]})$. ■

Coproducts of theories exist due to the cocompleteness of Sig.

Proposition 37 (Exact semantics of hiding constructs)
Let $th = \langle \Sigma, \Gamma \rangle$ be a theory and $\sigma : \Sigma' \to \Sigma$ be an injective (hiding) morphism. The view of th by σ is defined by the theory $\sigma^{-1}(th) = \langle \Sigma', \sigma^{-1}(\Gamma) \rangle$. Then, $[\sigma^{-1}(th)] = \sigma^{-1}[th]$. By consequence, $F(aut_{[\sigma^{-1}(th)]}) = F(\sigma^{-1}(aut_{th}))$. ■

Note that $\sigma^{-1}(\Gamma) = \{\varphi' \in Form_{\Sigma'} : \hat{\sigma}(\varphi') \in \Gamma\}$.

7. Concluding Remarks

The operational semantics of an object specification was not obtained, as it was the case of its denotational semantics, by taking a maximal final model. However, we managed to get a maximal behaviour G-automaton which is perfectly acceptable. This also applies to the exactness results obtained, despite the fact that their construction is not free. Note, however, that the set of goals is essential. Without it, we would end up with undesired computations whenever the starting object contained every arbitrary finite prefix of a certain life-cycle but not the life-cycle itself.

Unfortunately, G-automata do not even set up an institution, which hinders a much more accurate operational semantics. However, since G-automata obtained canonically from a given object were shown to be free of local goals with respect to any signature morphism, there would be no major changes if the institution existed. This happens because the absence of local goals is enough to the establishment of the corresponding satisfaction condition. However, a thorougher study of possible additions to the state and goal structure of G-automata is still necessary (for instance imposing that goals are formulae or atoms over the underlying signature, their fulfilment being derived from the observations at each state and the labels on each transition), if an institution and an adjunction between the semantic domains are to be obtained. This would yield a rigorous specification platform with a powerful institution morphism linking the two semantics. However, further object constructs and inheritance mechanisms remain to be studied.

The exact nature of the acceptance condition of G-automata must also be carefully looked at. Meanwhile, we are already able to compare it with the classical Büchi and Muller automata on infinite objects. Büchi automata are not suitable to our purposes since their acceptance condition is based on passing infinitely often in one of a number of states, which does not fit our need for stuttering (the persistent performance of the stuttering transition at an accepting state would be accepted but could, in general, be unaware of goal fulfilment). With respect to Muller automata we can say that G-automata are not less powerful i.e., that a G-automaton can be build out of a Muller automaton both accepting exactly the same computations. Muller automata acceptance condition is that the set of states visited infinitely often is one of a number of sets of states (frequent sets). A G-automaton equivalent to a given Muller automaton can be obtained by replication of its original state/transition structure through each frequent set and introduction of a persistent goal, in each of the replications, corresponding to each of its states, whose fulfilment is attained exactly by the transitions reaching that state. Moreover, the acceptance condition of G-automata is expectedly more general (or at least more intuitive) since it has to be accomplished by transitions rather than by states. At each step, each transition performed may be "helping" the computation to be accepted, which seems much more natural and feasible than ensuring that in the end a precise set of states was visited infinitely many times.

The direct synthesis of maximal behaviour finite G-automata from a given finite object specification has also been obtained[6] as an adaptation of the decision procedure for temporal logic[4,14].

8. Acknowledgements

The author is grateful to his supervisor, Amílcar Sernadas, for his indispensable help and support, and to Gunter Saake for many valuable suggestions. The comments of two anonymous referees also helped turning the presentation clearer.

9. References

1. J. Adámek, H. Herrlich and G. Strecker, *Abstract and Concrete Categories*, John Wiley, 1990.
2. H. Barringer, R. Kuiper and A. Pnueli, Now You May Compose Temporal Specifications, *Procs. of the 16th ACM Symp. on Theory of Computing*, 1984, pp.51-63.
3. M. Barr and C. Wells, *Category Theory for Computing Science*, Prentice-Hall, 1990.

4. M. Ben-Hari, A. Pnueli and Z. Manna, The Temporal Logic of Branching Time, *Acta Informatica*, **20**, Springer Verlag, 1983, pp.207-226.
5. C. Caleiro, Operational Semantics of Temporal Object Specifications, *The I-NESC Journal on Junior Activities in Science and Technology*, **1(1)**, 1994, pp.7-35.
6. C. Caleiro, G. Saake and A. Sernadas, Goal-Driven Operational Semantics of Temporal Specification of Reactive Systems Behaviour, Research Report, Secção de Ciência da Computação, Departamento de Matemática, Instituto Superior Técnico, 1994.
7. E. Clarke, O. Grumberg and R. Kurshan, A Synthesis of Two Approaches for Verifying Finite State Concurrent Systems, *Journal of Logic and Computation*, **2(5)**, Oxford University Press, 1992, pp.605-618.
8. E. Emerson, Temporal and Modal Logic, in J. van Leeuwen (ed), *Handbook of Theoretical Computer Science*, **vol 2**, Elsevier, 1990, pp.996-1072.
9. J. Goguen and R. Burstall, Introducing Institutions, in E. Clarke (ed), *Logics of Programming Workshop*, Springer Verlag, 1984, pp.221-256.
10. R. Goldblatt, *Logics of Time and Computation*, CSLI, 1987.
11. Z. Manna and A. Pnueli, Completing the Temporal Picture, *Theoretical Computer Science*, **83**, Elsevier, 1991, pp.97-130.
12. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, **vol 1**, Springer-Verlag, 1992.
13. Z. Manna and A. Pnueli, A Temporal Proof Methodology for Reactive Systems, *NATO ASI Series, Series F: Computer and System Sciences*, Springer-Verlag, 1993.
14. Z. Manna and P. Wolper, Synthesis of Communicating Processes from Temporal Logic Specifications, *ACM Transactions on Programming Languages and Systems*, **6(1)**, 1984, pp.68-93.
15. A. Pnueli, The Temporal Logic of Programs, *Procs. of the 18th IEEE Symp. on Foundations of Computer Science*, 1977, pp.46-57.
16. A. Sernadas, J. F. Costa and C. Sernadas, An Institution of Object Behaviour, *Proceedings of the 9th ADT/COMPASS Workshop*, 1992.
17. A. Sernadas and H.-D. Ehrich, What is an Object, After All?, in R. Meersman, W. Kent and S. Khosla (eds), *Object Oriented Databases: Analysis, Design and Construction*, North-Holland, 1991, pp.39-69.
18. A. Sernadas, J. L. Fiadeiro, C. Sernadas and H.-D. Ehrich, Basic Building Blocks of Information Systems, in E. Falkenberg and P. Lindgreen (eds), *Information Systems Concepts: An In-depth Analysis*, North-Holland, 1989, pp.225-246.
19. A. Sernadas, J. L. Fiadeiro, C. Sernadas and H.-D. Ehrich, Abstract Object Types: A Temporal Perspective, in B. Banieqbal, H. Barringer and A. Pnueli (eds), *Temporal Logic in Specification*, Springer Verlag, 1989, pp.324-350.
20. A. Sernadas and C. Sernadas, Denotational Semantics of Temporal Object Specification Within an Arbitrary Temporal Logic Institution, Research Report, Secção de Ciência da Computação, Departamento de Matemática, Insti-

tuto Superior Técnico, 1993. Presented in ISCORE Workshop 93 - Submitted for publication.

21. A. Sernadas and C. Sernadas, Object Certification, Research Report, Secção de Ciência da Computação, Departamento de Matemática, Instituto Superior Técnico, 1994. Presented in ISCORE Workshop 94 - Submitted for publication.
22. A. Sernadas, C. Sernadas and J. F. Costa, Object Specification Logic, *Journal of Logic and Computation*, Oxford University Press, 1995. In print.
23. A. Tarlecki, R. Burstall and J. Goguen, Some Fundamental Algebraic Tools for the Semantics of Computation. Part III: Indexed Categories, *Theoretical Computer Science*, Elsevier, 1991, pp.239-264.
24. W. Thomas, Automata on Infinite Objects, in J. van Leeuwen (ed), *Handbook of Theoretical Computer Science*, **vol 2**, Elsevier, 1990, pp.133-191.
25. G. Winskel, Synchronization Trees, *Theoretical Computer Science*, **34**, Elsevier, 1984.