Efficient Operational Semantics for EB^3 for Verification of Temporal Properties

Dimitris Vekris and Catalin Dima

LACL, Université Paris-Est,
61 av. du Général de Gaulle,
94010 Créteil, France
{dimitrios.vekris, dima}@u-pec.fr

Abstract. EB^3 is a specification language 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. We then demonstrate how this new semantics facilitates the translation of EB^3 specifications to LOTOS NT (LNT for short) for verification of temporal properties 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 for information systems (ISs) [17]. 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 static properties also known as invariants or dynamic properties known as temporal 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 temporal property relates several events. Tools such as *Atelier B* [7] provide methodologies on how to define and prove

F. Arbab and M. Sirjani (Eds.): FSEN 2013, LNCS 8161, pp. 133–149, 2013.
DOI: 10.1007/978-3-642-40213-5_9,

invariants. In [12], an automatic translation of EB^3's attribute functions into B is attempted. Although the *B Method* [1] is suitable for specifying static properties, temporal properties are very difficult to express and verify in B. Hence, in our attempt to verify temporal 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 specific industrial case studies, but the prospect of a uniform translation from EB^3 program specifications is not studied. [6] casts an IS specification into LOTOS NT (LNT for short) [5] that serves as an input language to the verification suite CADP [11]. 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.

But the main problem in verifying EB^3 specifications against temporal-logic properties relies in the difficulty to handle the recursive definition of attribute functions if one relies on the classical, trace-based semantics. This type of semantics necessitates an unbounded memory model, and therefore only bounded model-checking can be achieved, in the absence of good abstractions that allow constructing finite-state models. This restriction is present in the original approach [10] and the subsequent model-checking attempt [9] even if all the entities utilized in the specification are finite.

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.

Furthermore, we explore the implications of this result to the translation of EB^3 specifications into LNT. 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. The fundamental difficulties for designing a compiler from EB^3 into LNT are summarized in [6]. In particular, LNT does not feature global variables. 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 "$C(T) \Rightarrow E$" can be written, where $C(T)$ is a predicate that refers to the system trace (the history of events) and E is a valid EB^3 expression.

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. The automatic translation of EB^3 specifications into LNT is studied in the companion paper [18]. We note that the translation of our example into LNT is produced using the tool presented in [18].

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.

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. The relevance of the $Sem_{T/M}$ semantics stems from the fact that it is pivotal in proving the bisimulation between Sem_T and Sem_M.

Case Study. We start by providing a simple case study which serves for introducing both the syntax and the semantics of EB^3. In Fig. 1, we give the functional requirements of a library management system and the corresponding EB^3 specification.The library system contains two entity types: *books* and *members*. The process *main* is the parallel interleaving between m instances of process *book* and p instances of processes describing operations on *members*. To avoid confusion, action names begin with uppercase letters, while process and attribute function names begin with lowercase letters.

The member *mId* registers to the library in order to start borrowing books, i.e. the action *Register*(*mId*). By the action *Unregister*(*mId*), (s)he relinquishes membership from the library. The book *bId* is acquired by the library so as to become available for lending, i.e. *Acquire*(*bId*). The inverse operation is expressed by the action *Discard*(*bId*). The member *mId* borrows the book *bId*, i.e. *Lend*(*bId*, *mId*) and returns it to the library after use, i.e. *Return*(*bId*). The process *book*(*bId*) denotes the lifecycle of the *book* entity *bId* from the

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.

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

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

$borrower(T\text{: } tr,\ bId\text{: BID}) : \text{MID}_\bot =$
match T **with**
[ ] $\rightarrow \bot$
| $T'.\ Lend(bId, mId) \rightarrow mId$
| $T'.\ Return(bId) \rightarrow \bot$
| $T'.\_ \rightarrow borrower(T', bId)$
end match

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

moment of its acquisition until its eventual discard from the library. The process $member(mId)$ denotes the lifecycle of the *member* entity *mId* from the point of its registration up until its membership drop. In the body of $member(mId)$, the process expression "$|||bId : BID : loan(mId, bId)^*$" denotes the interleaving of m instances of the process expression $loan(mId, bId)^*$ that, according to the standard semantics of the *Kleene Closure* operator (*), denotes the execution of $loan(mId, bId)$, $bId = \{b_1, \ldots, b_m\}$ an arbitrary, but bounded number of times. The attribute function $borrower(T, bId)$, where T is the current trace, returns the current borrower of book *bId* or $\bot$ (meaning *undefined*) if the book is not lent, by looking for actions of the form $Lend(bId, mId)$ or $Return(bId)$ in the trace. In process $book(bId)$, the action $Discard(bId)$ is thus guarded by $borrower(T, bId) = \bot$ to guarantee that the 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 action occurs and are eventually discarded (given that there are no more demands), *main*'s code can be modified in the following manner:

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

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

$main$ (A)

$\xrightarrow{Acq(b_2).Acq(b_1)}$

$borrower(T,\ b_1) = \bot \rightarrow Discard(b_1)\ |||$

$borrower(T,\ b_2) = \bot \rightarrow Discard(b_2)\ |||$

$(\ ||| mId : \text{MID} : member(mId)^*\ )$ (B)

$\xrightarrow{Reg(m_2).Reg(m_1)}$

$borrower(T,\ b_1) = \bot \rightarrow Discard(b_1)\ |||$

$borrower(T,\ b_2) = \bot \rightarrow Discard(b_2)\ |||$

$(\ ||| bId : \text{BID} : loan(m_1, bId)^*\ ).\ Unregister(m_1).\ member(m_1)^*\ |||$

$(\ ||| bId : \text{BID} : loan(m_2, bId)^*\ ).\ Unregister(m_1).\ member(m_2)^*$ (C)

$\xrightarrow{Lend(b_1,\ m_1)}$

$borrower(T,\ b_1) = \bot \rightarrow Discard(b_1)\ |||$

$borrower(T,\ b_2) = \bot \rightarrow Discard(b_2)\ |||$

$(Return(b_1).\ loan(m_1, b_1)^*\ |||\ loan(m_1, b_2)^*).Unregister(m_1).\ member(m_1)^*\ |||$

$(\ ||| bId : \text{BID} : loan(m_2, bId)^*\ ).\ Unregister(m_1).\ member(m_2)^*$ (D)

Fig. 2. Sample execution

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 in the body of $loan(mId : MID, bId : BID)$, writing the specification without the use of the guard:

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

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.

Execution. As a means to provide the operational intuition behind the three semantics introduced later in this section, we show how the EB^3 specification above is transformed through a four-step trace, assuming that the library may contain at most two books and at most two members, that is, $BID = \{b_1, b_2\}$ and $MID = \{m_1, m_2\}$.

First, we associate with the attribute function *borrower* two "memory cells", $bor[b_1]$ and $bor[b_2]$, meant to encode the value computed by the function for each book ID after each trace T. Similarly, we associate two memory cells $nbL[m_1], nbL[m_2]$ to the attribute function *nbLoans*. We also set $NbLoans = 2$ for the constant used in the definition of the process term *loan*.

	T	$M = (bor[b_1], bor[b_2], nbL[m_1], nbL[m_2])$
A	$[\,]$	$(\bot, \bot, \bot, \bot)$
B	$Acq(b_2).Acq(b_1)$	$(\bot, \bot, \bot, \bot)$
C	$T_B.Reg(m_2).Reg(m_1)$	$(\bot, \bot, 0, 0)$
D	$T_C.Lend(b_1, m_1)$	$(m_1, \bot, 1, 0)$

Fig. 3. States for the sample execution

Figure 2 shows how the process term *main* evolves by executing the valid trace $T_D = Acq(b_2).Acq(b_1).Reg(m_2).Reg(m_1).Lend(b_1, m_1)$, in which *Acq* stands for

Acquire and *Reg* for *Register*, respectively. During this evolution, the two attribute functions are computed according to their specifications in Fig. 2, inductively on the length of the trace. Hence, initially and after the execution of actions $Acq(b_2).Acq(b_1)$, the two attribute functions are undefined for both their arguments, while after the execution of the sequence of actions $Reg(m_2).Reg(m_1)$ we have "$nbLoans(T[C], m_1) = nbLoans(T[C], m_2) = 0$" and $borrower(T, .)$ remains undefined. These values are employed in order to check "$borrower(T, b_1) = \perp \wedge nbLoans(T, m_1) < 2$", which leads to the possibility for member m_1 to lend book b_1, and therefore to transform the process term at (C) in the process term at (D).

On the other hand, the table in Fig. 3 indicates the memory status after each (pair of) actions in the given trace. Initially, all memory cells carry the undefined value. After the trace $T[C]$, the value of memory cell $nbL_C[m_1]$ equals $nbLoans(T[C], m_1)$, and, similarly, "$nbL_C[m_2] = nbLoans(T[C], m_2)$". Note that the constraint checked at step $C \rightarrow D$ gives the same value regardless of the utilization of the value computed recursively for the attribute functions *borrower* and *nbLoans*, or by using the corresponding memory cells. Furthermore, the execution of action $Lend(b_1, m_1)$ triggers the update of the memory cell $bor[b_1]$ to m_1 and the incrementation of $nbL[m_1]$ to 1. This is modeled by the application of a function *next*, which defines the evolution of the system memory, and which is defined as follows: "$bor_D[b_1] = next(bor_C[b_1])$ [1]$= m1$ [2]","$bor_D[b_2] = bor_C[b_2] = \perp$", "$nbL_D[m_1] = nbL_C[m_1] + 1 = 1$" and also "$nbL_D[m_2] = 0$".

EB^3 Syntax and 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 $\alpha \in lab$ [3] 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 all 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 $ListPE$.

Sem_T [10] is given in Fig. 4 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 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

[1] here notation $next(x_C)$ denotes the modification on x's value in state (C) after executing transition $C \rightarrow D$

[2] see *borrower*'s script for "$T = T'.Lend(bId, mId)$" in Fig. 1

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

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

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

$$AtF ::= AtFDef \mid AtFDef\ ;\ AtF$$

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

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

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

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

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

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

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

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

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

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

Fig. 4. EB^3 Syntax and Sem_T

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 "$C\,(T) \Rightarrow E$" can execute E if the predicate $C\,(T)$ holds. The syntax of $C(T)$ is given below:

$$C\,(T) ::= true \mid false \mid op\,(C\,(T), \ldots, C\,(T)) \mid f_i\,(T, \ldots),\ i \in 1..n,\ op \in \{\wedge, \vee\}$$

This syntax is simplified in the sense that certain expressions cannot be supported in practice, e.g. "$nbLoans(T, mId) < NbLoans$" in Fig. 1. To palliate this, we need to add an attribute function name *nbLoans_lt_NbLoans* to *AtFct* and a corresponding attrribute function definition that implements this inequality. Finally, "$nbLoans(T, mId) < NbLoans$" has to be replaced by *nbLoans_lt_NbLoans* in the EB^3 specification. Note also that this syntax makes strictly use of those "$f_i\,(T, \ldots),\ i \in 1..n$" with Boolean return-type. Thus, the interpretation function of *guarded expressions* $\|\ .\ \|$ is the standard Boolean interpretation.

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

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

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

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

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

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

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

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

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

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

Fig. 5. $Sem_{T/M}$

$E[x{:=}t_1]|\ldots|E[x := t_n]$ 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, $|x{:}\emptyset{:}E = |[\Delta]|x{:}\emptyset{:}E = \surd$.

Attribute functions are defined in *AtFDef* [4] in Fig. 4, 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 $1 \leq i \leq n$, all calls to an attribute function f_l 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 our case study 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$_{T/M}$. $Sem_{T/M}$ is given in Fig. 5 as a set of rules named $T_{T/M}-1$ upto $T_{T/M}-11$. Each state is represented as a tuple (E, T, M). $M_i(\overline{x})$ is the variable

[4] This notation is different from the standard pattern-matching notation for attribute functions [10], but more compact

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

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

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

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

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

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

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

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

Fig. 6. Sem_M

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 action 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 $l < i$. The classic intepretations for Peano Arithmetic, Set Theory and Boolean Logic suffice to evaluate them. In $(T_{T/M}-10)$, $C[f_i \leftarrow M_i]$ denotes replacing all calls to f_i in C by M_i. The notation $\|.\|$ in $\|C[f_i \leftarrow M_i]\|$ corresponds to the standard interpretation of Boolean operators.

Sem$_\mathbf{M}$. Sem_M is given in Fig. 6 as a set of rules named S_M-1 to S_M-11. Sem_M derives from $Sem_{T/M}$ by simple elimination of T from each tuple (E, T, M) in rules $T_{T/M}-1$ upto $T_{T/M}-11$. It gives a finite state system. Intuitively, this means that the information on the history of executions is kept in M, thus rendering the presence of trace T redundant.

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

We present the proof of the bisimulation equivalence for the three semantics: 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 $(S, \{\xrightarrow{a}\}_{a \in Act}, I)$, where: (1) S is a set of states, (2) $\xrightarrow{a} \subseteq S \times S$, 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 $TS_i = (S_i, \{\xrightarrow{a}_i\}_{a \in Act}, I_i)$, where $i = 1, 2$ and a relation $R \subseteq S_1 \times S_2$, R is said to be a bisimulation and 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 $s_1 \xrightarrow{a}_1 s_1'$ then $\exists s_2' \in S_2$ such that $s_2 \xrightarrow{a}_2 s_2'$ and $(s_1', s_2') \in R$;
   (b) if $s_2 \xrightarrow{a}_2 s_2'$ then $\exists s_1' \in S_1$ such that $s_1 \xrightarrow{a}_1 s_1'$ and $(s_1', s_2') \in R$.

LTS Construction. For a given EB^3 process E, we associate three LTSs w.r.t. Sem_T, $Sem_{T/M}$ and Sem_M respectively. These correspond to the LTSs generated inductively by the rules given in Fig. 4–6. The whole process mimicks the construction of a transition system associated with a *transition system specification*, as in [16]. 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 equivalent w.r.t. bisimulation.*

Proof. Let $\rightarrow_1$ be the transition relation for TS_T and $\rightarrow_2$ be the transition relation for $TS_{T/M}$. The relation, which will give the bisimulation between TS_T and $TS_{T/M}$, is: $R = \{\langle (E, T, M), (E, T)\rangle \mid (E, T, M) \in S_{T/M} \wedge (E, T) \in S_T\}$. Note first that $\langle (E^0, [\,], M^0), (E^0, [\,])\rangle \in R$. We show that for any $\langle (E, T, M), (E, T)\rangle \in R$ and $(E, T, M) \xrightarrow{\rho}_1 (E', T', M') \in \delta_{T/M}$, we obtain $(E, T) \xrightarrow{\rho}_2 (E', T') \in \delta_T$ and vice-versa. We proceed with structural induction on E and present the proof for some cases.

For $(T_{T/M}-1)$, suppose $(\rho, T, M) \xrightarrow{\rho}_1 (\surd, T \cdot \rho, next(M)(\rho)) \in \delta_{T/M}$. The rule (R_T-1) allows us to conclude that also $(\rho, T) \xrightarrow{\rho}_2 (\surd, T \cdot \rho) \in \delta_T$. Conversely, suppose $(\rho, T) \xrightarrow{\rho}_2 (\surd, T \cdot \rho) \in \delta_T$. Note that each state $(E, T, M) \in S_{T/M}$ is of the form:

$$(E, T, next'(T, M^0)), \text{where}$$
$$next'\,(T, M) = \textbf{match}\ T\ \textbf{with}\ [\,] \rightarrow\ M \mid T' \cdot \rho \rightarrow next'\,(T', next\,(M, \rho))$$

Thus, there exists $(\rho, T, next'\,(T, M^0)) \xrightarrow{\rho}_1 (\surd, T \cdot \rho, next'\,(T \cdot \rho, M^0)) \in \delta_{T/M}$, which establishes rule $(T_{T/M}-1)$ by replacing $next'(T, M^0)$ with M as well as $next'(T \cdot \rho, M^0)$ with $next\,(M)\,(\rho)$.

For $(T_{T/M}-2)$, suppose $(E_1.E_2, T, M) \xrightarrow{\rho}_1 (E_1'.E_2, T', M') \in \delta_{T/M}$, which relies on the existence of a transition $(E_1, T, M) \xrightarrow{\rho}_1 (E_1', T', M') \in \delta_{T/M}$. By the induction hypothesis, $(E_1, T) \xrightarrow{\rho}_2 (E_1', T') \in Sem_T$ and by (R_T-2), we get $(E_1.E_2, T) \xrightarrow{\rho}_2 (E_1'.E_2, T') \in \delta_T$. Vice-versa, by virtue of (R_T-2) a transition $(E_1.E_2, T) \xrightarrow{\rho}_2 (E_1'.E_2, T') \in \delta_T$ necessitates $(E_1, T) \xrightarrow{\rho}_2 (E_1', T') \in \delta_T$. Using

the induction hypothesis, $(E_1, T, M) \xrightarrow{\rho}_1 (E'_1, T', M')$. Finally, by $(T_{T/M}-2)$ we obtain $(E_1.E_2, T, M) \xrightarrow{\rho}_1 (E'_1.E_2, T', M')$.

For $(T_{T/M}-10)$, we must prove that $\| C(T) \| = \| C[f_i \leftarrow M_i] \|$. Making use of the syntactic definition of $C(T)$ and the interpretation of $\| \, . \, \|$, it suffices to prove that $f_i(T, \overline{x}) = M_i(\overline{x}), i \in 1..n$ for any parameter vector $\overline{x}$ and trace T. We prove this by induction on T.

For $T = [\,]$, it is trivially $f_i(T, \overline{x}) = M_i^0(\overline{x}) = \| exp_i^0(\overline{x}) \|$, as $exp_i^0(\overline{x})$ contains no calls to other attribute functions. If $f_i(tl(T), \overline{x}) = M_i(\overline{x})$, $i \in 1..n$, we need to prove that:

$$f_i(T, \overline{x}) = next(M_i)(hd(T))(\overline{x}),\ i \in 1..n. \tag{1}$$

which we do again by induction on i.

Starting with $i = 1$, $next(M_i)(hd(T))(\overline{x})$, can be written as:

$$\| exp_1^{j,k}(\overline{x})[f_l \leftarrow \text{if } l < 1 \text{ then } next(M_l)(hd(T)) \text{ else } M_l] \|,$$

where k is specified by $hd(T)$ [5] and all calls to f_l, $l \in 2..n$ are replaced by M_l. Thus, due to the inductive hypothesis, it will be:

$$\| exp_1^{j,k}(\overline{x}) \| = \| exp_1^{j,k}(\overline{x})[f_l \leftarrow M_l] \|$$

A similar result holds for $cond_1^{j,k}$.

For $i > 1$, we rely on $f_l = next(M_l)(\rho_j)$, $l < i$, which guarantees that the property 1 holds for all values $l < i$.

This completes the proof of the case $(T_{T/M}-10)$. □

Theorem 2. *$TS_{T/M}$ and TS_M are equivalent w.r.t. bisimulation.*

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 equivalent w.r.t. bisimulation.*

Proof. Combining the two Theorems and the transitivity of bisimulation. □

4 Demonstration in LNT

The translation of EB^3 specifications is formalized in [18]. We show here how Sem_M facilitates the translation of EB^3 specifications to LNT for verification with the toolbox CADP. To this end, we present the translation of the EB^3 specification of Fig. 1 into LNT for BID=$\{b_1\}$ and MID=$\{m_1, m_2\}$ as was produced by the $EB^3 2LNT$ compiler [18].

LNT. LNT combines, in our opinion, 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

[5] see Fig. 5

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

$O ::= !E \mid ?x$

Fig. 7. Syntax of LNT

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. 7. 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. Gates in LNT (denoted with letter G with or without subscripts) correspond to the notion of labels in EB^3. Their parameters are called offers [6]. An offer O can be either a send offer (!) or a receive offer (?). 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. The last syntactic construct defines calls to process P that take gates $G_1, \ldots, G_n$ and variables $E_1, \ldots, E_n$ as actual parameters. The semantics of LNT are formally defined in [5].

Formalization. The principal gain from Sem_M lies in the use of *attribute variables*, the memory that keeps the values to all attribute functions. We need a mechanism that simulates this memory in LNT. The theoretical foundations of our approach are developed in [18]. In particular, 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 M 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 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.

[6] Offers are not explicitely mentioned in the syntactic rules for **par** and for procedural calls

```
process M [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. 8. Memory in LNT

These ideas are implemented in a tool called $EB^3$2LNT, presented in the companion paper [18]. We provide here the translation of the case study of library (with two members and two books) into LNT, obtained using $EB^3$2LNT.

Process M is given in Fig. 8. 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 M and received (?) by *loan* (Fig. 8).

The main program is given in Fig. 9. All parallel quantification operations have been expanded as LNT is more structured and verbose than EB^3. For most EB^3 operators, there are equivalent LNT operators [18]. Making use of the expansion rule $E^* = \lambda \mid E.E^*$, the Kleene Closure (as in *member(mId)** in Fig. 1) can be written accordingly. The full LNT program is in the appendix.

5 Conclusion

In this paper, we presented an alternative, traceless 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, by means of a translation in which the memory used to model attribute functions is implemented using an extra process that computes at each step the effect of each action on the memory. We presented the LNT translation of a case study involving a library with a predefined number

[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
     book [ACQ, DIS] (b1)
  ||
     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
||
  M [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. 9. Main program and the process associated with the computation of the attribute function *Loan* in LNT

of books and members, translation obtained with the aid of a compiler called $EB^3$2LNT. The $EB^3$2LNT tool is presented in detail in [18].

A formal proof of the correctness of the $EB^3$2LNT compiler is under preparation. The proof strategy is by proving that the memory semantics of each EB^3 specification and its LNT translation are bisimilar, and works by providing a match between the reduction rules of Sem_M and the corresponding LNT rules [5].

As future work, we plan to 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.

References

1. Abrial, J.-R.: The B-Book - Assigning programs to meanings. Cambridge University Press (2005)
2. Bergstra, J.A., Ponse, A., Smolka, S.A.: Handbook of Process Algebra. Elsevier (2001)

3. Bergstra, J.A., Klop, J.W.: Algebra of Communicating Processes with Abstraction. Journal of Theor. Comput. Sci. 37, 77–121 (1985)
4. Bolognesi, T., Brinksma, E.: Introduction to the ISO specification language LOTOS. Computer Networks and ISDN Systems 14(1), 25–59 (1987)
5. Champelovier, D., Clerc, X., Garavel, H., Guerte, Y., McKinty, C., Powazny, V., Lang, F., Serwe, W., Smeding, G.: Reference Manual of the LOTOS NT to LOTOS Translator - Version 5.4. INRIA/VASY (2011)
6. Chossart, R.: 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. De Nicola, R., Vaandrager, F.: Three logics for branching bisimulation (extended abstract). In: Proc. of LICS, pp. 118–129 (1990)
9. Frappier, M., Fraikin, B., Chossart, R., Chane-Yack-Fa, R., Ouenzar, M.: Comparison of model checking tools for information systems. In: Dong, J.S., Zhu, H. (eds.) ICFEM 2010. LNCS, vol. 6447, pp. 581–596. Springer, Heidelberg (2010)
10. Frappier, M., St.-Denis, R.: EB^3: an entity-based black-box specification method for information systems. In: Proc. of Software and System Modeling (2003)
11. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2010: A toolbox for the construction and analysis of distributed processes. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 372–387. Springer, Heidelberg (2011)
12. Gervais, F.: Combinaison de spécifications formelles pour la modélisation des systèmes d'information. PhD thesis (2006)
13. Kozen, D.: Results on the propositional mu-calculus. Journal of Theor. Comput. Sci. 27, 333–354 (1983)
14. Löding, C., Serre, O.: Propositional dynamic logic with recursive programs. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 292–306. Springer, Heidelberg (2006)
15. Mateescu, R., Thivolle, D.: A model checking language for concurrent value-passing systems. In: Cuellar, J., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 148–164. Springer, Heidelberg (2008)
16. Mousavi, M.R., Reniers, M.A.: Congruence for Structural Congruences. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 47–62. Springer, Heidelberg (2005)
17. Symons, V., Geoff, W.: The Evaluation of Information Systems: A Critique. Journal of Applied Systems Analysis 15 (1988)
18. Vekris, D., Lang, F., Dima, C., Mateescu, R.: Verification of EB^3 specifications using CADP, http://hal.inria.fr/hal-00768310

A LTS Construction

The construction is given by structural induction on E. In particular, we show how to construct:

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

w.r.t. Sem_M for several cases of E. 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:

$$S_\rho = \{(\rho, M^0)\} \bigcup \{(\surd, next(M^0))\},\ \delta_\rho = \{(\rho, M^0) \xrightarrow{\rho} (\surd, next(M^0))\},\ \rho \neq \lambda$$
$$S_\lambda = \{(\lambda, M^0)\} \bigcup \{(\surd, M^0)\},\ \delta_\lambda = \{(\lambda, M^0) \xrightarrow{\lambda} (\surd, M^0)\}$$
$$S_{E_1.E_2} = \{(E_1'.E_2, M) \mid (E_1', M) \in S_{E_1}\} \bigcup \{\bigcup_{(\surd, M_1) \in S_{E_1}} S_{E_2}^{M_1}\}$$
$$\delta_{E_1.E_2} = \{(E_1'.E_2, M) \xrightarrow{\rho} (E_1''.E_2, M') \mid (E_1', M) \xrightarrow{\rho} (E_1'', M') \in \delta_{E_1}\} \bigcup \{\bigcup_{(\surd, M_1) \in S_{E_1}} \delta_{E_2}^{M_1}\}$$
$$TS_{E^*} = lfp_F, \text{ where } F(TS_{E_x}) = TS_{E \cdot E_x} \cup TS_\lambda, \quad E^* = \lambda \mid E.E^*$$
$$TS_{E_1|[\Delta]|E_2}, \text{ where } E_1|[\Delta]|E_2 \doteq \sum_{i=1}^{r} C\,(T)^i \Rightarrow \rho^i(\overline{a}^i).E^i$$
$$S_{C\,(T) \Rightarrow E} = \begin{cases} \{(C\,(T) \Rightarrow E, M^0)\} \bigcup S_E \setminus \{(E, M^0)\}, & \text{if } \|C[f_i \leftarrow M_i^0]\| \\ \{(C\,(T) \Rightarrow E, M^0)\}, & \text{otherwise} \end{cases}$$
$$\delta_{C\,(T) \Rightarrow E} = \begin{cases} \delta_E[(E, M^0) \leftarrow (C\,(T) \Rightarrow E, M^0)], & \text{if } \|C[f_i \leftarrow M_i^0]\| \\ \emptyset, & \text{otherwise} \end{cases}$$

For $(E_1', M) \in S_{E_1}$, it will be $(E_1'.E_2, M) \in S_{E_1 \cdot E_2}$. 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. For TS_{E^*}, we need to compute the least fix point of function $F : TS \rightarrow TS$ 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. For $TS_{E_1|[\Delta]|E_2}$, $E_1|[\Delta]|E_2$ is written as a sum 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 C^i ($=true$ in the absence of condition):

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

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. Note that the rules $(S_M - 8)$ and $(S_M - 9)$ for Sem_M ensure the existence of this normal form. In the last case, for $\|C[f_i \leftarrow M_i^0]\| = true$, we need to construct S_E and replace (E, M^0) with $(C\,(T) \Rightarrow E, M^0)$.

B LNT code for the Library Management System

```
module Libr_Manag_Syst 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..1] of MEMBERID end type

process M [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
     book [ACQ, DIS] (b1)
  ||
     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
|| M [ACQ, DIS, REG, UNREG, LEND, RET]
end par
end process
end module
```