Efficient Operational Semantics for EB^3 for Verification with CADP

Dimitris Vekris and Catalin Dima

LACL, Université Paris-Est
61 av. du Général de Gaulle
94400 Créteil, France
{dimitrios.vekris,dima}@u-pec.fr

Abstract. EB^3 is a specification language tailored for information systems. The core of the EB^3 language consists of process algebraic specifications describing the behaviour of the entity types in a system, and attribute function definitions describing the entity attribute types. The verification of EB^3 specifications against temporal properties is of great interest to users of EB^3.
We give here an operational semantics for EB^3 programs in which attribute functions are computed during program evolution and their values are stored into program memory. By assuming that all entities have finite domains, this gives a finitary operational semantics which can be better handled by model-checking tools. We then demonstrate how this new semantics facilitates the translation of EB^3 specifications to LOTOS NT (LNT for short) for verification with the use of the CADP toolbox.

Keywords: Information Systems, EB^3, Process Algebras, Operational Semantics, Bisimulation, Verification, Model Checking

1 Introduction

The EB^3 [10] method is an event-based paradigm tailored for information systems (ISs). A typical EB^3 specification defines entities, associations, and their respective attributes. The process algebraic nature of EB^3 permits the explicit definition of intra-entity constraints. Yet its specificity against common state-space specifications, such as the B method [1] and Z, lies in the use of attribute functions, a special kind of recursive functions on the system trace, which combined with guards, facilitate the definition of complex inter-entity constraints involving the history of events. The use of attribute functions is claimed to simplify system understanding, enhance code modularity and streamline maintenance.

In this paper, we present part of our work regarding the verification of EB^3, i.e. the detection of errors inherent in EB^3 specifications. Specification errors in EB^3 can be detected with the aid of invariants also known as static properties or temporal properties known as dynamic properties. From a state-based point of view, an invariant describes a property on state variables that must be preserved by each transition or event. A dynamic property relates several events. Tools such as Atelier B [7] provide methodologies on how to define and prove invariants. In [12], an automatic translation of EB^3's attribute functions with B is attempted. Although the B Method [1] is suitable for specifying static properties, dynamic properties are very difficult to express and verify in B. Hence, in our attempt to verify dynamic properties of EB^3 specifications we move our attention to model-checking techniques.

The verification of EB^3 specifications against temporal properties with the use of model checking has been the subject of some work in the recent years. [9] compares six model checkers for the verification of IS case studies. The specifications used in [9] derive from industrial case studies, but the prospect of a uniform translation from EB^3 program specifications is not studied. [6] summarizes the fundamental difficulties for designing a compiler that would translate a given EB^3 specification to process algebra LOTOS NT (LNT for short) [5], an approach that would make use of the verification suite CADP (*Construction and Analysis of Distributed Processes*) [11] to verify properties against the source system. In short, the majority of these works treat specific case studies drawn from the information systems domain leading to ad-hoc verification translations, but nonetheless lacking in generalization capability.

In the absence of model-checking tools for the verification of EB^3 specifications, translating EB^3 to LNT is a reasonable approach. LNT is a process algebra specification that derived from LOTOS [4]. As a process algebra, it shares many common features with EB^3 and it is one of the input languages of CADP, a toolbox with state-of-the-art verification features. CADP permits the verification of system specifications against action-based temporal properties.

Translating EB^3 specifications to LNT is not evident, because LNT does not feature global variables. In addition, accesses to local variables is restricted in parallel processes of the form "**par** $proc_1$ || $proc_2$ **end par**", so that every variable written in $proc_1$ cannot be accessed in $proc_2$. Although, EB^3 programmers cannot define global variables explicitly, EB^3 permits the use of a single state variable, the system trace, in predicates of guard statements. Attribute functions can express the evolution of entity attributes in time, option which introduces an indirect notion of state to the language. As a result, EB^3 expressions of the form "$p \Rightarrow E$" can be written, where p is a predicate that refers to the system trace (the history of events) and E is a valid EB^3 expression.

We propose a formal semantics for EB^3 that treats attribute functions as state variables (we call these variables *attribute variables*). This semantics will serve as the basis for applying a simulation strategy of state variables in LNT. Intuitively, coding attribute functions as part of the system state is beneficial from a model-checking point of view as the new formalisation dispenses with the system trace. Our main contribution is an operational semantics in which attribute functions are computed during program evolution and stored into program memory. We show that this operational semantics is bisimilar with the original, trace-based operational semantics, in the sense that, for each EB^3 specification, the transition system corresponding with its memory-based semantics is bisimilar with the transition system corresponding with its trace-based semantics. We then present how EB^3 specifications can be translated to LNT for verification with CADP through an intuitive example and give some conclusions and lines for future work.

2 EB^3

The EB^3 method has been specially designed to specify the functional behaviour of ISs. A standard EB^3 specification comprises:

1. a class diagram representing entity types and associations for the IS being specified;
2. a process algebra specification, denoted by *main*, describing the IS, i.e. the valid traces of execution describing its behaviour;
3. a set of attribute function definitions, which are recursive functions on the system trace; and
4. input/output rules, to specify outputs for input traces, or SQL used to specify queries on the business model

We limit the presentation to the process algebra and the set of attribute functions used in the IS. By means of a simple case study, we report on the expressive power of EB^3 and namely the use of attribute functions to express complex inter-process constraints. We then give three operational semantics for EB^3. The first, named Trace Semantics (Sem_T), is the standard semantics defined in [10]. The second, called Trace/Memory Semantics ($Sem_{T/M}$), is the alternative semantics, where attribute functions are computed during program evolution and their values are stored into program memory. By removing the trace from each state in $Sem_{T/M}$, we obtain the third semantics for EB^3 specifications, which we name Memory Semantics, Sem_M.
Case Study. In Fig.1, we give the functional requirements of a library management system and the corresponding EB^3 specification. Function *main* is the parallel interleaving between m instances of process *book* and p instances of process *member*. Process *book* stands for a book acquisition followed by its eventual discard. The attribute function *borrower*(T, bId), where T is the current trace, returns the current borrower of book bId or $\perp$ if the book is not lent, by looking for events of the form *Lend*(mId, bId) or *Return*(bId) in the trace. In process book, action *Discard*(bId) is thus guarded to guarantee that book bId cannot be discarded if it is currently lent.

The use of attribute functions is not adherent to standard process algebra practices as it may naively trigger the complete traversal and inspection of the system trace. Alternatively, one may come up with simpler specifications based solely on process algebra operations (without attribute functions) when the functional requirements imply loose interdependence between entities and associations. For instance, if all books are acquired by the library before any other event occurs and are eventually discarded (given that there are no more demands), *main*'s code can be modified in the following manner:

$$\boxed{\begin{aligned} main = {} & (\ |||\ bId : \mathrm{BID} : Acquire(bId)\ ).\ (\ |||\ mId : \mathrm{MID} : member(mId)*\ ).\\ & (\ |||\ bId : \mathrm{BID} : Discard(bId)\ ) \end{aligned}}$$

1. A book can be acquired by the library. It can be discarded, but only if it has not been lent.
2. An individual must join the library in order to borrow a book.
3. A member can relinquish library membership only when all his loans have been returned.
4. A member cannot borrow more than the loan limit defined at the system level for all users.

$$
\begin{aligned}
BID &= \{b_1, \ldots, b_m\},\ MID = \{m_1, \ldots, m_p\} \\
main &= (\ |||\ bId : \text{BID} : book(bId)\ )\ |||\ (\ |||\ mId : \text{MID} : member(mId)*\ ) \\
book(bId : \text{BID}) &= Acquire(bId).\ borrower(T,\ bId) = \bot \Rightarrow Discard(bId) \\
member(mId : \text{MID}) &= Register(mId).\ (\ |||\ bId : \text{BID} : loan(mId, bId)*\ ).\ Unregister(mId) \\
loan(mId : \text{MID},\ bId : \text{BID}) &= borrower(T,\ bId) = \bot \wedge nbLoans(T, mId) < \text{NbLoans} \\
&\quad \Rightarrow Lend(bId,\ mId).\ Return(bId)
\end{aligned}
$$

$nbLoans(T\text{: } tr,\ mId\text{: MID})\text{: } Nat_\bot =$	$borrower(T\text{: } tr,\ bId\text{: BID})\text{: MID} =$
match T **with**	**match** T **with**
$[\,] \to \bot$	$[\,] \to \bot$
$\mid T'.\ Lend(bId, mId) \to nbLoans(T', mId) + 1$	$\mid T'.\ Lend(bId, mId) \to mId$
$\mid T'.\ Register(mId) \to 0$	$\mid T'.\ Return(bId) \to \bot$
$\mid T'.\ \text{Unregister(mId)} \to \bot$	$\mid \_ \to borrower(T', bId)$
$\mid T'.\ Return(bId) \wedge mId = borrower(T, bId)$	
$\to nbLoans(T', mId) - 1$	
$\mid \_ \to nbLoans(T', mId)$	

Fig. 1. EB^3 Specification and Attribute Function Definitions

Note that the functional requirements are not contradicted, though the system's behaviour changes dramatically. Programming naturally in a purely process-algebraic style without attribute functions in EB^3 may not always be obvious. In some cases, ordering constraints involving several entities are quite difficult to express without guards and lead to less readable specifications than equivalent guard-oriented solutions in EB^3 style. For instance, writing the specification without the use of the guard:

$$\boxed{borrower(T,\ bId) = \bot \wedge nbLoans(T, mId) < \text{NbLoans}},$$

that illustrates the conditions under which a *Lend* can occur (notably when the book is available and *nbLoans* is less than the fixed bound *NbLoans*), is not trivial.

$\mathbf{EB^3}$ Syntax and $\mathbf{Sem_T}$. We proceed with the formal definition of EB^3. We define a set of attribute function names $AtFct = \{f_1, \ldots, f_n\}$ and a set of process function names $PFct = \{P_1, \ldots, P_m\}$. Let $\rho \in Act$ stand for an action of either form $\alpha(p_1 : T_1, \ldots, p_n : T_n)$, where $\alpha \in lab$ [1] is the *label* of the action and $p_i, i \in 1..n$ are elements of type T_i, or λ, which stands for the internal action. To simplify the presentation, we assume that every attribute functions f_i have the same formal parameters $\overline{x}$. An EB^3 specification is a set of attribute function definitions, *AtF* and a set of process definitions, *ListPE*.

The trace semantics of EB^3, Sem_T [10], are given in Fig.2 as a set of rules named $R_T - 1$ to $R_T - 11$. Each state is represented as a tuple (E, T), where E stands for an EB^3 expression and T for the current trace. An action ρ is the simplest EB^3 process, whose semantics are given by rules $R_T\!-\!1, 1'$. Note that λ is not visible in the EB^3 execution trace, i.e., it does not impact the definition of attribute functions. The symbol $\surd$ denotes successful execution. EB^3 processes

[1] we assume $lab = \{\alpha_1, \ldots, \alpha_q\}$

$$EB^3 ::= AttrF \ ; \ ListPE$$

$$ListPE ::= P_l(\overline{x_l}) = E \ \Big| \ P_l(\overline{x_l}) = E \ ; \ ListPE, \ l \in 1..m$$

$$AtF ::= AtFDef \ \Big| \ AtFDef \ ; \ AtF$$

$$AtFDef ::= f_i(T, \overline{x}) = \begin{cases} exp_i^0 & \text{if } T = [\,] \\ \bigvee_{j=1}^{q} \bigvee_{k=1}^{m_j} hd(T) = \alpha_j(\overline{x_j}) \wedge cond_i^{j,k} \Rightarrow exp_i^{j,k} & \text{otherwise} \end{cases}, \ i \in 1..n$$

$$E ::= \surd \ \Big| \ \lambda \ \Big| \ \alpha(\overline{v}) \ \Big| \ E.E \ \Big| \ E|E \ \Big| \ E^* \ \Big| \ E|[\Delta]|E \ \Big| \ |x:V:E \ \Big| \ |[\Delta]|x:V:E \ \Big| \ GE \Rightarrow E \ \Big| \ P(\overline{t})$$

$$R_T-1: \frac{}{(\rho, T) \xrightarrow{\rho} (\surd, T\cdot\rho)} \rho \neq \lambda \qquad R_T-1': \frac{}{(\lambda, T) \xrightarrow{\lambda} (\surd, T)}$$

$$R_T-2: \frac{(E_1, T) \xrightarrow{\rho} (E_1', T')}{(E_1.E_2, T) \xrightarrow{\rho} (E_1'.E_2, T')} \qquad R_T-3: \frac{(E, T) \xrightarrow{\rho} (E', T')}{(\surd.E, T) \xrightarrow{\rho} (E', T')}$$

$$R_T-4: \frac{(E_1, T) \xrightarrow{\rho} (E_1', T')}{(E_1|E_2, T) \xrightarrow{\rho} (E_1', T')} \qquad R_T-5: \frac{}{(E^*, T) \xrightarrow{\lambda} (\surd, T)}$$

$$R_T-6: \frac{(E, T) \xrightarrow{\rho} (E', T')}{(E^*, T) \xrightarrow{\rho} (E'.E^*, T')} \qquad R_T-7: \frac{}{(\surd|[\Delta]|\surd, T) \xrightarrow{\lambda} \surd, T)}$$

$$R_T-8: \frac{(E_1, T) \xrightarrow{\rho} (E_1', T'), \ (E_2, T) \xrightarrow{\rho} (E_2', T')}{(E_1|[\Delta]|E_2, T) \xrightarrow{\rho} (E_1'|[\Delta]|E_2', T')} in(\rho, \Delta)$$

$$R_T-9: \frac{(E_1, T) \xrightarrow{\rho} (E_1', T')}{(E_1|[\Delta]|E_2, T) \xrightarrow{\rho} (E_1'|[\Delta]|E_2, T')} \neg in(\rho, \Delta)$$

$$R_T-10: \frac{(E, T) \xrightarrow{\rho} (E', T')}{(GE \Rightarrow E, T) \xrightarrow{\rho} (E', T')} \ \|GE\|$$

$$R_T-11: \frac{(E[\overline{x} := \overline{t}], T) \xrightarrow{\rho} (E', T')}{(P(\overline{t}), T) \xrightarrow{\rho} (E', T')} P(\overline{x}) = E \in ListPE$$

Fig. 2. EB^3 Syntax and Sem_T

can be combined with classical process algebra operators such as the *sequence* ($R_T-2,3$), the *choice* (R_T-4) and the *Kleene* Closure ($R_T-5,6$) operators. Rules ($R_T-7,8,9$) refer to the *parallel* composition $E_1|[\Delta]|E_2$ of E_1, E_2 with synchronization on $\Delta \subseteq lab$. The condition $in(\rho, \Delta)$ is true, iff the label of ρ belongs to Δ. The symmetric rules for *choice* and *parallel* composition have been omitted. Expression $E_1|||E_2$ is equivalent to $E_1|[\emptyset]|E_2$ and $E_1||E_2$ to $E_1|[lab]|E_2$.

In R_T–10, the *guarded expression* process $GE \Rightarrow E$ can execute E if the predicate GE holds. GE contains calls to attribute functions, i.e. functions defined on the system trace. Concretely, the truth value of GE depends on the executed actions. Quantification is permitted for *choice* and *parallel* composition. If V is a set of attributes $\{t_1, \ldots, t_n\}$, $|x:V:E$ and $|[\Delta]|x:V:E$ stand respectively for $E[x:=t_1]| \ldots |E[x:=t_n]$ and $E[x:=t_1]|[\Delta]| \ldots |[\Delta]|E[x:=t_n]$, where $E[x:=t]$ denotes the replacement of all occurrences of x by t. For instance, $||x:\{1,2,3\}:a(x)$ stands for $a(1)||a(2)||a(3)$. By convention,$|x:\emptyset:E = |[\Delta]|x:\emptyset:E = \surd$.

We stipulate that attribute functions are defined as in *AtFDef* [2] in Fig.2, where $exp_i^{j,k}$ are expressions, $cond_i^{j,k}$ are boolean expressions, $hd(T)$ denotes the last element of the trace, and $tl(T)$ denotes the trace without its last element. Expressions can be constructed from objects and operations of user-defined domains, such as integers, booleans and more complex domains that we do not give formally. We also assume that for each $1 \leq i \leq n$, every calls to an attribute function f_j occurring in $exp_i^{j,k}$ or $cond_i^{j,k}$ are parameterized by T if $l \leq i$ or by $tl(T)$ if $l > i$. Such an ordering can be constructed if the EB^3 specification does not contain circular dependencies between function calls, which would lead to infinite attribute function evaluation. This restriction on *AtFct* is satisfied in Fig.1 as both *nbLoans* and *borrower* contain calls to *nbLoans* and *borrower* parameterized on $tl(T)$. Also, *nbLoans* makes call to *borrower* parameterized on T. Hence, $f_1 = borrower$ and $f_2 = nbLoans$.

Sem$_{\mathbf{T/M}}$. $Sem_{T/M}$ is given in Fig.3 as a set of rules named $T_{T/M}-1$ to $T_{T/M}-11$. Each state is represented as a tuple (E, T, M). $M_i(\overline{x})$ is the variable that keeps the current valuation for attribute function f_i with parameter vector $\overline{x}$. M_i refers to attribute function f_i. Given that the EB^3 specification is valid, there is at least one $cond_i^{j,k}$ that is evaluated true on every run. The event ρ_j to occur "chooses" the corresponding $cond_i^{j,k}$ non-deterministically (in the sense that there may be many k that make $cond_i^{j,k}$ evaluate to true). Function *next* updates M_i by making use of M_l for $l \geq i$ and the freshly computed $next(M_l)(\rho_j)$ for $l < i$. GE contains calls to attribute functions parameterized on T only. The classic intepretations for Peano arithmetics, set theory and boolean logic suffice to evaluate them. In $(T_{T/M}-10)$, $GE[f_i \leftarrow M_i]$ denotes replacing all calls to f_i in GE by M_i.

Sem$_{\mathbf{M}}$. Sem_M [3] derives from $Sem_{T/M}$ by simple elimination of T from each tuple (E, T, M) in rules $T_{T/M}-1$ upto $T_{T/M}-11$. Intuitively, this means that the information on the history of executions is kept in M, thus rendering the presence of trace T redundant.

Case Study Revisited. We show how the EB^3 specification above is evaluated w.r.t. Sem_T and Sem_M. Provided that *BID*=$\{b_1, b_2\}$, *MID*=$\{m_1, m_2\}$, we take state variables *M=(bor[bId1], bor[bId2], nbL[mId1], nbL[mId2])*, where *bor* stands for *borrower* and *nbL* for *nbLoans*, respectively. More formally, if $f(T, a_1 : T_1, \ldots, a_r : T_r)$ is an attribute function, we construct $|T_1| \times \ldots \times |T_r|$ state variables, where $|T_i|, i \in 1..r$ stands for T_i's cardinality. Intuitively, coding attribute functions as part of the system state is beneficial from a model-checking point of view as it avoids keeping (potentially huge) trace in memory.

We set *NbLoans=2*. Fig.4 shows how *main* is modified for the valid trace: T_D *=Lend(bId1,mId1).Reg(mId2).Reg(mId1).Acq(bId2).Acq(bId1)* [4].

[2] this notation is different from the standard pattern-matching notation for attribute functions [10], but still more compact

[3] Sem_M can be found in the appendix for referee's eyes only

[4] *Acq* stands for *Acquire* and *Reg* for *Register*, respectively

$$M_i^0(\overline{x}) = \|exp_i^0(\overline{x})\|$$
$$next(M_i)(\rho_j)(\overline{x}) = \|exp_i^{j,k}(\overline{x})[f_l \leftarrow \text{if } l < i \text{ then } next(M_l)(\rho_j) \text{ else } M_l]\|,$$
$$\text{if } \|cond_i^{j,k}(\overline{x})[f_l \leftarrow \text{if } l < i \text{ then } next(M_l) \text{ else } M_l]\|,\ i \in 1..n\ 1,\ k \in 1..m_j$$

$$T_{T/M}{-}1: \frac{\rho \neq \lambda}{(\rho, T, M) \xrightarrow{\rho} (\surd, T\cdot\rho, next(M)(\rho))} \quad T_{T/M}{-}1': \frac{}{(\lambda, T, M) \xrightarrow{\lambda} (\surd, T, M)}$$

$$T_{T/M}{-}2: \frac{(E_1, T, M) \xrightarrow{\rho} (E_1', T', M')}{(E_1.E_2, T, M) \xrightarrow{\rho} (E_1'.E_2, T', M')} \quad T_{T/M}{-}3: \frac{(E, T, M) \xrightarrow{\rho} (E', T', M')}{(\surd.E, T, M) \xrightarrow{\rho} (E', T', M')}$$

$$T_{T/M}{-}4: \frac{(E_1, T, M) \xrightarrow{\rho} (E_1', T', M')}{(E_1|E_2, T, M) \xrightarrow{\rho} (E_1', T', M')} \quad T_{T/M}{-}5: \frac{}{(E^*, T, M) \xrightarrow{\lambda} (\surd, T, M)}$$

$$T_{T/M}{-}6: \frac{(E, T, M) \xrightarrow{\rho} (E', T', M')}{(E^*, T, M) \xrightarrow{\rho} (E'.E^*, T', M')} \quad T_{T/M}{-}7: \frac{}{(\surd|[\Delta]|\surd, T, M) \xrightarrow{\lambda} \surd, T, M)}$$

$$T_{T/M}{-}8: \frac{(E_1, T, M) \xrightarrow{\rho} (E_1', T', M'),\ (E_2, T, M) \xrightarrow{\rho} (E_2', T', M')}{(E_1|[\Delta]|E_2, T, M) \xrightarrow{\rho} (E_1'|[\Delta]|E_2', T', M')} in(\rho, \Delta)$$

$$T_{T/M}{-}9: \frac{(E_1, T, M) \xrightarrow{\rho} (E_1', T', M')}{(E_1|[\Delta]|E_2, T, M) \xrightarrow{\rho} (E_1'|[\Delta]|E_2, T', M')} \neg in(\rho, \Delta)$$

$$T_{T/M}{-}10: \frac{(E, T, M) \xrightarrow{\rho} (E', T', M')}{(GE \Rightarrow E, T, M) \xrightarrow{\rho} (E', T', M')}\ \|GE[f_i \leftarrow M_i]\|$$

$$T_{T/M}{-}11: \frac{(E[\overline{x} := \overline{t}], T, M) \xrightarrow{\rho} (E', T', M')}{(P(\overline{t}), T, M) \xrightarrow{\rho} (E', T', M')} P(\overline{x}) = E \in ListPE$$

Fig. 3. $Sem_{T/M}$

The intermediate states A, B, C and D for Sem_T and Sem_M are given in Fig. 5. The first column corresponding to Sem_T keeps track of the system trace, whereas Sem_M gives a finite state system since the domains of its attribute functions are finite and bounded. All variables are equal to $\perp$ [5] for $T = [\ ]$. Focussing on transition $C \rightarrow D$, in order to check $borrower(T, bId1) = \perp \wedge nbLoans(T, mId1) < 2$, Sem_T evaluates $borrower(T, bId1)$ and $nbLoans(T, mId1)$ by traversing the trace and applying their corresponding attribute function formulas. On the contrary, Sem_M evaluates M based solely on the current memory and the event to occur i.e. $Lend(bId1, mId1)$. We get $bor_D[bId1] = next(bor_C[bId1])(Lend(bId1, mId1)) = mId1$ [6], $bor_D[bId2] = bor_C[bId2] = \perp$,

[5] see *borrower*'s and *nbLoans*'s script for $T = [\ ]$ in Fig.2

[6] see *borrower*'s script for $T = T'.Lend(bId, mId)$ in Fig.2

$main$ (A)

$\xrightarrow{Acq(bId2).Acq(bId1)}$

$borrower(T,\ bId1) = \bot \rightarrow Discard(bId1)\ |||$
$borrower(T,\ bId2) = \bot \rightarrow Discard(bId2)\ |||$
$(\ |||\ mId : \mathrm{MID} : member(mId)*\ )$ (B)

$\xrightarrow{Reg(mId2).Reg(mId1)}$

$borrower(T,\ bId1) = \bot \rightarrow Discard(bId1)\ |||$
$borrower(T,\ bId2) = \bot \rightarrow Discard(bId2)\ |||$
$(\ |||\ bId : \mathrm{BID} : loan(mId1, bId)*\ ).\ Unregister(mId1).\ member(mId1)*\ |||$
$(\ |||\ bId : \mathrm{BID} : loan(mId2, bId)*\ ).\ Unregister(mId1).\ member(mId2)*$ (C)

$\xrightarrow{Lend(bId1,\ mId1)}$

$borrower(T,\ bId1) = \bot \rightarrow Discard(bId1)\ |||$
$borrower(T,\ bId2) = \bot \rightarrow Discard(bId2)\ |||$
$(Return(bId1).\ loan(mId1, bId1)*\ |||\ loan(mId1, bId2)*).Unregister(mId1).\ member(mId1)*\ |||$
$(\ |||\ bId : \mathrm{BID} : loan(mId2, bId)*\ ).\ Unregister(mId1).\ member(mId2)*$ (D)

Fig. 4. Execution

	T	$M = (bor[bId1], bor[bId2], nbL[mId1], nbL[mId2])$
A	[ ]	$(\bot, \bot, \bot, \bot)$
B	$Acq(bId2).Acq(bId1)$	$(\bot, \bot, \bot, \bot)$
C	$T_B.Reg(mId2).Reg(mId1)$	$(\bot, \bot, 0, 0)$
D	$T_C.Lend(bId1,\ mId1)$	$(mId1, \bot, 1, 0)$

Fig. 5. States

$nbL_D[mId1] = nbL_C[mId1] + 1 = 1$ and also $nbL_D[mId2] = 0$, if the event *Lend(bId1,mId1)* is to be executed.

3 Bisimulation Equivalence of Sem_T, $Sem_{T/M}$ and Sem_M

We present the theoretical proof of the bisimulation equivalence for Sem_T, $Sem_{T/M}$ and Sem_M.
LTSs. We consider finite labeled transition systems (LTSs) as interpretation models, which are particularly suitable for action-based description formalisms such as EB^3. Formally, an LTS is a triple $(S, \{\rightarrow^a\}_{a\in Act}, I)$, where:

1. S is a set of states;
2. $\rightarrow^a \subseteq S \times S$, for all $a \in Act$;
3. $I \subseteq$ S is a set of initial states.

Bisimulation. Bisimulation is a fundamental notion in the framework of concurrent processes and transition systems. A system is bisimilar to another system if the former can mimic the behaviour of the latter and vice-versa. In this sense, the associated systems are considered indistinguishable. Given two LTSs

$TS_i = (S_i, \{\rightarrow^a\}_{a \in Act}, I_i)$, where $i = 1, 2$ and a relation $R \subseteq S_1 \times S_2$, systems TS_i are said to be equivalent w.r.t. bisimulation iff

1. $\forall\, s_1 \in I_1\ \exists s_2 \in I_2$ such that $(s_1, s_2) \in R$
2. $\forall\, s_2 \in I_2\ \exists s_1 \in I_1$ such that $(s_1, s_2) \in R$
3. $\forall (s_1, s_2) \in R :$
   (a) if $s_1 \rightarrow^a s_1'$ then $\exists s_2' \in S_2$ such that $s_2 \rightarrow^a s_2'$ and $(s_1', s_2') \in R$
   (b) if $s_2 \rightarrow^a s_2'$ then $\exists s_1' \in S_1$ such that $s_1 \rightarrow^a s_1'$ and $(s_1', s_2') \in R$

LTS Construction. For given EB^3 process E, we need to construct the corresponding LTSs w.r.t. Sem_T, $Sem_{T/M}$ and Sem_M respectively. The construction is given by structural induction on E. We show here how to construct:

$$TS_E = (S_E, \delta_E, I_E)$$

w.r.t. Sem_M only. We refer to the initial memory as $M^0 \in \mathcal{M}$ ($\mathcal{M}$ is the set of memory mappings in the IS) defined upon the fixed body of attribute function definitions. It is $I_E = \{(E, M^0)\}$. More precisely,

1. $S_{\surd} = \left\{(\surd, M^0)\right\}, \delta_{\surd} = \emptyset$
2. $S_{\rho} = \left\{(\rho, M^0)\right\} \bigcup \left\{(\surd, next(M^0))\right\},\ \delta_{\rho} = \left\{(\rho, M^0) \rightarrow^{\rho} \left(\surd, next(M^0)\right)\right\}$, where $\rho \neq \lambda$
3. $S_{\lambda} = \left\{(\lambda, M^0)\right\} \bigcup \left\{(\surd, M^0)\right\},\ \delta_{\lambda} = \left\{(\lambda, M^0) \rightarrow^{\rho} \left(\surd, M^0\right)\right\}$
4. $\begin{cases} S_{E_1.E_2} = \left\{(E_1'.E_2, M) \mid (E_1', M) \in S_{E_1}\right\} \bigcup \\ \quad \bigcup_{(\surd, M_1) \in S_{E_1}} \left\{(E_2', M) \mid (E_2', M) \in S_{E_2}^{M_1}\right\}, \\ \delta_{E_1.E_2} = \left\{(E_1'.E_2, M) \rightarrow (E_1''.E_2, M') \mid (E_1', M) \rightarrow^{\rho} (E_1'', M') \in \delta_{E_1}\right\} \bigcup \\ \quad \bigcup_{(\surd, M_1) \in S_{E_1}} \{(E_2', M) \rightarrow^{\rho} (E_2'', M') \mid (E_2', M) \rightarrow^{\rho} (E_2'', M') \in \delta_{E_2}^{M_1}\} \end{cases}$
5. $\begin{cases} S_{E_1|E_2} = S_{E_1} \setminus \{(E_1, M^0)\} \bigcup S_{E_2} \setminus \{(E_2, M^0)\} \bigcup \left\{(E_1|E_2, M^0)\right\}, \\ \delta_{E_1|E_2} = \delta_{E_1}\, [E_1 \leftarrow E_1|E_2] \bigcup \delta_{E_2} [E_2 \leftarrow E_1|E_2] \end{cases}$
6. $TS_{E^*} = lfp_F$, where $F(TS_{E_x}) = TS_{E\,.\,E_x} \cup TS_{\lambda}, \quad E^* \doteq E.E^*|\lambda$
7. $TS_{E_1|[\mathit{\Delta}]|E_2} = TS_{\sum_{i=1}^{r} GE^i \Rightarrow \rho^i(\overline{a}^i).E^i}$,

   where $E_1|[\mathit{\Delta}]|E_2 \doteq \displaystyle\sum_{i=1}^{r} GE^i \Rightarrow \rho^i(\overline{a}^i).E^i$
8. $\begin{cases} S_{GE \Rightarrow E} = \begin{cases} \{(GE \rightarrow E, M^0)\} \bigcup S_E \setminus \{(E, M^0)\}, & \text{if } \|GE[f_i \leftarrow M_i^0]\| \\ \{(GE \rightarrow E, M^0)\}, & \text{otherwise} \end{cases} \\ \delta_{GE \Rightarrow E} = \begin{cases} \delta_E[(E, M^0)\ \leftarrow\ (GE \Rightarrow E, M^0)], & \text{if } \|GE[f_i \leftarrow M_i^0]\| \\ \emptyset, & \text{otherwise} \end{cases} \end{cases}$
9. $TS_{P(\bar{t})} = TS_{E[\overline{x} := \bar{t}]}$, where $P(\overline{x}) \doteq E$

The difference between 2 and 3 lies in the fact that λ does not affect the memory. In 4, it is $(E_1'.E_2, M) \in S_{E_1 \cdot E_2}$ if $(E_1', M) \in S_{E_1}$. For $(\surd, M_1) \in S_{E_1}$, we obtain $(E_2', M) \in S_{E_2}^{M_1}$, where $S_{E_2}^{M_1}$ stands for state space S_{E_2} with initial memory M_1. In 6, we need to compute the least fix point of function $F : TS \to TS$ w.r.t. the *lattice* $\mathcal{TS} = (TS, \subseteq)$, where TS is the possibly infinite set of LTSs simulating EB^3 specifications w.r.t. $Sem_{T/M}$ and $\subseteq$ denotes inclusion. In 7, $E_1|[\Delta]|E_2$ is written as a sum ($\sum \doteq [\,] \ldots [\,]$) of EB^3 expressions. The first action of each summand would be $\rho^i(\overline{a}^i)$ for all possible execution paths picking this summand. This action would be taken under condition GE^i (=true in the absence of condition):

$$\boxed{E_1|[\Delta]|E_2 \doteq \sum_{i=1}^{r} GE^i \Rightarrow \rho^i(\overline{a}^i).E^i}$$

This form is known as head normal form (HNF) in the literature. The construction of HNFs for process algebra expressions is discussed in [2]. It is a common practice developed principally in the context of the Algebra of Communicating Processes (ACP) [3] as a means to analyse the behaviour of recursive process algebra definitions. In 8 for $\|GE[f_i \leftarrow M_i^0]\|$ true, we need to construct S_E and replace (E, M^0) with $(GE \Rightarrow E, M^0)$.

For the rest, we denote TS_T, $TS_{T/M}$ and TS_M for TS_E w.r.t. Sem_T, $Sem_{T/M}$ and Sem_M respectively.

Theorem 1. *TS_T and $TS_{T/M}$ are bisimilar.*

Proof. We consider relation $R = \{\langle (E,T,M),(E,T)\rangle \mid (E,T,M) \in S_{T/M} \wedge (E,T) \in S_T\}$. It is $\langle (E^0,[\,],M^0),(E^0,[\,])\rangle \in R$. We demonstrate that for any $\langle (E,T,M),(E,T)\rangle \in R$ and $(E,T,M) \to^{\rho} (E',T',M') \in \delta_{T/M}$, we obtain $(E,T) \to^{\rho} (E',T') \in \delta_T$ and vice-versa. We proceed with structural induction on E. We show the proof for some cases.

For $(T_{T/M}-1)$, we get $(\rho, T, M) \to^{\rho} (\surd, T \cdot \rho, next(M)(\rho)) \in \delta_{T/M}$. By elimination of M and $next(M)$, we get that $(\rho, T) \to^{\rho} (\surd, T \cdot \rho) \in \delta_T$. Inversely, we imagine $(\rho, T) \to^{\rho} (\surd, T \cdot \rho) \in \delta_T$. Every state $(E, T, M) \in S_{T/M}$ is of the form:

$$\boxed{\begin{array}{c} (E, T, next'(T, M^0)), \text{ where} \\ next'(T, M) = \textbf{match } T \textbf{ with } [\,] \to\ M \mid T \cdot \rho \to next'(T', next(M, \rho)) \end{array}},$$

As a result, there exists $(\rho, T, next'(T, M^0)) \to^{\rho} (\surd, T \cdot \rho, next'\left(T \cdot \rho, M^0\right)) \in \delta_{T/M}$ which establishes $(T_{T/M}–1)$ by replacing $next'(T, M^0)$ with M and $next'(T \cdot \rho, M^0)$ with $next\,(M)\,(\rho)$.

For $(T_{T/M}-2)$, we have $(E_1.E_2, T, M) \to^{\rho} (E_1'.E_2, T', M') \in \delta_{T/M}$, that assumes $(E_1, T, M) \to^{\rho} (E_1', T', M') \in \delta_{T/M}$. By induction hypothesis, $(E_1, T) \to^{\rho} (E_1', T') \in Sem_T$ and by $(R_T - 2)$, we get $(E_1.E_2, T) \to^{\rho} (E_1'.E_2, T') \in \delta_T$. Vice-versa, by virtue of $(R_T - 2)$ a transition $(E_1.E_2, T) \to^{\rho} (E_1'.E_2, T') \in \delta_T$ gives $(E_1, T) \to^{\rho} (E_1', T') \in \delta_T$. Using the induction hypothesis, $(E_1, T, M) \to^{\rho} (E_1', T', M')$. Finally, by $(T_{T/M}-2)$ we obtain $(E_1.E_2, T, M) \to^{\rho} (E_1'.E_2, T', M')$.

For $(T_{T/M}-4)$, we have $(E_1|E_2, T, M) \to^{\rho} (E_1', T', M') \in \delta_{T/M}$, that assumes $(E_1, T, M) \to^{\rho} (E_1', T', M') \in \delta_{T/M}$. By induction hypothesis, we have $(E_1, T) \to^{\rho} (E_1', T') \in \delta_T$ and by $(R_T - 4)$, we get $(E_1|E_2, T) \to^{\rho} (E_1', T')$. Vice-versa, a transition $(E_1|E_2,T) \rightarrow^{\rho} (E'_1,T')$ gives, by virtue of $(R_T\text{–}4)$, $(E_1,T) \rightarrow^{\rho} (E'_1,T')$. Using the induction hypothesis, $(E_1,T,M) \rightarrow^{\rho} (E'_1,T',M')$. Finally, by $(T_{T/M}\text{–}4)$ we obtain $(E_1|E_2,T,M) \rightarrow^{\rho} (E'_1,T',M')$.

Theorem 2. *$TS_{T/M}$ and TS_M are bisimilar.*

Proof. The proof is straightforward, because the effect of the trace on the attribute functions and the program execution is coded in memory M. Hence, intuitively the trace is redundant.

Corollary 1. *TS_T and TS_M are bisimilar.*

Proof. Combining the two Theorems and with transitivity, we prove the lemma.

4 Demonstration in LNT

We show how Sem_M facilitates the translation of EB^3 specifications to LNT for verification with the toolbox CADP. We translate the library management system (Fig.1) in LNT for $BID=\{b_1\}$ and $MID=\{m_1,m_2\}$.
LNT. LNT combines the best features of imperative and functional programming languages and value-passing process algebras. It has a user friendly syntax and formal operational semantics defined in terms of labeled transition systems (LTSs). LNT is supported by the LNT.OPEN tool of CADP, which allows the on-the-fly exploration of the LTS corresponding to an LNT specification. We present the fragment of LNT that is useful for this translation. Its syntax is given in Fig.6. LNT terms denoted by B are built from actions, choice (**select**), conditional (**if**), sequential composition (;), breakable loop (**loop** and **break**) and parallel composition (**par**). Communication is carried out by rendezvous on gates G with bidirectional transmission of multiple values. Synchronizations may also contain optional guards (**where**) expressing boolean conditions on received values. The special action δ is used for defining the semantics of sequential composition. The internal action is denoted by the special gate i, which cannot be used for synchronization. The parallel composition operator allows multiway rendezvous on the same gate. Expressions E are built from variables, type constructors, function applications and constants. Labels L identify loops, which can be stopped using "break L" from inside the loop body. Offer O can be either a send other (!) or a receive offer (?). Processes are parameterized by gates and data variables. The semantics of LNT are formally defined in [5].
Formalization. We explicitly model in LNT a memory, which stores the *attribute variables* and is modified each time an action is executed. We model the memory as a process $Memory$ placed in parallel with the rest of the system (a common approach in process algebra). To read the values of attribute variables, processes need to communicate with the memory M, and every action must have an immediate effect on the memory (so as to reflect the immediate effect on the execution trace). To achieve this, the memory process synchronizes with the rest of the system on every possible action of the system, and updates its attribute

$B ::= \textbf{stop} \mid \textbf{null} \mid G(O_1,\ldots,O_n)\ \textbf{where}\ E \mid B_1;B_2 \mid \textbf{if}\ E\ \textbf{then}\ B_1\ \textbf{else}\ B_2\ \textbf{end if} \mid$
$\quad \textbf{var}\ x{:}T\ \textbf{in}\ B\ \textbf{end var} \mid x := E \mid \textbf{loop}\ L\ \textbf{in}\ B\ \textbf{end loop} \mid \textbf{break}\ L \mid$
$\quad \textbf{select}\ B_1[\,]\ldots[\,]B_n\ \textbf{end select} \mid \textbf{par}\ G_1,\ldots,G_n\ \textbf{in}\ B_1||\ldots||B_n\ \textbf{end par} \mid$
$\quad P[G_1,\ldots,G_n](E_1,\ldots,E_n)$
$O ::= !E \mid ?x$

Fig. 6. Syntax of LNT

```
process memory[ACQ, DIS, REG, UNREG, LEND, RET: ANY] is
var mId : MEMBERID, bid : BOOKID, borrower : BOR, nbLoans : NB in
  (* attribute variables initialized *)
 mId := m_bot; borrower := BOR(m_bot); nbLoans := NB(0);
loop select
    ACQ(?bid)
[]  DIS(?bid, ?borrower)
[]  REG(?mid)
[]  UNREG(?mid)
[]  LEND(?bid, ?mid, !nbLoans, !borrower); borrower[ord(bid)] := mid;
    nbLoans[ord(mid)] := nbLoans[ord(mid)] + 1
[]  RET(?bid); mId := borrower[ord(bid)]; borrower[ord(bid)] := m_bot;
    nbLoans[ord(mid)] := nbLoans[ord(mid)] - 1
end select
end loop
end var
end process
```

Fig. 7. Memory in LNT

variables accordingly. Additional offers are used on each action, so that the current value of attribute variables can be read by processes during communication, and used to evaluate guarded expressions wherever needed.

Process *Memory* is given in Fig. 7. It runs an infinite loop, which "listens" to all possible actions of the system. We define two instances of the attribute variable *nbLoans* (one for each member) and one instance for *borrower* (one book). In the LNT expression *nbLoans*[*ord*(*mid*)], *ord*(*mid*) denotes the ordinate of value *mid*, i.e., a unique number between 0 and the cardinal of *mid*'s type minus 1. *nbLoans*[*ord*(*mid*)] is incremented after a *Lend* and decremented after a *Return* [7]. The action *Lend*(*mId*,*bId*) takes, besides *mid* and *bid*, *nbLoans* and *borrower* as parameters, because the latter are used in the evaluation of the guarded expression preceding *Lend* (**where** statement in Fig.8). Note how upon synchronisation on *Lend*, *nbLoans* and *borrower* are offered (!) by the *Memory* and received (?) by *loan* (Fig.8).

[7] see the definition of *nbLoans* in Fig.1

```
process MAIN [ACQ, DIS, REG, UNREG, LEND, RET: ANY] () is
par ACQ, DIS, REG, UNREG, LEND, RET in
par
   loop L in select break L [] book[ACQ, DIS](b1) end select end loop
||
   par
      loop L in select break L [] member[REG, UNREG, LEND, RET](m1)
      end select end loop
  ||
      loop L in select break L [] member[REG, UNREG, LEND, RET](m2)
      end select end loop
  end par
end par
||
  memory[ACQ, DIS, REG, UNREG, LEND, RET]
end par
end process

process loan[LEND, RET : ANY](mid: MEMBERID, bid : BOOKID) is
var borrower: BOR, nbLoans: NB in  (* NbLoans is set to 1 *)
  LEND(bid, mid, ?nbLoans, ?borrower) where
     ((borrower[ord(bid)] eq m_bot) and (nbLoans[ord(mid)] eq 1));
  RET(bid) end var
end process
```

Fig. 8. Main Program and Loan in LNT

The main program is given in Fig.8. All parallel quantification operations have been expanded as LNT is more structured and verbose than EB^3. Making use of the expansion rule $E^* = E.E^*|\lambda$, the Kleene Closure (as in $member(mId)^*$ in Fig.1) has been written accordingly. The full LNT program is in the appendix.
Case study. The case study examined here is the library management system enhanced wih extra functionalities:

1. A book can always be acquired by the library when it is not currently acquired.
2. A book cannot be acquired by the library if it is already acquired.
3. An acquired book can be discarded only if it is neither borrowed nor reserved.
4. A person must be a member of the library in order to borrow a book.
5. A book can be reserved only if it has been borrowed or already reserved by some member.
6. A book cannot be reserved by the member who is borrowing it.
7. A book cannot be reserved by a member who is reserving it.
8. A book cannot be lent to a member if it is reserved.
9. A member cannot renew a loan or give the book to another member if the book is reserved.
10. A member is allowed to take a reserved book only if he owns the oldest reservation.
11. A book can be taken only if it is not borrowed.
12. A member who has reserved a book can cancel the reservation at anytime before he takes it.
13. A member can relinquish library membership only when all his loans have been returned and all his reservations have either been used or canceled.
14. Ultimately, there is always a procedure that enables a member to leave the library.
15. A member cannot borrow more than the loan limit defined at the system level for all users.

Verification with CADP. All 15 requirements (that we name $p1-15$) to verify were expressed in MCL [15]. MCL is an extension of the alternation-free

μ-calculus [13] with ACTL-like [8] action formulas and PDL-like [14] regular expressions, allowing a concise and intuitive description of safety, liveness, and fairness properties without sacrificing the efficiency of verification. MCL combines data handling mechanisms (quantified variables and fixed point parameters), extended regular expressions, and constructs inspired from programming languages.

We verified the LNT specification corresponding to the library management system on an Intel(R) Core(TM) i7 CPU 880 @ 3.07GHz. All properties were proven true as expected and the results can be found in Fig.9 [8]. In the first line, (b, m) corresponds to the number of books and members of the IS. The second line gives the time needed to generate the corresponding LTS to the LNT model and the rest lines give the verification time for each requirement. Property $p2$ is formalised by the following MCL formula:

```
[true*. { ACQ !"B1" }. (not { DIS !"B1" })*. { ACQ !"B1" }] false
```

This formula follows the standard *safety* pattern: "$[\alpha]false$". It evaluates to true if no execution path matches the regular expression written inside the box modality. The meaning of $p2$'s regular expression is that we cannot have a sequence of ACQUIRE operations for book B1, if there is no DISCARD operation for B1 in the meantime. Notation *true* inside [ ] refers to all possible actions in the IS. Property $p12$ is formalised in the following manner:

```
[ true*. { RES !"M1" !"B1" }.
   ( not ({ TAKE !"M1" !"B1" } or { TRANSFER !"M1" !"B1" } ))* ]
<( not ( { TAKE !"M1" !"B1" } or { TRANSFER !"M1" !"B1" } ))*.
   { CANCEL !"M1" !"B1" }> true
```

This formula is a *liveness* property. Liveness properties of the form "$[\alpha]\langle\beta\rangle$true" state that every execution path matching the regular expression α (in this case, book B1 has been reserved by member M1 and subsequently neither taken not transfered) ends in a state from which there exists an execution path matching the regular expression β (in this case, the reservation can be cancelled before being taken or transfered).

5 Conclusion

In this paper, we presented an alternative semantics Sem_M for EB^3 that we proved equivalent to the standard semantics Sem_T. We showed how Sem_M facilitates the translation of EB^3 specifications to LNT for verification of temporal properties with CADP. The implementation of a compiler EB^3toLNT that automates the translation is in progress. We plan to formalize theoretically the translation and prove the strong equivalence of Sem_M with the standard semantics of LNT.

We will also study abstraction techniques for the verification of properties regardless of the number of components e.g. members, books that participate in the IS (Parameterized Model Checking). We will observe how the insertion of new functionalities to the ISs affects this issue. Finally, we will formalize this in the context of EB^3 specifications.

[8] analytical explanations are found in the appendix

(b,m)	(3,2)	(3,3)	(3,4)	(4,3)
time	1.892s	14.421s	31m39.743s	140m22.734s
p1	0.328s	1.800s	5m19.108s	20m13.876s
p2	0.208s	2.960s	9m26.623s	36m7.443s
p3	0.236s	4.960s	12m52.984s	55m33.744s
p4	0.240s	1.716s	5m15.644s	18m40.526s
p5	0.268s	2.188s	6m46.537s	21m52.770s
p6	0.216s	1.900s	5m53.798s	19m40.458s
p7	0.232s	2.224s	6m45.353s	22m39.589s
p8	0.224s	2.240s	6m52.046s	22m27.872s
p9	0.244s	2.288s	6m38.593s	22m29.164s
p10	0.280s	13.345s	43m59.497s	62m7.837s
p11	0.260s	2.544s	6m36.161s	22m14.027s
p12	0.260s	4.060s	10m47.316s	45m9.069s
p13	0.380s	4.288s	11m46.216s	1m7.924s
p14	0.264s	3.616s	10m41.476s	37m33.689s
p15	0.220s	2.828s	7m53.570s	28m56.505s

Fig. 9. Experimental Results

References

1. J.-R. Abrial. *The B-Book - Assigning programs to meanings.* Cambridge University Press, 2005.
2. J.A. Bergstra, A. Ponse, S.A. Smolka. *Handbook of Process Algebra.* Elsevier, 2001.
3. J.A. Bergstra, J. W. Klop. Algebra of Communicating Processes with Abstraction. *TCS*, 37:77–121, 1985.
4. T. Bolognesi, E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.
5. D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, C. McKinty, V. Powazny, F. Lang, W. Serwe, G. Smeding. *Reference Manual of the LOTOS NT to LOTOS Translator – Version 5.4.* INRIA/VASY, 2011.
6. R. Chossart. Évaluation d'outils de vérification pour les spécifications de systèmes d'information. Master's thesis, Université de Sherbrooke, 2010.
7. ClearSy. *Atelier B.* http://www.atelierb.societe.com.
8. R. De Nicola, F. Vaandrager. Three logics for branching bisimulation (extended abstract). In *LICS*, pages 118–129, 1990.
9. M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, M. Ouenzar. Comparison of model checking tools for information systems. In *Proc. of ICFEM*, Springer, 2010.
10. M. Frappier, R. St.-Denis. EB 3: an entity-based black-box specification method for information systems. *Software and System Modeling*, 2003.
11. H. Garavel, F. Lang, R. Mateescu, W. Serwe. CADP 2010: A toolbox for the construction and analysis of distributed processes. In *Proc. of TACAS*, Springer, 2011.
12. F. Gervais. *Combinaison de spécifications formelles pour la modélisation des systèmes d'information.* PhD thesis, 2006.
13. D. Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
14. C. Löding, O. Serre. Propositional dynamic logic with recursive programs. In *Proc. of FOSSACS*, Springer, 2006.
15. R. Mateescu, D. Thivolle. A model checking language for concurrent value-passing systems. In *Proc. of FM*, Springer, 2008.

6 Appendix

$$S_M - 1 : \frac{\rho \neq \lambda}{(\rho, M) \xrightarrow{\rho} (\surd, next(M)(\rho))} \quad S_M - 1' : \frac{}{(\lambda, M) \xrightarrow{\lambda} (\surd, M)}$$

$$S_M - 2 : \frac{(E_1, M) \xrightarrow{\rho} (E_1', M')}{(E_1.E_2, M) \xrightarrow{\rho} (E_1'.E_2, M')} \quad S_M - 3 : \frac{(E, M) \xrightarrow{\rho} (E', M')}{(\surd.E, M) \xrightarrow{\rho} (E', M')}$$

$$S_M - 4 : \frac{(E_1, M) \xrightarrow{\rho} (E_1', M')}{(E_1|E_2, M) \xrightarrow{\rho} (E_1', M')} \quad S_M - 5 : \frac{}{(E^*, M) \xrightarrow{\lambda} (\surd, M)}$$

$$S_M - 6 : \frac{(E, M) \xrightarrow{\rho} (E', M')}{(E^*, M) \xrightarrow{\rho} (E'.E^*, M')} \quad S_M - 7 : \frac{}{(\surd|[\Delta]|\surd, M) \xrightarrow{\lambda} \surd, M)}$$

$$S_M - 8 : \frac{(E_1, M) \xrightarrow{\rho} (E_1', M'),\ (E_2, M) \xrightarrow{\rho} (E_2', M')}{(E_1|[\Delta]|E_2, M) \xrightarrow{\rho} (E_1'|[\Delta]|E_2', M')} in(\rho, \Delta)$$

$$S_M - 9 : \frac{(E_1, M) \xrightarrow{\rho} (E_1', M')}{(E_1|[\Delta]|E_2, M) \xrightarrow{\rho} (E_1'|[\Delta]|E_2, M')} \neg in(\rho, \Delta)$$

$$S_M - 10 : \frac{(E, M) \xrightarrow{\rho} (E', M')}{(GE \Rightarrow E, M) \xrightarrow{\rho} (E', M')} \ \|GE[f_i \leftarrow M_i]\|$$

$$S_M - 11 : \frac{(E[\overline{x} := \overline{t}], M) \xrightarrow{\rho} (E', M')}{(P(\overline{t}), M) \xrightarrow{\rho} (E', M')} P(\overline{x}) = E \in ListPE$$

Fig. 10. Sem_M

```
module library is
type MEMBERID is m1, m2, m_bot with "eq", "ne", "ord" end type
type BOOKID is b1, b_bot with "eq", "ne", "ord" end type
type ACQUIR is array [0..1] of BOOL end type
type NB is array [0..2] of NAT end type
type BOR is array [0..2] of MEMBERID end type

process memory[ACQ, DIS, REG, UNREG, LEND, RET: ANY] is
var mId : MEMBERID, bid : BOOKID, borrower : BOR, nbLoans : NB in
  (* attribute variables initialized *)
  mId := m_bot; borrower := BOR(m_bot); nbLoans := NB(0);
loop select
    ACQ(?bid)
[]  DIS(?bid, ?borrower)
[]  REG(?mid)
[]  UNREG(?mid)
[]  LEND(?bid, ?mid, !nbLoans, !borrower); borrower[ord(bid)] := mid;
    nbLoans[ord(mid)] := nbLoans[ord(mid)] + 1
[]  RET(?bid); mId := borrower[ord(bid)]; borrower[ord(bid)] := m_bot;
    nbLoans[ord(mid)] := nbLoans[ord(mid)] - 1
end select
end loop end var
end process

process loan[LEND, RET : ANY](mid: MEMBERID, bid : BOOKID) is
var borrower: BOR, nbLoans: NB in  (* NbLoans is set to 1 *)
  LEND(bid, mid, ?nbLoans, ?borrower) where
     ((borrower[ord(bid)] eq m_bot) and (nbLoans[ord(mid)] eq 1));
  RET(bid) end var
end process

process book[ACQ, DIS: ANY](bid: BOOKID) is
var borrower: BOR in
  ACQ(bid); DIS(bid, ?borrower) where (borrower[ord(bid)] eq m_bot)
end var
end process

process member[REG, UNREG, LEND, RET: ANY](mid: MEMBERID) is
  REG(mid);
  loop L in select break L [] loan[LEND, RET](mid, b1)
            end select end loop; UNREG(mid)
end process

process MAIN [ACQ, DIS, REG, UNREG, LEND, RET: ANY] () is
par ACQ, DIS, REG, UNREG, LEND, RET in
par
   loop L in select break L [] book[ACQ, DIS](b1) end select end loop
||
   par
      loop L in select break L [] member[REG, UNREG, LEND, RET](m1)
      end select end loop
  ||
      loop L in select break L [] member[REG, UNREG, LEND, RET](m2)
      end select end loop
  end par
end par
|| memory[ACQ, DIS, REG, UNREG, LEND, RET]
end par
end process
end module
```

Fig. 11. LNT code for the Library Management System (2 members, 1 book)

P1 is a classical liveness P. The second conjunct expresses the eventuality that a book be withdrawn from the library before it is reacquired.

```
macro P (B) =
  (
    (
      [ ( not { ACQ !B } )* ] < { ACQ !B } > true
    )
    and
    (
      [ true*. { DIS !B }. ( not { ACQ !B } )* ] < { ACQ !B } > true
    )
  )
end_macro

P ("B1") and P ("B2") and P ("B3")
```

P2 is a safety P.

```
macro P (B) =
  (
     [ true*. { ACQ !B }. ( not { DIS !B } )*. { ACQ !B } ] false
  )
end_macro

P ("B1") and P ("B2") and P ("B3")
```

P3.

```
macro P (B) =
  (
     (
        [ true*. ( ({ LEND ?ANY : STRING !B } or { TAKE ?ANY: STRING !B }) ).
          (not { RET !B })*. { DIS !B } ] false
     )
     and
     (
        [ true*. { RES ?ANY : STRING !B }.
          (not ( { CANCEL ?ANY : STRING !B } or { RET !B } ))*. { DIS !B } ] false
     )
  )
end_macro

P ("B1") and P ("B2") and P ("B3")
```

P4. The first conjunct expresses the fact that a member cannot borrow a book if (s)he has not registered to the library. The second conjunct expresses that if a member relinquishes his/her membership, (s)he may not lend a book neither via the regular loan process *Lend* nor the reservation action *RES*.

```
macro P (M) =
  (
     (
        [ ( not { JOIN !M })*.
          ( {LEND !M ? ANY: STRING } or { TAKE !M ?ANY : STRING } ) ] false
     )
     and
     (
        [ true*. { LEAVE !M }.
          ( not { JOIN !M })*. ({LEND !M ? ANY: STRING } or { TAKE !M ?ANY : STRING } ) ] false
     )
  )
end_macro

P ("M1") and P ("M2")
```

Fig. 12. Verifications of Requirements P1-P4

P5: The first conjunct expresses the obligation for a book not to be lent in order to be added to the reservation list. The second conjunct complements the first in the sense that at least one loan cycle is completed in the beginning of the transition sequence via *RET !B* thus making the book available for loan again. The third conjunct denies any reservation history for the book in question. All possible loan operations should be excluded as well. Notably the P:

```
[ (not ( { RES ?ANY: STRING !B } ))*. { RES ?ANY: STRING !B } ] false
```

is false as the regular expression inside the *box* modality may trigger a loan before the reservation.

```
macro P(B) =
  (
     [ ( not ( { LEND ?ANY: STRING !B } or { TAKE ?ANY: STRING !B } ))*.
       { RES ?ANY: STRING !B } ] false
  )
  and
  (
     [ true*. { RET !B }. ( not ( { LEND ?ANY: STRING !B } or { TAKE ?ANY: STRING !B } ))*.
       { RES !B } ] false
  )
  and
  (
     [ ( not ( { LEND ?ANY: STRING !B } or { TAKE ?ANY: STRING !B } or
               { TRANSFER ?ANY : STRING !B } or { RES ?ANY: STRING !B } ))*.
       { RES ?ANY: STRING !B } ] false
  )
end_macro

P("B1") and P ("B2") and P("B3")
```

P6: The difficulty here lies in the fact that the borrower may transfer the book to another member. For this reason, the specification

```
[ true*. { LEND !M !B }. (not ({ RET !B }))*. { RES !M !B } ] false
```

would be false as can be verified by the model checker. Note that we need the conjunction of all instances of the macro Q for possible values of books and members. Symmetry in the EB^3 specification may lead us to consider one instance only e.g. $Q(''B1'',''M1'')$ in the P thus reducing the time it needs to evaluate.

```
macro Q (B, M) =
(
   [    true*. { LEND !M !B }. (not ({ RET !B } or { TRANSFER ?M2: STRING !B } ))*.
        { RES !M !B } ]
   false
)
end_macro

Q("B1", "M1") and Q("B2", "M1") and Q("B3", "M1") and
Q("B1", "M2") and Q("B2", "M2") and Q("B3", "M2") and
Q("B1", "M3") and Q("B2", "M3") and Q("B3", "M3")
```

P7

```
macro Q (B, M) =
(
   [  true*. { RES !M !B }. (not ({ TAKE !M !B } or { CANCEL !M !B }))*.
      { RES !M !B } ]
   false
)
end_macro

Q("B1", "M1") and Q("B2", "M1") and Q("B3", "M1") and
Q("B1", "M2") and Q("B2", "M2") and Q("B3", "M2") and
Q("B1", "M3") and Q("B2", "M3") and Q("B3", "M3")
```

Fig. 13. Verifications of Requirements P5-P7

P8: In this case, exploiting the symmetry is crucial to avoid the exponential state space explosion.

```
macro Q (B, M1, M2) =
(
   [ true*. { RES !M1 !B }. ( not ({ TAKE !M1 !B } or { CANCEL !M1 !B }) )*. { LEND !M2 !B } ]
   false
)
end_macro

Q("B1", "M1", "M2")
```

P9

```
macro Q (B, M) =
(
      [ true*. { RES !M !B  }. ( not ({ TAKE !M !B } or { CANCEL !M !B }) )*.
        { RENEW !B } ] false
)
end_macro

Q("B1", "M1")
```

P10: This P should be rephrased in the following way: It may not happen that a first member reserves a book and another member that reserves the book later takes it before the first member.

```
[
   true*.
   { RES ?M1: STRING ?B: STRING }.
   ( not ( { TAKE !M1 !B } or { CANCEL !M1 !B } or { TRANSFER !M1 !B } ))*.
   { RES ?M2: STRING !B where M2 <> M1 }.
   ( not ({ TAKE !M1 !B } or { CANCEL !M1 !B } or {TRANSFER !M1 !B } ))*.
   { TAKE !M2 !B }
] false
```

P11 corresponds to the classical P pattern: α_1 is not true between processes α_2 and α_3, which is expressed by the formula:
[true*. α_2. ($\neg$ α_3)*. α_1. ($\neg$ α_3)*. α_3] false, where
$\alpha_1 =$ ({ LEND !M !B } or { TAKE !M !B }) , $\alpha_2 =$ ({ LEND !M !B } or { TAKE !M !B }) and $\alpha_3 =$ { RET !B }.

```
macro Q (B, M) =
(
  [ true*. ({ LEND !M !B } or { TAKE !M !B }). ( not ( { RET !B } ) )*.
    ( { LEND !M !B } or { TAKE !M !B } ). ( not ( { RET !B } ))*.
    { RET !B } ] false
)
end_macro

Q("B1", "M1")
```

P12

```
macro Q (B, M) =
(
  [  true*. { RES !M !B }. ( not ({ TAKE !M !B } or { TRANSFER !M !B } ))* ]
     < ( not ( { TAKE !M !B } or { TRANSFER !M !B } ))*. { CANCEL !M !B } > true
)
end_macro

Q("B1", "M1")
```

Fig. 14. Verifications of Requirements P8-P12

P13

```
macro Q (B, M) =
(
  (
     [ true*.
       ( {LEND !M !B } or { TAKE !M !B } ).
       ( not ( {RET !B } or { TRANSFER !"M2" !B } or { TRANSFER !"M3" !B } ))*.
       { LEAVE !M }. ( not ( {RET !B }  or { TRANSFER !"M2" !B } or { TRANSFER !"M3" !B } ))*.
       ( {RET !B }  or { TRANSFER !"M2" !B } or { TRANSFER !"M3" !B }) ] false
  )
  and
  (
     [ true*. { RES !M !B }. ( not ( { TAKE !M !B } or { CANCEL !M !B } ))*.
       { LEAVE !M }. ( not ( { TAKE !M !B } or { CANCEL !M !B } ))*.
       ( { TAKE !M !B } or { CANCEL !M !B } ) ] false
  )
)
end_macro

Q("B1", "M1")
```

P14

```
macro Q (M) =
(
  [ true*. { JOIN !M }. ( not { LEAVE !M })* ] < ( not { LEAVE !M })*. { LEAVE !M } > true
)
end_macro

Q("M1")
```

P15: This P is dependent on the maximum number of books a member can have in his possession at any time. Supposing that this number is set to two the P can be written:

```
macro Q (M) =
(
  [ true*. let B1: STRING:= "B1", B2: STRING:= "B2" in
    ( { LEND !M !B1 } or { TAKE !M !B1 } ).
    ( not ({ TRANSFER ?M2: STRING !B1 } or { RET !B1 } ))*.
    ( { LEND !M !B2 } or { TAKE !M !B2 } ) end let ] false
)
end_macro

Q("M1")
```

Fig. 15. Verifications of Requirements P13-15