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
invariants. In [12], an automatic translation of EB3’s attribute functions into B
is attempted. Although the B Method [1] is suitable for specifying static prop-
erties, temporal properties are very difficult to express and verify in B. Hence,
in our attempt to verify temporal properties of EB3 specifications we move our
attention to model-checking techniques.

The verification of EB3 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 specifi-
cations used in [9] derive from specific industrial case studies, but the prospect
of a uniform translation from EB3 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 EB3 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 se-
manics necessitates an unbounded memory model, and therefore only bounded
model-checking can be achieved, in the absence of good abstractions that al-
low 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 EB3 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 pro-
gram 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
EB3 specifications into LNT. LNT is a process algebra specification that derived
from LOTOS [4]. As a process algebra, it shares many common features with
EB3 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 EB3 specifications to LNT is not evident. The fundamental dif-
ficulties for designing a compiler from EB3 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 proc1 || proc2 end par”,
so that every variable written in proc1 cannot be accessed in proc2. Although,
EB3 programmers cannot define global variables explicitly, EB3 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.

$$BID = \{b_1, \ldots, b_m\}, \ MIDs = \{m_1, \ldots, m_p\}$$

$$\text{main} = ( || bId : BID : \text{book}(bId) || ( || mId : MIDs : \text{member}(mId)^* ) )$$

$$\text{book}(bId : BID) = \text{Acquire}(bId), \ \text{borrower}(T, bId) = \bot = \Rightarrow \text{Discard}(bId)$$

$$\text{member}(mId : MIDs) = \text{Register}(mId), ( || bId : BID : \text{loan}(mId, bId)^* ). \ \text{Unregister}(mId)$$

$$\text{loan}(mId : MIDs, bId : BID) = \text{borrower}(T, bId) = \bot \land \text{nblloans}(T, mId) < \text{NbLoans}$$

$$\Rightarrow \text{Lend}(bId, mId), \ \text{Return}(bId)$$

$$\text{nblloans}(T : \text{tr}, mId : \text{MIDs}) : \text{Nat}_\bot =$$

$$\text{match T with}$$

$$\begin{array}{l}
| [] \Rightarrow \bot \\
| T'. \ \text{Lend}(bId, mId) \rightarrow \text{nblloans}(T', mId) + 1 \\
| T'. \ \text{Register}(mId) \rightarrow 0 \\
| T'. \ \text{Unregister}(mId) \rightarrow \bot \\
| T'. \ \text{Return}(bId) \land mId = \text{borrower}(T, bId)$$

$$\rightarrow \text{nblloans}(T', mId) - 1$$

$$| T'. \rightarrow \text{nblloans}(T', mId)$$

$$\text{end match}$$

$$\text{borrower}(T : \text{tr}, bId : \text{BID}) : \text{MIDs}_\bot =$$

$$\text{match T with}$$

$$\begin{array}{l}
| [] \Rightarrow \bot \\
| T'. \ \text{Lend}(bId, mId) \rightarrow mId \\
| T'. \ \text{Return}(bId) \rightarrow \bot \\
| T'. \rightarrow \text{borrower}(T', bId)$$

$$\text{end match}$$

Fig. 1. \textit{EB}^3 \text{ Specification and Attribute Function Definitions}

moment of its acquisition until its eventual discard from the library. The process \text{member}(mId) denotes the lifecycle of the \text{member} entity \text{mId} from the point of its registration up until its membership drop. In the body of \text{member}(mId), the process expression \text{"} || \text{bId} : \text{BID} : \text{loan}(mId, bId)^* \text{"} denotes the interleaving of \text{m} instances of the process expression \text{loan}(mId, bId)^* that, according to the standard semantics of the \textit{Kleene Closure} operator (*), denotes the execution of \text{loan}(mId, bId), \text{bId} = \{b_1, \ldots, b_m\} an arbitrary, but bounded number of times. The attribute function \text{borrower}(T, bId), where \text{T} is the current trace, returns the current borrower of book \text{bId} or \bot (meaning \textit{undefined}) if the book is not lent, by looking for actions of the form \text{Lend}(bId, mId) or \text{Return}(bId) in the trace. In process \text{book}(bId), the action \text{Discard}(bId) is thus guarded by \text{borrower}(T, bId) = \bot to guarantee that the book \text{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), \text{main}'s code can be modified in the following manner:

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

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 \textit{EB}^3 may not always be
Efficient Operational Semantics for $EB^3$

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
\[\text{loan}(mId : MID, bId : BID)\], writing the specification without the use of the
guard:

\[\text{borrower}(T, bId) = \bot \land nbLoans(T, mId) < NbLoans\]

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

\[
\begin{array}{c|c}
T & M = (\text{bor}[b_1], \text{bor}[b_2], \text{nbL}[m_1], \text{nbL}[m_2]) \\
A & \bot \\
B & \text{Acq}(b_2), \text{Acq}(b_1) \\
C & \text{Reg}(m_2), \text{Reg}(m_1) \\
D & \text{Lend}(b_1, m_1) \\
\end{array}
\]

Fig. 3. States for the sample execution

Figure 2 shows how the process term \text{main} evolves by executing the valid trace
\[T_D = \text{Acq}(b_2). \text{Acq}(b_1). \text{Reg}(m_2). \text{Reg}(m_1). \text{Lend}(b_1, m_1)\], in which $\text{Acq}$ 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 \[ nb\text{Loans}(T[C], m_1) = nb\text{Loans}(T[C], m_2) = 0 \] and borrower$(T)$ remains undefined. These values are employed in order to check \[ "\text{borrower}(T, b_1) = \bot \land nb\text{Loans}(T, m_1) < 2" \], 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 $nb\text{Loans}(T[C], m_1)$, and, similarly, \[ nbL_C[m_2] = nb\text{Loans}(T[C], m_2) \]. 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 $\text{Lend}(b_1, m_1)$ 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: \[ bor_D[b_1] = \text{next}(bor_C[b_1]) = m_1 \] \[ bor_D[b_2] = bor_C[b_2] = \bot \], \[ nbL_D[m_1] = nbL_C[m_1] + 1 = 1 \] and also \[ nbL_D[m_2] = 0 \].

**EB^3 Syntax and SemT.** We proceed with the formal definition of $EB^3$. We define a set of attribute function names $AtFct = \{ f_1, \ldots, f_m \}$ and a set of process function names $PFct = \{ P_1, \ldots, P_m \}$. Let $\rho \in \text{Act}$ stand for an action of either form $\alpha(p_1 : T_1, \ldots, p_n : T_n)$, where $\alpha \in \text{lab}$ is the label of the action and $p_i, i \in 1..n$ are elements of type $T_i$, or $\lambda$, which stands for the internal action. To simplify the presentation, we assume that all attribute functions $f_i$ have the same formal parameters $\pi$. 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 $\rho$ is the simplest $EB^3$ process, whose semantics are given by rules $R_T - 1, 1'$. Note that $\lambda$ is not visible in the $EB^3$ execution trace, i.e., it does not impact the definition of attribute functions. The symbol $\checkmark$ 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 $\Delta \subseteq \text{lab}$. The condition $in(\rho, \Delta)$ is true, iff the label of $\rho$ belongs to $\Delta$. The symmetric rules

---

1. here notation $next(xc)$ denotes the modification on $x$’s value in state (C) after executing transition $C \rightarrow D$
2. see borrower’s script for "$T = T'.\text{Lend}(bId, mId)$" in Fig. 1
3. we assume $\text{lab} = \{ \alpha_1, \ldots, \alpha_q \}$
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 $E_1 [[\text{lab}]] E_2$.

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:

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

This syntax is simplified in the sense that certain expressions cannot be supported in practice, e.g. “$\text{nbLoans}(T, mId) < \text{NbLoans}$” in Fig.1. To palliate this, we need to add an attribute function name $\text{nbLoans}_{lt}_{\text{NbLoans}}$ to $\text{AtFct}$ and a corresponding attribute function definition that implements this inequality. Finally, “$\text{nbLoans}(T, mId) < \text{NbLoans}$” has to be replaced by $\text{nbLoans}_{lt}_{\text{NbLoans}}$ 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 $\parallel$ 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
\[ \text{Sem}_{T/M} \]

\[ M_i^o(\overline{e}) = ||exp_i^o(\overline{e})|| \]
\[ \text{next}(M_i)(\rho_j)(\overline{e}) = ||exp_i^{j,k}(\overline{e})||[f_i \leftrightarrow \text{if } l < i \text{ then } \text{next}(M_i)(\rho_j) \text{ else } M_i][i] \]
\[ \text{if } ||\text{cond}_i^{j,k}(\overline{e})||[f_i \leftrightarrow \text{if } l < i \text{ then } \text{next}(M_i) \text{ else } M_i][i], i \in 1..n, k \in 1..m_j \]

\[ T_{T/M-1} : \]
\[ (\rho, T, M) \xrightarrow{\lambda} ( \sqrt{\lambda}, T, M ) \]

\[ T_{T/M-2} : \]
\[ (E_1, T, M) \xrightarrow{\rho} (E_1', T', M') \]
\[ (E_2, T, M) \xrightarrow{\rho} (E_2', T', M') \]

\[ T_{T/M-3} : \]
\[ (E, T, M) \xrightarrow{\rho} (E', T', M') \]

\[ T_{T/M-4} : \]
\[ (E_1, T, M) \xrightarrow{\rho} (E_1', T', M') \]
\[ (E_2, T, M) \xrightarrow{\rho} (E_2', T', M') \]

\[ T_{T/M-5} : \]
\[ (E, T, M) \xrightarrow{\rho} (E', T', M') \]

\[ T_{T/M-6} : \]
\[ (E, T, M) \xrightarrow{\rho} (E', T', M') \]
\[ (E, T, M) \xrightarrow{\rho} (E', T', M') \]

\[ T_{T/M-7} : \]
\[ (E, T, M) \xrightarrow{\rho} (E', T', M') \]
\[ (E, T, M) \xrightarrow{\rho} (E', T', M') \]

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

\[ T_{T/M-9} : \]
\[ (E_1, T, M) \xrightarrow{\rho} (E_1', T', M') \]
\[ (E_2, T, M) \xrightarrow{\rho} (E_2', T', M') \]

\[ T_{T/M-10} : \]
\[ (E, T, M) \xrightarrow{\rho} (E', T', M') \]
\[ (C(T) \Rightarrow E, T, M) \xrightarrow{\rho} (E', T', M') \]

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

Fig. 5. Sem_{T/M}

\[ E[x := t_1] \ldots E[x := t_n] \text{ and } E[x := t_1]||[\Delta] \ldots ||[\Delta]|E[x := t_n], \text{ where } E[x := t] \]
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, \( |x : \emptyset : E = ||\Delta|| : \emptyset : E = \sqrt{\Delta} \).

Attribute functions are defined in AtFDef \(^4\) in Fig. 4, where \( \text{exp}_i^{j,k} \) are expressions, \( \text{cond}_i^{j,k} \) are boolean expressions, \( \text{hd}(T) \) denotes the last element of the trace, and \( \text{tl}(T) \) 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 \( f_i \) occurring in \( \text{exp}_i^{j,k} \) or \( \text{cond}_i^{j,k} \) are parameterized by \( T \) if \( l \leq i \) or by \( \text{tl}(T) \) if \( l > i \). Such an ordering can be constructed if the \( \text{EB}^3 \) 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 \( \text{nbLoans} \) and \( \text{borrower} \) contain calls to \( \text{nbLoans} \) and \( \text{borrower} \) parameterized on \( \text{tl}(T) \). Also, \( \text{nbLoans} \) makes call to \( \text{borrower} \) parameterized on \( T \). Hence, \( f_1 = \text{borrower} \) and \( f_2 = \text{nbLoans} \).

\text{Sem}_{T/M} \cdot \text{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 \( (E, T, M) \). \( M_i(\overline{e}) \) is the variable

\(^4\) This notation is different from the standard pattern-matching notation for attribute functions [10], but more compact
Efficient Operational Semantics for $EB^3$

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 $cond_{i,k}^{j}$ that is evaluated true on every run. The action $\rho_j$ to occur “chooses” the corresponding $cond_{i,k}^{j}$ non-deterministically (in the sense that there may be many $k$ that make $cond_{i,k}^{j}$ 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 interpretations 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 $\| \cdot \|$ in $\| C[f_i \leftarrow M_i] \|$ corresponds to the standard interpretation of Boolean operators.

$Sem_M$. $Sem_M$ 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 $T_{T/M} - 1$ up to $T_{T/M} - 11$. 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 $(S, \{a \rightarrow\}_{a \in Act}, I)$, where: (1) $S$ is a set of states, (2) $\rightarrow \subseteq S \times S$, 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 \( TS_i = (S_i, \{ a_i \}_{a \in Act}, I_i) \), 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 = \{ (E, T, M), (E', T') \} | (E, T, M) \in S_{T/M} \land (E, T) \in S_T \}. Note first that \( (E^0, [], M^0) \) and \( (E^0, [], []) \) are such that \( (E, T, M) \in S_{T/M} \land (E, T) \in S_T \}. We show that for any \( (E, T, M), (E', T') \in R \) and \( (E, T, M) \xrightarrow{\rho, T, next(M)} (E', T', M') \in \delta_{T/M} \), we obtain \( (E, T) \xrightarrow{\rho, T, next(M)} (E', T', M') \in \delta_T \) and vice-versa. We proceed with structural induction on \( E \) and present the proof for some cases.

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

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

Thus, there exists \( (\rho, T, next'(T, M^0)) \xrightarrow{\rho, T, \rho, next'(T \cdot \rho, M^0)} \in \delta_{T/M} \), 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 \( (E_1, T, M) \xrightarrow{\rho, 1} (E'_1, T', M') \in \delta_T \). 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 \( (E_1, E_2, T) \xrightarrow{\rho, 2} (E'_1, E_2, T') \in \delta_T \). Using
the induction hypothesis, \((E_1, T, M) \xrightarrow{\rho_{i+1}} (E_1', T', M')\). Finally, by \((T_{T'/M-2})\) we obtain \((E_1,E_2, T, M) \xrightarrow{\rho_{i+1}} (E_1',E_2, T', M')\).

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 \(f_i(T, \pi) = M_i(\pi), i \in 1..n\) for any parameter vector \(\pi\) and trace \(T\). We prove this by induction on \(T\).

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

\[
f_i(T, \pi) = next(M_i)(hd(T))(\pi), i \in 1..n.
\]

which we do again by induction on \(i\).

Starting with \(i = 1\), \(next(M_i)(hd(T))(\pi)\), can be written as:

\[
\| \exp_i^{1,k}(\pi)[f_i \leftarrow if l < 1 then next(M_i)(hd(T)) else M_i] \|
\]

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

\[
\| \exp_i^{1,k}(\pi) \|=\|= \exp_i^{1,k}(\pi)[f_i \leftarrow M_i] \|
\]

A similar result holds for \(cond_i^{j,k}\).

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

This completes the proof of the case \((T_{T'/M-10})\).

\(\square\)

**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. \(\square\)

**Corollary 1.** \(TS_T\) and \(TS_M\) are equivalent w.r.t. bisimulation.

**Proof.** Combining the two Theorems and the transitivity of bisimulation. \(\square\)

### 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 \(EB^32LNT\) 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
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 $\delta$ 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 explicitly mentioned in the syntactic rules for par and for procedural calls.
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 \( nbLoans[\text{ord}(\text{mid})] \), \( \text{ord}(\text{mid}) \) denotes the ordinate of value \( \text{mid} \), i.e., a unique number between 0 and the cardinal of \( \text{mid} \)’s type minus 1. \( nbLoans[\text{ord}(\text{mid})] \) is incremented after a \( Lend \) and decremented after a \( Return \) \(^7\). The action \( Lend(\text{mid}, \text{bid}) \) takes, besides \( \text{mid} \) and \( \text{bid} \), \( nbLoans \) 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 \( E^* = \lambda E.E^* \), the Kleene Closure (as in \( member(\text{mid})^* \) 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 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\textsuperscript{3}2LNT. The EB\textsuperscript{3}2LNT tool is presented in detail in [18].

A formal proof of the correctness of the EB\textsuperscript{3}2LNT compiler is under preparation. The proof strategy is by proving that the memory semantics of each EB\textsuperscript{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\textsuperscript{3} specifications.

References

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 $M^0 \in M$ ($M$ 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:

18. Vekris, D., Lang, F., Dima, C., Mateescu, R.: Verification of $EB^3$ specifications using CADP, http://hal.inria.fr/hal-00768310
process M [ACQ, DIS, REG, UNREG, LEND, RET : ANY]

module Libr_Manag_Syst is

B LNT code for the Library Management System

end Libr_Manag_Syst
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