Verification of EB-3 specifications with model checking techniques

Dimitrios Vekris

UNIVERSITE PARIS-EST

École doctorale MSTIC

Thèse de doctorat

Pour obtenir le titre de

Docteur de l'Université Paris-Est

Spécialité : Informatique

Vérification de Spécifications EB^3 à l'aide de Model Checking

---

Dimitris Vekris

Directeur de thèse : Catalin Dima

préparée au LACL

Soutenue le 10 décembre 2014 devant le jury composé de :

Directeur:	Catalin Dima	Université Paris-Est - LACL
Raporteurs:	Marc Frappier	Université de Sherbrooke
	Gwen Salaün	INRIA Grenoble Rhône Alpes
Examinateurs:	Régine Laleau	Université Paris-Est - LACL
	Pascal Poizat	Université Paris-Ouest Nanterre la Défense - LIP6

Sommaire

Un système d'information est un système qui contrôle l'interaction entre plusiers composantes interdépendantes dont le rôle est de produire des informations. La méthode EB^3 [FSt03] est un langage de spécification développé pour la spécification des systèmes d'information. EB^3 a été utilisé à des projets de recherche inspirés par les domaines bancaire et médical, comme SELKIS [MIL$^+$11] et EB^{3}SEC [JFG$^+$10], mais il est aussi adapté pour un usage industriel. Le noyau du language EB^3 comprend des spécifications d'algèbre de processus afin de décrire le comportement des entités du système et des fonctions d'attributs qui sont des fonctions récursives dont l'évaluation se fait sur la trace d'exécution du système décrivant les attributs des entités. C'est à noter que l'algèbre de processus de EB^3 est inspirée de CSP [Hoa78] et de LOTOS [Lot01].

En EB^3, le comportement du système est défini au moyen d'expressions de processus. Celles-ci sont composées d'operateurs permettant d'ordonnancer les actions du système comme l'operateur de la séquence, du choix, de l'entrelacement et de la fermeture de Kleene. Parmi les expressions de processus de EB^3 figurent notamment les *expressions gardées* de la sorte "$ge \Rightarrow E$", où ge est une formule logique quantifiée du premier ordre contenant des appels à des fonctions d'attributs. Étant donné que l' évaluation des fonctions d'attributs dépend de la trace d'exécution actuelle, EB^3 permet la définition de contraintes assez complexes entre les différentes entités du système. De ce fait, la spécification des systèmes d'information en EB^3 facilite la compréhension du système et l'entretien du code.

Le sujet de cette thèse s'inscrit dans le cadre de la vérification des spécifications EB^3. Les propriétés nous concernant sont divisées principalement entre les propriétés de *sûreté*, c'est à dire celles satisfaites dans tous les états du système et les propriétés de *vivacité* exprimant l'éventualité que certaines actions puissent s'exécuter dans le système. À présent, la plupart des travaux effectués concernant la vérification des spécifications EB^3 se porte sur B [Abr05], une méthode spécialement conçue pour la vérification des propriétés de *sûreté*. Or, B n'est pas particulièrement adaptée à la verification des propriétés de *vivacité*. Ici, on se focalise sur les propriétés de *vivacité* concernant des systèmes d'information dont la vérification se fait à l'aide de model checking désignant une famille de techniques de vérification automatique des systèmes.

Une première approche vers la vérification de spécifications EB^3 à l'aide de model checking a été entreprise dans [FFC$^+$10], où le système de gestion d'une bibliothèque a été vérifié à l'aide des six outils de model checking notamment SPIN [Hol04], NuSMV [CCG$^+$02], FDR2 [Ros98], CADP [GLM$^+$11], ALLOY [Jac06] et ProB [LB03]. Les résultats concernant ALLOY ont été particulièrement encourageants. Cependant, [FFC$^+$10] ne prévoyait aucune possibilité de vérification générique pour les spécifications EB^3, dans le sens où la méthode se limitait à la vérification du système de gestion de la bibliothèque.

Dans cette thèse, on développe une traduction de specifications EB^3 vers LNT, un langage de spécifications de systèmes parallèles communicants trouvant application dans la modélisation et vérification formelle de protocoles de transfert de données. LNT est doté d'un ensemble des types abstraits algèbriques pour la définition des données et d' une algèbre de processus pour éxprimer le contrôle du système. De plus, il est un des langages d'entrée de CADP. De ce fait, le but de ce projet est de concevoir une méthode qui permettrait la vérification automatique des specifications EB^3 sans que la moindre intervention de l'utilisateur ne soit requise fournissant aux utilisateurs de EB^3 tous les outils de vérification fonctionnelle disponible dans CADP.

Malgré certains points communs de EB^3 et de LNT en matière d' opérations d'algèbre de processus, la représentation des spécifications EB^3 sous forme de système de transition d'états donne lieu à de potentielles explosions d'espace d'états. Cela s'expliquerait par le fait que, selon la sémantique classique de EB^3 [FSt03], l'évaluation des fonctions d'attributs pourrait entraîner des traversées complètes de la trace d'exécution du système à moins que des techniques du model checking borné visant à de réductions efficaces de l'espace d'état ne soient appliquées [BCC$^+$99].

Afin de pallier à cet inconvénient, on presente une sémantique opèrationnelle de EB^3, selon laquelle les fonctions d'attributs sont évaluées pendant l'exécution du programme puis stockées. Cette sémantique nous permet de définir la traduction automatique de EB^3 vers LNT. Notre traduction assure la correspondance un à un entre les états et les transitions des systèmes de transition étiquetés correspondant respectivement à des spécifications EB^3 et LNT. On automatise la traduction grace à l'outil EB^{3}2LNT. Par la suite, on s'en sert du *Model Checking Language* (MCL) [MT08] pour éxprimer et vérifier les propriétés temporelles sur les spécifications extraites de EB^{3}2LNT. La vérification se fait à la volée, c'est à dire l'exploration des états devant satisfaire la propriété en question se fait en même temps que le système de transition représentant la spécification LNT se construit, au moyen du model checker EVALUATOR 4.0 qui fait partie de CADP. C'est important de préciser à ce stade que c'est à l'utilisateur d'éxprimer les propriétés en MCL à vérifier ce qui fait que l'utilisateur doit avoir un certain niveau de connaissance de la syntaxe et de la sémantique de MCL pour s'en servir de notre méthode.

Comme CADP n'utilise que de méthodes explicites pour représenter l'espace d'états du système, la vèrification à l'aide de EB^{3}2LNT devient fastidieuse pour des systèmes d'information gérant de nombreux composants. Dans le but d'améliorer les résultats de notre approche concernant le model checking, on explore des techniques d'abstraction dédiées aux systèmes d'information spécifiées en EB^3. En particulier, on se focalise sur une famille spécifique de systèmes appelés paramétriques dont le comportement varie en fonction de la valeur prédéfinie d'un paramètre du système. Dans un premier temps, en portant sur de méthodes standards du model checking parametrique [CGB86, EN95], on construit une répresentation d'une variante du système de gestion de la bibliothèque sous forme de système de transition d'états que l'on nomme TS_n, où n est le nombre de membres du système. Ensuite, on prouve que TS_{n+1} est équivalent à TS_n modulo une relation appelée *stuttering bisimulation* qui préserve la logique PARCTL désignant un sur-ensemble de CTL* mis à part l'operateur *next-time* X. PARCTL nous permet d'éxprimer toute la gamme des propriétés auxquelles nous nous intéressons. Enfin, on applique cette méthode dans le contexte de EB^3 et on définit les conditions que les spécifications EB^3 doivent satisfaire afin que leurs systèmes de transition correspondents soit paramétriques.

Contents

Chapter 1

Introduction

Subject: This thesis deals with formal verification of information systems (ISs) that are specified in the EB^3 method [FSt03]. The focus is on model checking techniques, since we aim at verifying dynamic properties.

1.1 Context

ISs are systems that describe the interaction of interrelated components, whose role is to produce information. The EB^3 method is a special action-based specification language tailored for ISs that has been used extensively in the research projects SELKIS [MIL$^+$11] and EB^{3}SEC [JFG$^+$10], which deal with case studies inspired by the medical and banking domain. The formal specification of ISs enriched with security properties is the main objective for both projects.

A typical EB^3 specification defines entities, associations, and their respective attributes. It is worth noting that the process algebraic nature of EB^3 enables the explicit definition of intra-entity constraints, making them easy for the IS designer to review and understand. Yet, its particular feature compared to classical process algebras, such as CSP [Hoa78], lies in the use of *attribute functions*, a special kind of recursive functions evaluated on the system execution trace. Combined with guards, attribute functions facilitate the definition of complex inter-entity constraints involving the history of actions. Note that in classical state-based IS specifications, ordering constraints between actions are expressed via conditions on variables that denote the system state, the so-called state variables. As the history of actions in the IS is not reflected directly in the state variables, it is very difficult for the user to inspect and validate properties that are based on the history of actions. In that sense, the use of attribute functions and, consequently, of the system trace is intended to facilitate system understanding, enhance code modularity, and support maintenance.

1.2 Issues

Since ISs are complex systems involving data management and concurrency, a rigorous design process based on formal specification using EB^3 must be completed with effective formal verification features. Our attention is drawn to the *liveness* and *safety* properties of ISs. *Liveness* properties express the possibility that a certain action take place and *safety* properties express the certitude that a certain action should always or should never take place.

Existing attempts for verifying properties of EB^3 specifications are based on translations from EB^3 to other formal methods equipped with verification capabilities. A first line of work [GFL05, GFL06] focusses on devising translations from EB^3 attribute functions and processes to the B method [Abr05], which opens the way for proving *safety* properties of EB^3 specifications in the form of invariants using tools like Atelier B [Cl].

Let the following functional requirements describing the behaviour of a simplified library management system:

R1. A book can be acquired by the library. It can be discarded, but only if it has not been lent.

R2. An individual must join the library in order to borrow a book.

R3. A member can relinquish library membership only when all his loans have been returned.

R4. A member cannot borrow more than the loan limit defined at the system level for all users.

Requirements R1 and R3 are *liveness* properties, whereas requirements R2 and R4 are *safety* properties. Following the technique of [GFL05, GFL06], one may devise an equivalent B model from the EB^3 specification describing the behaviour of the previous library management system. Unfortunately, the B method is mainly used to verify safety properties over system specifications. Hence, this approach suffices to verify requirements R2 and R4, since *safety* properties can be formalized in the B method, but the verification of requirements R1 and R3 cannot be addressed under this technique and a full-scale verification of B specifications is impossible.

Another approach involves the translation of EB^3 specification to CSP‖B [ST02] as proposed in [ETL$^+$04]. *Liveness* properties expressed as temporal properties can be cast to FDR2 [Ros98] as explained in [LMC00]. As a result, both *liveness* and *safety* properties of the library specification can be verified with the aid of refinement checking and the FDR model checker. Although the method of [ETL$^+$04] is systematic, the translation of EB^3 specifications to equivalent CSP‖B models necessitates user intervention.

In [FM11], the verification of CTL formulas of the form "$AG(\psi \Rightarrow EF\phi)$" with the aid of B method is undertaken. CTL formulas "$AG(\psi \Rightarrow EF\phi)$" also known as reachability formulas denote *liveness* properties that are satisfied if for all states that are reachable from the initial states of the system, where formula ψ is valid, there is a sequence of transitions that lead to states of the system, where formula ϕ is valid. The goal of this study is to construct a program p that *refines* a reachability property q in the sense that the set of execution paths related to program p is strictly a subset of the execution paths related to the satisfaction of reachability formula q. The construction of p follows the algorithmic refinement laws of Morgan [Mor98] that are intended to decompose q into a sequence of properties, which can be trivially refined by simple system transitions (actions).

On the other hand, it is known that temporal logic can deal both with *safety* and *liveness* properties. Hence, another approach concerned with the verification of temporal logic properties of EB^3 specifications by means of model checking techniques is taken. For this purpose, the formal description and verification of the library management system specification [Ger06] using six model checkers are undertaken in [FFC$^+$10, Cho10] namely SPIN [Hol04], NuSMV [CCG$^+$02], FDR2, CADP [GLM$^+$11], ALLOY [Jac06] and ProB [LB03]. SPIN, CADP and FDR2 are called explicit model checkers. In CADP and FDR2, the transition

system describing the system specification is constructed explicitly prior to the property verification, whereas in the case of SPIN, the property is verified while the transition system is constructed in parallel (on-the-fly verification). Nu-SMV belongs to a group of model-checkers called symbolic model checkers. In symbolic model checking language specifications, the transition system is given in the form of Boolean formula. ALLOY is a contraint satisfaction model checker. In the case of contraint satisfaction model checkers, the verification is carried out by means of logic programming. It is worth mentioning that ALLOY's language used to model the system is the same as the language used to specify the property we need to verify. The same applies to PROB, whereas Spin, CADP, NuSMV use either action-based or state-based temporal logics.

ALLOY turned out to be the most efficient model checker of all six model checkers used in the paper, as model checking in Alloy makes use of efficient abstractions of the state space. Also, this study revealed the necessity of branching-time logics for accurately characterizing properties of ISs, and the fact that process algebraic languages are suitable for describing the behaviour and synchronization of IS entities. However, no attempt of providing a systematic translation from EB^3 to a target language accepted as input by a model checker was made so far.

1.3 Contribution

1.3.1 Part I

LOTOS-NT (LNT) [CCG$^+$11] is a new generation process algebraic specification language inspired from E-LOTOS [Lot01]. In this thesis, knowing that LNT is one of the input languages accepted by the CADP verification toolbox featuring full-blown temporal property validation capabilities, we address the problem of automatic translation of EB^3 models to equivalent LNT models.

At first sight, given that EB^3 has structured operational semantics based on a labelled transition system (LTS) model, its translation to a process algebra may seem straightforward. However, this exercise is rather complex, the main difficulty being to translate a history-based language (the control-flow depends on the entire system trace) to a process algebra with standard LTS semantics. On the other hand, the original trace-based semantics defined for finite-state systems in [FSt03] denoted Sem_{T} gives rise to unbounded memory models as the current trace is part of the current system state and, therefore, in the absence of good abstractions capable of reducing them to finite-state models, only bounded model-checking can be applied [BCC$^+$99]. This restriction is present in the original approach [FSt03] and the subsequent model-checking attempt [FFC$^+$10] even if all entities considered in the IS are finite.

To overcome this difficulty, we propose a formal semantics for EB^3 that treats attribute functions as state variables, the so-called *attribute variables*. 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. The derived memory semantics denoted Sem_{M} serves as the basis for applying a simulation strategy of attribute variables in LNT. We demonstrate that Sem_{M} is equivalent to Sem_{T}. This part of the work appears in [VD13].

Also, based on the efficient memory Sem_{M}, we propose a rigorous translation algorithm from EB^3 to LNT [CCG$^+$11]. As far as we know, this is the first attempt to provide a general translation from EB^3 to a classical value-passing process algebra. As noted previously, CSP

and LNT have been already considered in [FFC+10] for describing ISs, and identified as candidate target languages for translating EB^3. Since our primary objective is to provide temporal property verification features for EB^3, we focus our attention on LNT, and, hence, is equipped with on-the-fly model checking for action-based, branching-time logics involving data.

Another important ingredient of the translation is the multiway value-passing rendezvous of LNT, which allows to obtain a one-to-one correspondence between the transitions of the two LTSs underlying the EB^3 and LNT descriptions, and, hence, to preserve strong bisimulation. The presence of array types and of usual programming language constructs (e.g., loops and conditionals) in LNT is also helpful for specifying the memory, the Kleene star-closure operators, and the EB^3 guarded expressions containing attribute function calls. At last, the constructed data types and pattern-matching mechanisms of LNT enable a natural description of EB^3 data types and attribute functions.

Note that translating process algebra specifications to LNT is not a new idea. In [MS10], an automatic translation of π-calculus [MPW+92] to LNT is undertaken. In [LSH+09], FSP [MK+06] specifications are translated to LOTOS. The aim for both projects is to make the verification features of CADP available to π-calculus and FSP users. Another interesting project is found in [GSS+09], in which CHP (Communicating Hardware Processes), a process algebra devoted to the description of asynchronous hardware processes, is translated to LOTOS.

We implement our translation in the EB^{3}2LNT tool, thus, making possible the analysis of EB^3 specifications using all the state-of-the-art features of the CADP toolbox, in particular the verification of data-based temporal properties expressed in MCL [MT08] using the on-the-fly model checker EVALUATOR 4.0. This part of the thesis appears in [VLD+13]. We also provide a formal correctness proof of bisimulation equivalence of EB^3 and LNT specifications (those generated by EB^{3}2LNT). The work in [VLD+13] accompanied with the proof of bisimulation equivalence for EB^3 and corresponding LNT specifications will appear in the new issue of the Journal of Formal Aspects of Computing.

1.3.2 Part II

In practice, ISs are deemed to be complex systems for reasons involving heavy data management and concurrency between multiple components. This downside of ISs is confirmed in [FFC+10], since the model-checking problem of the library management system turns out to be intractable for more than four books and four members when model-checkers SPIN, NuSMV, FDR2, CADP, and ProB are employed. Only the use of ALLOY seems to produce efficient model-checking results. Similarly inadequate are the results of the approach we take in [VLD+13], mainly because the EVALUATOR 4.0 model checker of CADP applies strictly explicit techniques for state space generation.

The limitations of automatically verifying EB^3 specifications, as encountered in [FFC+10, VLD+13], motivate the development of efficient abstraction techniques. The inherent symmetry of EB^3 models may serve as the basis for cutting down on the number of components participating in the IS model. For example, the standard EB^3 pattern "$N_n \doteq |||\, x : X : proc\,(x)$", denoting the parallel interleaving of $|X|$ processes $proc\,(x)$, where $X = \{x_1, \ldots, x_n\}$ is a set in the IS and $|X| = n$ stands for X's cardinality, can be reduced to "$N_c \doteq |||\, x : Y : proc\,(x)$", where $Y = \{x_1, \ldots, x_c\}$ and c $(c < n)$ is called the *cut-off*. Then, the temporal property

$\bigvee_{i=1}^{n} \exists \Phi_i$ will hold on the initial system only if $\bigvee_{i=1}^{c} \exists \Phi_i$ holds on the reduced system[1]. Thus, the state space is reduced considerably and better model-checking results are obtained. The branch of model-checking concerned with discovering *cut-offs* for systems is called *parametric model checking* (PMC).

The rest of the work presented in this thesis intends to enrich existing model-checking techniques for verification of EB^3 specifications against temporal properties. Using standard techniques of PMC [CGB86, EN95] as a reference, we address the PMC problem in the context of ISs specified in EB^3. First, we show how the standard theory on PMC can be applied to the library management system on a merely state-based level, i.e. on the labelled transition system (LTS) representing the IS. The goal of this task is to give the interested reader the necessary insight into how to apply similar reasoning on a purely abstract process algebraic level, i.e. directly on the EB^3 specifications. We also qualify the ISs specified in EB^3 that are amenable to PMC, which we call *parametric* ISs (PISs). More specifically, we set precise criteria that *parametric* ISs should satisfy.

1.4 Thesis Structure

The rest of the manuscript is organized as follows:

- In Chapter 2, we present the syntax and trace-based semantics Sem_{T} of the EB^3 method. Also, we define the alternative memory semantics Sem_{M} and we demonstrate the correctness proof of bisimulation equivalence for Sem_{M} and Sem_{T}.
- In Chapter 3, we explain why Petri Nets [Pet75] and Nu-SMV are not suitable for formal verification of EB^3 specifications.
- In Chapter 4, we present the syntax and semantics of LNT. Then, we define the translation from EB^3 to LNT specifications, implemented by the EB^{3}2LNT translator and we present the application of this tool on two case studies. Furthermore, we establish the correctness proof of bisimulation equivalence of EB^3 specifications and LNT specifications generated by EB^{3}2LNT.
- In Chapter 5, we show the verification results, in conjunction with CADP, regarding the correctness requirements of the two case studies.
- In Chapter 6, we study the PMC problem in the context of ISs specified in EB^3.
- Finally, in Chapter 7, we summarize the results of this thesis and we draw up lines for future work.

[1] among elements of X, Φ_i contains only x_i and $\exists$ denotes CTL*'s *exists path* operator

Chapter 2

EB^3 Syntax and Semantics

2.1 Introduction

First, we give the standard trace-based operational semantics of EB^3. Then, 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 [CCG$^+$11]. 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.

Definition 2.1.1. *A standard* EB^3 *specification comprises the following elements:*

(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 execution trace, and

(4) input/output rules to specify outputs for input traces, or SQL expressions used to specify queries on the class diagram.

We limit the presentation to bullets (2) and (3). In the following, the term "EB^3 specification" refers strictly to the process algebraic specification in question and the related set of attribute function definitions.

2.2 EB^3 Syntax

2.2.1 Preliminary Definitions

Data types in EB^3. Let $\mathcal{C}$ be a set of constant symbols of various types Typ and let $\mathcal{V}$ be a set of function identifiers (variable symbols) of various types over $\mathcal{V}$. In particular, $\mathcal{V}$ serves as a generator of functional identifiers, from which the classic **if-then-else** function

identifier has been excluded. This particularity intends to accentuate the role of conditionals in the formalization of EB^3 specifications and to highlight the distinction between conditional and other functional terms at a purely conceptual level. More details can be found on the section discussing type (1) value expressions. Let also $\mathcal{X}$ be a set of variable symbols that are distinct from function identifiers in $\mathcal{V}$, i.e. $\mathcal{X} \cap \mathcal{V} = \emptyset$.

Let type assignment function $\sigma : \mathcal{C} \to Typ$ that assigns types from Typ to elements of $\mathcal{C}$ and let type assignment function $\pi : \mathcal{V} \to Typ$ that assigns types from Typ to elements of $\mathcal{V}$.

Definition 2.2.1. *The set of data types Typ comprise data types S and complex data types T that are defined inductively as follows:*

$$\begin{array}{rcl} S & ::= & S_0 \\ & ::= & (S, \ldots, S) \to S_0 \\ T & ::= & S \\ & ::= & T \to T \end{array}$$

Elementary data types $S_0 \in Typ$ may refer to abstract or enumerated sets, useful basic types like naturals $\mathbb{N}$, integers $\mathbb{Z}$, Booleans, Cartesian product of data types and finite power-set of data types. A more rigorous formalization of data type constructions by way of primitive data types, e.g. the inductive construction of data type "list of integers" by way of primitive data type "integer", is beyond the scope of this thesis and is, therefore, not addressed at all.

Notice that the types of function identifiers $g \in \mathcal{V}$ should belong to Typ. Intuitively, function identifiers $g \in \mathcal{V}$ taking l formal parameters, where the value of l varies with the function identifier g, are assigned type $(T_1, \ldots, T_l) \to T$ by type assignment function π provided that $T_1, \ldots, T_l$ are the types assigned to g's respective formal parameters and T is the return-type of g. As an example, consider the operator "+" denoting addition between integers, for which it follows that "$\pi(+) = (\mathbb{Z}, \mathbb{Z}) \to \mathbb{Z}$".

For the sake of simplicity, EB^3 specifications allow for null-ary constants $c \in \mathcal{C}$ only, and, therefore, type assignment function σ assigns to constants $c \in \mathcal{C}$ data types that belong to $S_0 \in Typ$.

2.2.2 EB^3 Specifications

In practice, an EB^3 specification comprises:

(1) a set of *action prototype definitions* $D_1; \ldots; D_q$,

(2) a set of *attribute function definitions* $A_1; \ldots; A_n$, and

(3) a set of *process definitions* $S_1; \ldots; S_m$ of the form "$P\,(x_1 : T_1, \ldots, x_r : T_r) = E$", where P is a process name and E is a *process expression*.

The complete EB^3 syntax is depicted in Figure 2.1.

Action prototype definitions.

For index $j \in \{1, \ldots, q\}$, action prototype definitions D_j of Figure 2.1 state that action labels α_j receive exactly p distinct typed *formal parameters* x_k for index $k \in \{1, \ldots, p\}$. Vector $\overline{x} = (x_1, \ldots, x_p)$ is called *formal vector* of action label α_j. Note that the use of

EB^3 specification	EB^3	::=	$\{D_1; \ldots; D_q; \quad A_1; \ldots; A_n; \quad S_1; \ldots; S_m\}$
Action prototype definitions	D_j	::=	$\alpha_j\,(x_1 : T_1, \ldots, x_p : T_p)$
Attribute function definitions	A_i	::=	$f_i\,(\mathsf{T} : \mathcal{T},\, y_1 : T_1, \ldots, y_s : T_s) : T =$
			match *last* (T) **with**
			$\quad \perp_{\mathsf{T}} : v_0$
			$\quad \mid c\_expr_1 \mid \ldots \mid c\_expr_q$
			$\quad [\mid \_ : v_{q+1}]$
			end match
Case expressions	c_expr_j	::=	$\alpha_j\,(z_1, \ldots, z_p) : v_j$
Pattern matching expressions	z	::=	$v \quad \mid \quad \_$
Type (1) value expressions	v_j	::=	$c \quad \mid \quad x \quad \mid \quad g\,(v_1, \ldots, v_l)$
		$\mid$	$f_h\,(\mathsf{T},\, v_1, \ldots, v_s)$
		$\mid$	$f_h\,(\mathit{front}\,(\mathsf{T}),\, v_1, \ldots, v_s)$
		$\mid$	**if** $g\,(v_1, \ldots, v_l)$ **then** v_{l+1} **else** v_{l+2} **end if**
Process expression definitions	S	::=	$P\,(x_1 : T_1, \ldots, x_r : T_r) = E$
Process expressions	E	::=	$\lambda \quad \mid \quad \alpha_j\,(u_1, \ldots, u_p) \quad \mid \quad E_1.E_2$
		$\mid$	$E_1 \mid E_2 \quad \mid \quad {E_0}^*$
		$\mid$	$E_1 \mid [\Delta] \mid E_2 \quad \mid \quad \mid x{:}V{:}E_0$
		$\mid$	$\mid [\Delta] \mid x{:}V{:}E_0 \quad \mid \quad ge \;\Rightarrow E$
		$\mid$	$P\,(u_1, \ldots, u_r)$
Type (2) value expressions	u	::=	$c \quad \mid \quad g\,(u_1, \ldots, u_l)$
Guards	ge	::=	$c \quad \mid \quad g\,(ge_1, \ldots, ge_l)$
		$\mid$	$f_i\,(\mathsf{T},\, u_1, \ldots, u_s)$

Figure 2.1: EB^3 syntax

duplicate variable identifiers among $x_1, \ldots, x_p$ is strictly forbidden. In the following, we will use notation $\overline{x}$ to denote the common *formal vector* of action labels α_j for $j \in \{1, \ldots, q\}$. Parameter p is called the *size* of vector $\overline{x}$.

Notation "$x_k : T_k$" denotes that *formal parameter* x_k has type T_k. Type T_k can only be one of the valid data types specified in Definition 2.2.1 (see Definition 2.2.2 for *well-formedness* of action prototype definitions). In practice, the *formal vector* of α_j varies depending on the index $j \in \{1, \ldots, q\}$, which, though, not reflected in the EB^3 syntax of Figure 2.1 for the sake of brevity, can be expressed more formally in the following manner:

Let $\overline{x}$ be the formal vector of action label $\alpha_j,\ j \in \{1, \ldots, q\}$ and let $p \in \mathbb{N}$ be the size of vector $\overline{x}$. Let also $\overline{x}'$ be the formal vector of action label $\alpha_k,\ k \in \{1, \ldots, q\}$ and let $p' \in \mathbb{N}$ be the size of vector $\overline{x}'$. In general, for formal parameter $x_k,\ k \in \{1, \ldots, p\}$ of α_j, i.e. $x_k \in \overline{x}$, such that "$x_k : T_k$" and formal parameter $x'_k,\ k \in \{1, \ldots, p'\}$ of α_j, i.e. $x'_k \in \overline{x}'$, such that "$x'_k : T'_k$", it follows that $x_k \neq x'_k$ and $T_k \neq T'_k$. Furthermore, it may be $p \neq p'$.

We, also, say that action labels α_j and α_k such that $j \neq k$ receive different *formal vectors* modulo renaming. The return-type of action label α_j has been omitted in the syntax of Figure 2.1, as return-types of action labels are mainly part of specific input/output rules stipulating interaction with users [FSt03]. However, this sort of information is not pertinent to the process algebraic specification at this level. Hence, for the sake of simplicity, action calls of the form $\alpha_j(x_1, \ldots, x_p)$ are considered untyped. Notice that every call to action label α_j employed throughout the EB^3 specification should be consistent with the typing of α_j in the corresponding action prototype definition D_j (see the corresponding part on *well-formedness* of process expressions for more details) and that, obviously, multiple action prototype definitions for given label α_j are forbidden.

The syntactic conditions to which action label definitions are subject are summarized in the following definition of *well-formedness* (the constraints over uniqueness of *formal parameter* identifiers and action names are omitted for brevity):

Definition 2.2.2. *Action prototype definitions $D_1, \ldots, D_q$ are said to be* well-formed *denoted by "$D_1; \ldots; D_q \leftharpoonup$" if and only if:*

$$\frac{\forall j \in \{1, \ldots, q\},\ \exists D_j ::= \alpha_j(\ldots, x_k : T_k, \ldots)\ \wedge\ \forall\ k \in \{1, \ldots, s\},\ T_k \in Typ}{D_1; \ldots; D_q \leftharpoonup}$$

Attribute function definitions.

For $i \in \{1, \ldots, n\}$, *attribute function* names f_i with unique prototype (as it appears in *attribute function* definition A_i of Figure 2.1):

$$f_i\,(\mathsf{T} : \mathcal{T},\ y_1 : T_1, \ldots, y_s : T_s) : T$$

receive the same vector of typed formal parameters $\overline{y} = (y_1, \ldots, y_s)$, which we call *attribute vector* modulo renaming, and, therefore, have the same arity s, which is by no means restrictive in terms of language expressiveness. Especially, for *formal parameters* $y_k \in \overline{y},\ k \in \{1, \ldots, s\}$, we use the alternative term *attribute parameter*. Based on the prototype of f_i, notation "$y_k : T_k$" denotes that *attribute parameter* y_k has type $T_k \in Typ$ and the return-type of f_i is T. In real EB^3 specifications, as is the case for the *formal vectors* of action labels, *attribute vectors* are distinct and have different arities given explicitly in the *attribute function* definitions $A_1; \ldots A_n$. *Attribute functions* f_i are defined recursively on the current trace "$\mathsf{T} : \mathcal{T}$" representing the history of actions executed in the system ($\mathcal{T}$ denotes the special type of system traces), with the aid of functions *last* (T), which denotes the last action of the system trace until present, and *front* (T), which denotes the trace without its last action. Symbol $\perp_{\mathsf{T}}$ represents the bottom (undefined) value with regards to the *lattice* of traces "$(\ \mathfrak{T}, \langle\ )$", where $\mathfrak{T}$ is the infinite domain of traces and notation "$\mathsf{T_1} \langle \mathsf{T_2}$" stands for the fact that trace $\mathsf{T_1}$ is a prefix of trace $\mathsf{T_2}$. In particular, both *last* (T) and *front* (T) match $\perp_{\mathsf{T}}$ when the trace is empty. Case expressions c_expr_j take one of the following syntactic forms:

- $\perp_{\mathsf{T}} : v_0$,
- $\alpha_j\,(z_1, \ldots, z_p) : v_j$, and
- $\_ : v_{q+1}$,

where v_j represents type (1) value expressions for $j \in \{0, \ldots, q+1\}$. The wild-card symbol ($\_$) matches all actions not matched by any of the preceding action patterns $\alpha_j\,(z_1, \ldots, z_p)$ for $j \in \{0, \ldots, q\}$. We, also, write $\overline{z}_j = (z_1, \ldots, z_p)$ for brevity. For index $k \in \{1, \ldots, p\}$, pattern matching expressions z_k are reduced to:

- type (1) value expressions v or
- the wild-card symbol ($\_$) that matches any constant $c_j \in \mathcal{C}$ appearing in the current action $\alpha_j\,(c_1, \ldots, c_p)$ of the system trace.

On the other hand, type (1) value expressions v comprise:

- constants $c \in \mathcal{C}$,
- variables $x \in \mathcal{X}$,
- functional terms $g\,(v_1, \ldots, v_l)$, in which $g \in \mathcal{V}$ and $v_1, \ldots, v_l$ are inductively defined type (1) value expressions,
- special functional terms on the trace variable T taking one of the following forms:
  - $f_h\,(\mathsf{T},\, v_1, \ldots, v_s)$,
  - $f_h\,(\mathit{front}\,(\mathsf{T}),\, v_1, \ldots, v_s)$

  for $h \in \{1, \ldots, n\}$, for which $v_1, \ldots, v_s$ are inductively defined type (1) value expressions, and
- conditional terms of the form "**if** $g\,(v_1, \ldots, v_l)$ **then** v_{l+1} **else** v_{l+2}", for which $g \in \mathcal{V}$ and $g\,(v_1, \ldots, v_l), v_1, \ldots, v_{l+2}$ are inductively defined type (1) value expressions.

Let "$Lab = \{\alpha_1, \ldots, \alpha_q\}$" be the set of action labels appearing uniquely in action prototype definitions $D_1; \ldots; D_q$ of Figure 2.1, and let also "$Attr = \{f_1, \ldots, f_n\}$" be the set of *attribute function* names appearing uniquely in *attribute function* definitions $A_1; \ldots; A_n$. By abuse of notation, "$y_1 : T_1, \ldots, y_n : T_n$" in A_i can be replaced by the syntactically simpler notation $\overline{y} : \overline{T}$, where $\overline{T} = (T_1, \ldots, T_l)$.

Definition 2.2.3. *For $i \in \{1, \ldots, n\}$, attribute function f_i is uniquely defined by the set of* EB^3 *type (1) value expressions $w_i^0, \ldots, w_i^q$, and w_i^{q+1} as follows:*

$$
\begin{aligned}
f_i\,(\mathsf{T} : \mathcal{T}, \overline{y} : \overline{T}) = \; & \textbf{\textit{match}}\ \mathit{last}\,(\mathsf{T})\ \textbf{\textit{with}} \\
& \quad \bot_{\mathsf{T}} : w_i^0 \\
& \quad \mid \alpha_1\,(\overline{z_1}) : w_i^1 \mid \ldots \mid \alpha_q\,(\overline{z_q}) : w_i^q \\
& \quad [\,\mid \_ : w_i^{q+1}\,] \\
& \textbf{\textit{end match}}
\end{aligned}
$$

where $\overline{z_1}, \ldots, \overline{z_q}$ are distinct formal vectors characterizing action labels $\alpha_1, \ldots, \alpha_q$ that have common arity p as was stipulated previously.

Let notation $var(w_i^j)$ denote the set of variable identifiers x appearing in value expression (1) w_i^j for indexes $i \in \{1,\ldots,n\}$, $j \in \{0,\ldots,q{+}1\}$. Let also notation $var(\overline{z}_j)$ denote the set of variable identifiers x among the pattern matching expressions $z \in \overline{z}_j$ for index $z \in \{1,\ldots,q\}$; that is to say the wild-card symbols (_) among the $z \in \{1,\ldots,q\}$ are eliminated. Notation $var(\overline{y})$ has a similar meaning to $var(\overline{z}_j)$.

As a means to help EB^3 programmers avoid writing meaningless or erroneous code, type (1) value expressions w_i^j are subject to some syntactic constraints. In particular, the type-checking rules for type (1) value expressions w_i^j are given as natural deduction rules with sequents of the form $w_i^j \leftharpoondown_1 T$. Notation $w_i^j \leftharpoondown_1 T$ asserts, on the one hand, that w_i^j is a *well-formed* expression of type T and, on the other hand, that the variables that are used in w_i^j have the types assigned to them by π and that the constants that are used in w_i^j have the types assigned to them by σ or more formally:

Definition 2.2.4. *By way of structural induction over* EB^3 *type (1) value expressions* w_i^j *(see Figure 2.1), the set of well-formed* w_i^j *for* $i \in \{1,\ldots,n\}$ *and* $j \in \{0,\ldots,q+1\}$ *is recursively defined as follows:*

$$\frac{\sigma(c) = T}{c \leftharpoondown_1 T} \tag{2.1}$$

$$\frac{\begin{array}{c}\exists z_k \in \overline{z}_j,\ c\_expr_j = \alpha_j(\ldots,z_k,\ldots) : w_i^j \ \wedge \\ \exists D_j ::= \alpha_j(\ldots,x_k : T_k,\ldots)\ \wedge\ x = z_k\ \wedge\ T_k = T\end{array}}{x \leftharpoondown_1 T} \tag{2.2}$$

$$\frac{\exists y_k \in \overline{y},\ A_i ::= f_i\,(\mathsf{T} : \mathcal{T},\ldots,y_k : T_k,\ldots) : \ldots\ \wedge\ y_k = x\ \wedge\ T_k = T}{x \leftharpoondown_1 T} \tag{2.3}$$

$$\frac{\pi(g) = (T_1,\ldots,T_l) \to T\ \wedge\ v_1 \leftharpoondown_1 T_1,\ldots,v_l \leftharpoondown_1 T_l}{g\,(v_1,\ldots,v_l) \leftharpoondown_1 T} \tag{2.4}$$

$$\frac{\exists A_h ::= f_h\,(\mathsf{T} : \mathcal{T}, y_1 : T_1,\ldots,y_s : T_s) : \ldots\ \wedge\ v_1 \leftharpoondown_1 T_1,\ldots,v_s \leftharpoondown_1 T_s}{f_h\,(\mathsf{T},\ v_1,\ldots,v_s) \leftharpoondown_1 T} \tag{2.5}$$

$$\frac{\exists A_h ::= f_h\,(\mathsf{T} : \mathcal{T}, y_1 : T_1,\ldots,y_s : T_s) : \ldots\ \wedge\ v_1 \leftharpoondown_1 T_1,\ldots,v_s \leftharpoondown_1 T_s}{f_h\,(\mathit{front}\,(\mathsf{T}),\ v_1,\ldots,v_s) \leftharpoondown_1 T} \tag{2.6}$$

$$\frac{g\,(v_1,\ldots,v_l) \leftharpoondown_1 \mathit{bool}\ \wedge\ v_{l+1} \leftharpoondown_1 T, v_{l+2} \leftharpoondown_1 T}{\textbf{\textit{if}}\ g\,(v_1,\ldots,v_l)\ \textbf{\textit{then}}\ v_{l+1}\ \textbf{\textit{else}}\ v_{l+2}\ \textbf{\textit{end if}} \leftharpoondown_1 T} \tag{2.7}$$

- Rule (2.1) states that constant $c \in \mathcal{C}$ has type T if T is exactly the type assigned to c by the type assignment function σ.
- Rules (2.2) states that variable x is of type T if a) there exists parameter z_k of action label α_j in the pattern matching expression "$c\_expr_j = \alpha_j(z_1,\ldots,z_p) : w_i^j$" such that $z_k = x$ for some index $k \in \{1,\ldots,p\}$, and b) the type of variable x is equal to the type of the k-th *formal parameter* x_k appearing in the corresponding action prototype definition D_j, i.e. $T_k = T$ (see Figure 2.1) for all $j \in \{1,\ldots,q\}$.
- Rules (2.3) states that variable x is of type T if for some $k \in \{1,\ldots,s\}$, the $(k+1)$-th parameter of *attribute function* prototype definition A_i namely $y_k : T_k$ (see Figure 2.1) is such that $x = y_k$, and the the type of *formal parameter* y_k referring to *attribute function* f_i is equal to T, i.e. $T_k = T$.

- Rule (2.4) states that a function call of the form $g\,(v_1, \ldots, v_l)$ is of type T if and only if the type assigned to variable identifier $g \in \mathcal{V}$ is $\pi(g) = (T_1, \ldots, T_l) \to T$ and for every value expression (1) v_k, for which $k \in \{1, \ldots, l\}$, it can be established that $v_k \leftarrow_1 T_k$.
- Rule (2.5) states that for index $h \in \{1, \ldots, n\}$, *attribute function* call "$f_h\,(\mathsf{T}, v_1, \ldots, v_s)$" has type T if and only if its corresponding (unique) *attribute function* definition "$A_h = f_h\,(\mathsf{T} : \mathcal{T}, y_1 : T_1, \ldots, y_s : T_s) : \ldots$" in the EB^3 specification (see Figure 2.1) is such that for every value expression (1) v_k, for which $k \in \{1, \ldots, s\}$, it can be established that $v_k \leftarrow_1 T_k$. Similar is the meaning of rule (2.6).
- In rule (2.7), the sequent "$g\,(v_1, \ldots, v_l) \leftarrow_1 \mathit{bool}$" denotes that $g\,(v_1, \ldots, v_l)$ is of type Boolean. Notice that v_{l+1} and v_{l+2} should be of the same type T to ensure the *well-formedness* of the **if-then-else** construct.

Attribute function ordering. *Attribute function* definitions are subject to further syntactic restrictions. In particular, we assume that the *attribute functions* are ordered, so that for all $h, i \in \{1, \ldots, n\}$ and for all $j \in \{1, \ldots, q\}$, every function call of the form $f_h\,(\mathsf{T}, \ldots)$ occurring in w_i^j satisfies $h < i$ and calls of the form $f_h\,(\mathit{front}\,(\mathsf{T}), \ldots)$ may occur in w_i^j for every $h \in \{1, \ldots, n\}$. For example, an *attribute function* definition f_i cannot contain recursive calls of the form $f_i\,(\mathsf{T}, \ldots)$, as circular dependencies among *attribute function* calls of that sort would lead to infinite *attribute function* evaluation.

In the following, we refer to the aforementioned hypotheses over *attribute function* definitions as the *attribute function ordering*. Furthermore, the *attribute function ordering* allows for a straightforward and considerably simple translation of EB^3 specifications to equivalent LNT specifications, whose objective is the efficient verification of EB^3 specifications over temporal properties. How this assumption facilitates the translation will be clarified later on in the corresponding section. Notice that this does not limit the expressiveness of EB^3 *attribute functions*, because every recursive computation on data expressions only (which keeps the trace unchanged) can be described using standard functions and not *attribute functions*.

Definition 2.2.5. *As a means to obtain well-formed attribute function definitions A_i for all $i \in \{1, \ldots, n\}$ (the exact syntax of A_i can be found in Definition 2.2.3) denoted by "$A_1; \ldots; A_n \leftarrow$", we stipulate the following:*

- $\mathit{var}(w_i^j) \subseteq \mathit{var}(\overline{y}) \cup \mathit{var}(\overline{z}_j)$,
- $w_i^j \leftarrow_1 T$, *where T is the return-type of attribute function name f_i.*
- *attribute function definitions A_i satisfy the attribute function ordering*

for $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, q+1\}$.

Process name definitions.

In keeping with process prototype definition $S \in \{S_1, \ldots, S_m\}$ of Figure 2.1, process expression name P receives r distinct typed *formal parameters*. Vector "$\overline{x} = (x_1, \ldots, x_r)$" is called *formal vector* of process expression name P. Like action labels, process expression names have no return-type. The set of process expression defined uniquely in process expression definitions "$S_k = P_k(x_1 : T_1, \ldots, x_r : T_r)$" for $k \in \{1, \ldots, m\}$ is denoted by "$\mathit{Func} = \{P_1, \ldots, P_m\}$".

Among these function names a special name *main* is used, which takes no *formal vector* and, whose meaning is similar to that of function *main* in programming language C. In real EB^3 specifications, the size r of *formal vector* $\overline{x}$ varies with process name P_k.

Process expressions are principally made of actions. Let Act be a set of *actions* written $\rho, \rho_1, \rho_2, \ldots$ appearing in EB^3 specifications. Action ρ is either the *internal action* written λ that does not appear in any of the action prototype definitions $D_1; \ldots; D_q$, or a *visible action* of the form $\alpha_j\,(\overline{u})$, where $\alpha_j \in Lab$ for some $j \in \{1, \ldots, q\}$ and $\overline{u} = (u_1, \ldots, u_p)$ is a vector of type (2) value expressions.

Type (2) value expressions v comprise:

- constants $c \in \mathcal{C}$,
- functional terms $g\,(v_1, \ldots, v_l)$, in which $g \in \mathcal{V}$ and $v_1, \ldots, v_l$ are type (2) value expressions defined inductively.

EB^3 processes can be combined with classical process algebra operators such as the *sequential composition* "$E_1.E_2$", the *non-deterministic choice* "$E_1 \mid E_2$", the *Kleene* closure "$E_0{}^*$", and the *parallel* composition "$E_1 \,|[\Delta]|\, E_2$" of E_1, E_2 with synchronization on action labels $\Delta \subseteq Lab$. Notation "$E_1 \,|||\, E_2$"stands for process expression "$E_1 \,|[\emptyset]|\, E_2$" and notation "$E_1 \,||\, E_2$" stands for process expression "$E_1 \,|[Lab]|\, E_2$". Besides, EB^3 syntax features calls to process expressions of the form $P_j\,(\overline{u})$, where $\overline{u} = (u_1, \ldots, u_r)$.

Furthermore, EB^3 is endowed with a special type of process expression called the *guarded expression* process "$ge \Rightarrow E_0$". The *guard* ge is a Boolean condition which may contain:

- constants $c \in \mathcal{C}$,
- *attribute function* calls of the form "$f_i\,(\mathsf{T},\, ge_1, \ldots, ge_s)$".

Last but not least, quantification is permitted for *choice* and *parallel* composition used in EB^3 specifications. For a set of constants "$V = \{c_1, \ldots, c_n\}$" such that $V \subseteq \mathcal{C}$, we define the quantified *choice* operator "$\mid x : V$" as follows:

$$\mid x : V : E_0 = E_0[x := c_1] \mid \ldots \mid E_0[x := c_n]$$

Moreover, for a set of action labels Δ such that "$\Delta \subseteq Lab$", we define the quantified *parallel* opearator "$|[\Delta]|\, x : V$" as follows:

$$|[\Delta]|\, x : V : E_0 = E_0[x := c_1] \,|[\Delta]|\, \ldots \,|[\Delta]|\, E_0[x := c_n],$$

where notation "$E[x := c]$" denotes the replacement of all occurrences of x by c in E. For instance, "$||\, x : \{1, 2, 3\} : a\,(x)$" stands for "$a\,(1) \,||\, a\,(2) \,||\, a\,(3)$".

The precedence of process algebra operators (from highest to lowest) is given below:

1	*
2	.
3	$\mid$
4	$\mid[\Delta]\mid$, $\mid\mid\mid$ as binary operator
5	$\mid[\Delta]\mid$, $\mid\mid\mid$ as quantified operator
6	$\Rightarrow$

We proceed with the definition of *well-formedness* for EB^3 process expressions.

Definition 2.2.6. *Let "$S ::= P(x_1 : T_1, \ldots, x_r : T_r) = E$" be a process expression definition such that $T_k \in Typ$ for all $k \in \{1, \ldots, r\}$. Let u denote type (2) value expressions appearing in process expression E (the syntax of u can be found Figure 2.1). The well-formedness of type (2) value expressions u denoted as "$u \leftarrow_2 T$" for some $T \in Typ$ is defined recursively by way of structural induction over u as follows:*

$$\frac{\sigma(c) = T}{c \leftarrow_2 T} \tag{2.8}$$

$$\frac{\pi(g) = (T_1, \ldots, T_l) \to T \ \wedge \ u_1 \leftarrow_2 T_1, \ldots, u_l \leftarrow_2 T_l}{g\,(u_1, \ldots, u_l) \leftarrow_2 T} \tag{2.9}$$

Definition 2.2.7. *Let "$S ::= P(x_1 : T_1, \ldots, x_r : T_r) = E$" be a process expression definition such that $T_k \in Typ$ for all $k \in \{1, \ldots, r\}$. Let ge denote guards appearing in process expression E (the syntax of ge can be found Figure 2.1). The well-formedness of guards ge denoted as $ge \leftarrow_3 bool$ is defined recursively as follows:*

$$\frac{\sigma(c) = T}{c \leftarrow_3 T} \tag{2.10}$$

$$\frac{\pi(g) = (T_1, \ldots, T_l) \to T \ \wedge \ ge_1 \leftarrow_3 T_1, \ldots, ge_l \leftarrow_3 T_n}{g\,(ge_1, \ldots, ge_l) \leftarrow_3 T} \tag{2.11}$$

$$\frac{\exists A_i ::= f_i\,(\mathsf{T} : \mathcal{T}, y_1 : T_1, \ldots, y_s : T_s) : T \ \wedge \ u_1 \leftarrow_3 T_1, \ldots, u_s \leftarrow_3 T_n}{f_i\,(\mathsf{T},\, u_1, \ldots, u_s) \leftarrow_3 T} \tag{2.12}$$

Definition 2.2.8. *Let "$S ::= P(x_1 : T_1, \ldots, x_r : T_r) = E$" be a process expression definition such that $T_k \in Typ$ for all $k \in \{1, \ldots, r\}$. The well-formedness of process expressions E denoted by "$E \leftarrow_4$" is defined by way of structural induction over E as follows:*

$$\frac{}{\lambda \leftarrow_4} \tag{2.13}$$

$$\frac{\forall u_k \in \overline{u},\ \exists D_j ::= \alpha_j(\ldots, x_k : T_k, \ldots) \ \wedge \ (u_k \leftarrow_2 T_k)}{\alpha_j(u_1, \ldots, u_p) \leftarrow_4} \tag{2.14}$$

$$\frac{(E_1 \leftarrow_4) \ \wedge \ (E_2 \leftarrow_4)}{E_1.E_2 \leftarrow_4} \tag{2.15}$$

$$\frac{(E_1 \leftarrow_4) \ \wedge \ (E_2 \leftarrow_4)}{E_1 \mid E_2 \leftarrow_4} \tag{2.16}$$

$$\frac{E_0 \leftarrow_4}{{E_0}^* \leftarrow_4} \tag{2.17}$$

$$\frac{(E_1 \leftarrow_4) \ \wedge \ (E_2 \leftarrow_4) \ \wedge \ (\Delta \subseteq Lab)}{E_1 |[\,\Delta\,]| E_2 \leftarrow_4} \tag{2.18}$$

$$\frac{(E_0 \leftarrow_4) \ \wedge \ (V \subseteq \mathcal{C}) \ \wedge \ (\Delta \subseteq Lab)}{|\, x{:}V : E_0 \leftarrow_4} \tag{2.19}$$

$$\frac{(E_0 \leftarrow_4) \ \wedge \ (V \subseteq \mathcal{C}) \ \wedge \ (\Delta \subseteq Lab)}{|[\,\Delta\,]|\ x{:}V : E_0 \leftarrow_4} \tag{2.20}$$

$$\frac{(ge \leftarrow_3) \ \wedge \ (E \leftarrow_4)}{ge \Rightarrow E \leftarrow_4} \tag{2.21}$$

$$\frac{\forall u_k \in \overline{u},\ \exists S_o ::= P_o(\ldots, x_k : T_k, \ldots) = \ldots\ \wedge\ (u_k \leftharpoonup_2 T_k)}{P_o\,(u_1, \ldots, u_r) \leftharpoonup_4} \tag{2.22}$$

- Typing rule (2.14) states that the *well-formedness* of action calls $\alpha_j(u_1, \ldots, u_p)$ for $j \in \{1, \ldots, q\}$ suggests the *well-formedness* of type (2) value expressions $u_1, \ldots, u_p$ denoted as $u_k \leftharpoonup_2 T_k$ for all $k \in \{1, \ldots, p\}$ (see Definition 2.2.6), where T_k represents the type of the k-th *formal parameter* of action name α_j as is specified in the corresponding action prototype definition D_j for $j \in \{1, \ldots, q\}$.

- Typing rule (2.21) states that the *well-formedness* of *guarded expression* process "$ge \Rightarrow E$" suggests the *well-formedness* of guard ge denoted as $ge \leftharpoonup_3$ (see Definition 2.2.7) and to the *well-formedness* of expression E.

- Typing rule (2.22) states that the *well-formedness* of process expression $P_o\,(u_1, \ldots, u_r)$ is guaranteed if there exists a process expression definition "$S_o ::= P_o(\ldots, x_k : T_k, \ldots) = E_o$" with $S_o \in \{S_1, \ldots, S_m\}$ such that for all value expression (2) u_k with $k \in \{1, \ldots, r\}$, it can be established that $u_k \leftharpoonup_3 T_k$.

- The remaining typing rules are straightforward, as they rely trivially on the inductive nature of EB^3 process expressions.

EB^3 process expressions must satisfy a number of additional restrictions that are not reflected in the previous definitions of *well-formedness*. We say that a variable $x \in V$ is *bound* in an expression E if all its occurrences are within the scope of a quantifier, and a formula is called *closed* if all variables occurring in the formula are bound. The following restrictions apply to EB^3 expressions:

1) Binary operators $|[\Delta]|$ can only be applied to EB^3 process expressions in which, for each action name $a \in \Delta$, if $a(\overline{x})$ occurs in one of the operands then all the variables in $\overline{x}$ must be bound in that operand. This restriction forbids expressions of the type $a(x_1)[a]a(x_2)$, which may be interpreted as either a single process being executed, when both variables have the same value, or two different processes being executed in an arbitrary order, when the two variables are instantiated with different values.

2) A similar constraint must be satisfied by expressions of the form $|[\Delta]|\,x : V : E_0$, namely for each action name $\alpha \in Lab$ for which $\alpha\,(\overline{u})$ occurs in E_0 for some vector $\overline{u}$ of type (2) value expressions, the variables "$x \in \mathcal{V} \setminus V$" in E_0 that do not occur in $\overline{u}$ must be bound in the body of E_0.

Definition 2.2.9. *Let "$S ::= P(x_1 : T_1, \ldots, x_r : T_r) = E$" be a process expression definition. We say that S is well-formed denoted by $S \leftharpoonup$ if and only if:*

- *$T_k \in Typ$ for all $k \in \{1, \ldots, r\}$,*
- *$E \leftharpoonup_4$*
- *the syntax of process expression E is consistent with bullets 1) and 2).*

Definition 2.2.10. *Let "$S_1; \ldots; S_m$" be a set of process expression definitions. We say that "$S_1; \ldots; S_m$" is well-formed denoted by "$S_1; \ldots; S_m \leftharpoonup$" if $S_k \leftharpoonup$ for all $k \in \{1, \ldots, m\}$.*

Definition 2.2.11. *Let "$EB^3 ::= D_1; \ldots; D_q;\ A_1; \ldots; A_n;\ S_1; \ldots; S_m$" denote* EB^3 *specification (the syntax of EB^3 can be found in Figure 2.1). We say that "EB^3" is well-formed denoted by "$EB^3 \leftharpoonup$" if and only if:*

- $D_1; \ldots; D_q \leftharpoonup$,
- $A_1; \ldots; A_n \leftharpoonup$,
- $S_1; \ldots; S_m \leftharpoonup$.

In the following, only *well-formed* EB^3 specifications are considered.

Example. As a simple example on the use of EB^3 process algebra, we consider a simplified version of the library management system, whose specification in EB^3 can be found in its entirety in Annex C of [Ger06]. Throughout this presentation, we explore the effect of *attribute functions* on the expressiveness of EB^3 specifications. First, we provide a succinct description in natural language of the system, which is summarized in the following specifications:

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 not borrowed.
4. A person must be a member of the library in order to borrow a book.
5. A member can relinquish library membership only when all his loans have been returned.
6. Ultimately, there is always a procedure that enables a member to leave the library.
7. A member cannot borrow more than the loan limit defined at the system level for all users.

The EB^3 specification describing the library management system is given in Figure 2.2. In Figure 2.2, the actual set of books is denoted by "$BID = \{b_1, \ldots, b_m\}$" and the set of persons eventually obtaining membership in the library is denoted by "$MID = \{m_1, \ldots, m_p\}$".

By abuse of notation, $MID_\perp$ (the set of member IDs plus value $\perp$) is also used to denote the return-type of *attribute function borrower* and $Nat_\perp$ (the set of natural number plus value $\perp$) is also used to denote the return-type of *attribute function nbLoans*. Moreover, notice the use of symbol $\perp$ in Figure 2.2 meaning *undefined* in the bodies of *attribute function borrower* and *attribute function nbLoans*. A more rigorous specification would involve the use of symbol $\perp_{NAT}$ in the body of *nbLoans* referring exclusively to the *undefined* value that belongs to set $NAT_\perp$ and the use of symbol $\perp_{MID}$ in the body of *attribute function borrower* referring exclusively to the *undefined* value that belongs to $MID_\perp$.

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 upper-case letters, while process and *attribute function* names begin with lower-case letters.

Member *mId* registers to become member to the library via action "*Register* (*mId*)". By way of action "*Unregister* (*mId*)", member *mId* relinquishes membership from the library.

```
Acquire (bId : BID);
Discard (bId : BID);
Register (mId : MID);
Unregister (mId : MID);
Lend (bId : BID, mId : MID);
Return (bId : BID);
```

```
book (bId : BID) =
    Acquire (bId). (borrower (T, bId) = ⊥) ⇒ Discard (bId);
loan (mId : MID, bId : BID) =
    (borrower (T, bId) = ⊥) ∧ (nbLoans (T, mId) < NbLoans) ⇒
        Lend (bId, mId). Return (bId);
member (mId : MID) =
    Register (mId). (||| bId : BID : loan (mId, bId)*). Unregister (mId);
main =
    (||| bId : BID : book (bId)*) ||| (||| mId : MID : member (mId)*);
```

```
nbLoans (T : T, mId : MID) : Nat⊥ =
    match last (T) with
        ⊥T : ⊥
    |   Lend (bId, mId) :
            nbLoans (front (T), mId) + 1
    |   Register (mId) : 0
    |   Unregister (mId) : ⊥
    |   Return (bId) :
          if mId = borrower (T, bId) then
            nbLoans (front (T), mId) − 1
          else nbLoans (front (T), mId) end if
    |   _ : nbLoans (front (T), mId)
    end match;
```

```
borrower (T : T, bId : BID) : MID⊥ =
    match last (T) with
        ⊥T : ⊥
    |   Lend (bId, mId) : mId
    |   Return (bId) : ⊥
    |   _ : borrower (front (T), bId)
    end match
```

Figure 2.2: EB^3 specification of a library management system

Action "*Acquire* (*bId*)" denotes acquisition of book *bId*, which automatically makes *bId* available for borrowing. The inverse operation is carried out by action "*Discard* (*bId*)". Member *mId* borrows book *bId* via action "*Lend* (*bId*, *mId*)" and returns it to the library after use via action "*Return* (*bId*)".

Process "*book* (*bId*)" denotes the life-cycle of *book* entity *bId* from the moment of its acquisition by the library until its eventual discard. Process *member* (*mId*) denotes the life-cycle of *member* entity *mId* from the moment of their registration until their membership removal. In the body of "*member* (*mId*)", process expression:

$$(|||\, bId : BID : loan\,(mId, bId)^*)$$

denotes the interleaving of m instances of process expression:

$$loan\,(mId, bId)^*$$

that denotes the execution of "*loan* (mId, bId)" for "$bId = \{b_1, \ldots, b_m\}$", an arbitrary, but bounded number of times (see semantics of *Kleene closure* operator in Section 2.3 for more details).

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 actions of the form "*Lend* (bId, mId)" or "*Return* (bId)" in the trace. In the body of process "*book* (bId)", action "*Discard* (bId)" is, thus, guarded by the following condition:

$$borrower\,(\mathsf{T},\ bId) = \perp$$

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*) provided that 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), the code of process *main* can be modified in the following manner:

$$\begin{array}{rl} main = & (|||\, bId : BID : Acquire\,(bId))\ .\ (|||\, mId : MID : member\,(mId)^*)\ . \\ & (|||\, bId : BID : Discard\,(bId)) \end{array}$$

Notice 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 in the body of process "*loan* $(mId : MID,\ bId : BID)$", writing the specification without the use of the following guard:

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

that illustrates the conditions under which a "*Lend* (bId, mId)" operation can take place (notably when the book is available and *attribute function* call "*nbLoans* (T, mId)" evaluates to an integer that is inferior to the fixed bound $NbLoans$), is not trivial.

2.3 EB^3 Semantics

We present three operational semantics for EB^3. The first, named Trace Semantics (Sem_{T}), is a rigorous redefinition of the standard EB^3 semantics, which can be found in [FSt03]. The second, named Trace/Memory Semantics ($Sem_{\mathsf{T/M}}$), 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 $Sem_{\mathsf{T/M}}$, we obtain the third semantics for EB^3 specifications, which we name Memory Semantics (Sem_{M}). The relevance of the $Sem_{\mathsf{T/M}}$ semantics stems from the fact that it is pivotal in proving the bisimulation between Sem_{T} and Sem_{M}.

2.3.1 Trace Semantics Sem_{T}

Before defining the formal semantics of EB^3 process expressions, we need to define the interpretation of type (1) value expressions v with respect to Sem_{T} that appear in the pattern matching constructs of *attribute function* definitions, the interpretation of type (2) value expressions u with respect to Sem_{T} that may appear either among the *actual vectors* of action labels α_j for $j \in \{1, \ldots, q\}$ or among the *actual vectors* of process names P_k for $k \in \{1, \ldots, m\}$ (see Figure 2.1 for details). We also need to define the interpretation of *guards ge* with respect to Sem_{T}.

We recall the existence of set $\mathcal{C}$ generating all constants c and the existence of set $\mathcal{V}$ generating all function name identifiers g. We also recall the corresponding type assignment function $\sigma : \mathcal{C} \to Typ$ assigning types to constants $c \in \mathcal{C}$ and function $\pi : \mathcal{V} \to Typ$ assigning types to function name identifiers $x, g \in \mathcal{V}$.

Following the standard approach of denotational semantics [Ten91] for programming languages, we define the interpretation of value expressions of EB^3 with respect to a given finite domain of values D.

Definition 2.3.1. *The semantics (denotation) of type $T \in Typ$ denoted by $\lfloor \cdot \rfloor_D$ with respect to finite domain D, where the subscript D is often omitted when it is obvious from the context, is defined by way of structural induction over type T as follows:*

$$\begin{aligned} \lfloor S_0 \rfloor_D &= D \\ \lfloor (S_1, \ldots, S_l) \to S_0 \rfloor_D &= [\, (\lfloor S_1 \rfloor_D, \ldots, \lfloor S_l \rfloor_D) \to D \,] \\ \lfloor T_1 \to T_2 \rfloor_D &= [\, \lfloor T_1 \rfloor_D \to \lfloor T_2 \rfloor_D \,], \end{aligned}$$

where $[\, A \to B \,]$ denotes the set of all functions from domain A to domain B.

The semantics of constant symbols c and non-user defined functions is specified by a given interpretation function $\mathcal{F}$ with respect to D. In particular, function $\mathcal{F}$ assigns:

- ◇ to function identifiers $g \in \mathcal{V}$ corresponding to standard non-user defined Boolean and arithmetic predicates, e.g. the addition operator "+" on integers, the inclusion operator "$\subseteq$" on sets and the "less than" operator "$<$" on real numbers, their standard denotations, and
- ◇ to usual constants, e.g. the Boolean "true" and the integer "1", their standard denotation.

Let U_T be the set of:

- all type (1) value expressions v such that $v \leftarrow_1 T$,
- all type (2) value expressions u such that $u \leftarrow_2 T$, and
- all *guards ge* such that $ge \leftarrow_3 T$.

Let $\prod$ denote the following generalization of set products, i.e.:

$$\prod_{i \in I} A_i = \{f : I \to A_i\}$$

for any set of indexes I generating finite domains A_i, $i \in I$. In the following, we use the standard notion of environment as the data base associating variable identifiers to special values named *denotations*. More formally,

Definition 2.3.2. *An environment $\tau : \mathcal{X} \to D$ is a partial function from variable identifiers in $\mathcal{X}$ to denotations (values) in domain D.*

Definition 2.3.3. *Let τ be an environment such that for all variable identifiers $x \in \mathcal{X}$ that appear in type (1) value expressions v such that $x \leftarrow_1 \pi(x)$, it follows that $\tau(x) \in \lfloor \pi(x) \rfloor_D$. We denote the set of environments in the context of* EB^3 *specifications satisfying the previous condition that we call the set of π-compatible environments as Env_π. It follows directly that:*

$$Env_\pi = \prod_{x \in \mathcal{X}} \lfloor \pi(x) \rfloor_D.$$

In Definition 2.3.3, we have assumed that all variable (non-function) identifiers x appearing in EB^3 specifications are generated by $\mathcal{X}$, i.e. $x \in \mathcal{X}$, and that the type assigned to x by type assignment function π is exactly the type deduced when applying the rules of *well-formedness* (see the corresponding definitions in Section 2.1) after statically analysing the EB^3 specification.

Then, let A, B, S be domains such that $S \subseteq A$. In the following, notation $f \oplus g$ refers to the *perturbation* of function $f : A \to B$ with respect to function $g : S \to B$ or more formally:

$$(f \oplus g)(x) = \begin{cases} g(x), & \text{if } x \in S \\ f(x), & \text{otherwise} \end{cases}$$

In the following, if T' denotes the current system trace, and action $\alpha_j(c_1, \ldots, c_p)$ is executed for some index $j \in \{1, \ldots, q\}$, and some constants $c_1, \ldots, c_p \in \mathcal{C}$, then "$\mathsf{T}'.\alpha_j(c_1, \ldots, c_p)$" shall denote the system trace just after action $\alpha_j(c_1, \ldots, c_p)$ has been executed.

Definition 2.3.4. *A trace environment $\tau^\star : \mathcal{T} \to D$ is a partial function from system traces to values in domain D that is defined as follows:*

$$\tau^\star(\mathsf{T}) = \begin{cases} [\,], & \textit{if} \quad \mathsf{T} = [\,] \\ [\; z_1 \leftarrow \mathcal{F}(c_1),\; \ldots,\; z_p \leftarrow \mathcal{F}(c_p)\;], & \textit{if} \quad \mathsf{T} = \mathsf{T}'.\alpha_j(c_1, \ldots, c_p) \end{cases}$$

where $\mathsf{T}' \in \mathcal{T}$ is a valid trace, $j \in \{1, \ldots, q\}$, and $c_1, \ldots, c_p \in \mathcal{C}$ are constants.

Function $\tau^\star$ assigns to every *formal parameter* z_k of action label a_j (corresponding to the current action occurring in the system) that appears in the pattern matching expression "$c\_expr_j ::= \alpha_j(z_1, \ldots, z_p) : w_h^j$" value $\mathcal{F}(c_k)$ for all $k \in \{1, \ldots, p\}$.

We proceed with the semantics of EB^3 type (1) value expressions, which is defined using valuation functions $[\![\cdot]\!]_{1,D}^{\mathsf{T},T} : U_T \to [\,\tau_\pi^T \to \lfloor T \rfloor_D\,]$, where the subscripts and superscripts D, T and π are omitted when they can be extracted from the context.

Definition 2.3.5. *Let EB^3 be a well-formed* EB^3 *specification and let $\tau \in Env_\pi$ be a π-compatible environment. For the different values of trace variable T given as follows:*

$$\mathsf{T} = \begin{cases} [\,] \\ \mathsf{T}'.\; \alpha_j(c_1, \ldots, c_p) \quad \textit{for} \quad j \in \{1, \ldots, q\},\; c_1, \ldots, c_p \in \mathcal{C} \end{cases}$$

where $\mathsf{T}' \in \mathcal{T}$ is a valid trace, the semantics $[\![\cdot]\!]_1^{\mathsf{T}} : Env_\pi \times \mathcal{T} \to \lfloor T \rfloor_D$ for some type $T \in Typ$ of EB^3 *type (1) value expressions $w_i^j \leftarrow T$ appearing in EB^3 for some $i \in \{1, \ldots, n\}$ and some*

$j \in \{1, \ldots, q\}$ with respect to Sem_{T} under environment τ and trace T is defined by way of structural induction over w_i^j as follows:

$$\begin{aligned}
[\![c]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) &= \mathcal{F}(c) && (2.23)\\
[\![x]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) &= \tau(x) && (2.24)\\
[\![g\,(v_1, \ldots, v_l)]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) &= \mathcal{F}(g)([\![v_1]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}),\ \ldots,\ [\![v_l]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T})) && (2.25)\\
[\![f_h\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) &= [\![w_h^j]\!]_1^{\mathsf{T}}(\tau^\star(\mathsf{T}) \cup \tau',\ \mathsf{T}) && (2.26)\\
\textit{where}\quad \tau' &= [\ y_1 \leftarrow [\![v_1]\!]_1^{\mathsf{T}}\ (\tau,\ \mathsf{T}),\ \ldots,\ y_s \leftarrow [\![v_s]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T})\ ]\\
[\![f_h\,(\mathit{front}(\mathsf{T}), v_1, \ldots, v_s)]\!]_1^{\mathsf{T}}(\tau,\ [\ ]) &= \perp^{\mathcal{F}} && (2.27)\\
[\![f_h\,(\mathit{front}(\mathsf{T}), v_1, \ldots, v_s)]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) &= [\![f_h\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}') && (2.28)\\
\textit{where}\quad \mathsf{T} &= \mathsf{T}'.\alpha_j(c_1, \ldots, c_p)\\
[\![\textbf{\textit{if}}\ g\,(v_1, \ldots, v_l)\ \textbf{\textit{then}}\ v_{l+1} && && (2.29)\\
\textbf{\textit{else}}\ v_{l+2}\ \textbf{\textit{end if}}]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) &= ([\![v_{l+1}]\!]_1^{\mathsf{T}} \oplus [\![v_{l+2}]\!]_1^{\mathsf{T}})(\tau,\ \mathsf{T})\\
\textit{where}\quad [\![v_{l+2}]\!]_1^{\mathsf{T}} &: \mathit{Env}_\pi \times \mathcal{T}' \to D\\
\mathcal{T}' &= \{\mathsf{T} \in \mathcal{T}\ \mid\ [\![g\,(v_1, \ldots, v_l)]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) = \mathsf{false}\}
\end{aligned}$$

It should be noted that operator $[\![\cdot]\!]_1^{\mathsf{T}}$ is actually overloaded. A more accurate definition would involve the use of $[\![\cdot]\!]_{1,D}^{\mathsf{T},T}$ defined in respect to given domain D and given data type T. For the sake of simplicity, we have dropped subscript D and superscript T.

We recall that *attribute function* calls appearing in type (1) value expressions may take one of the following forms (see Figure 2.1):

- $f_h\,(\mathit{front}(\mathsf{T}),\ v_1, \ldots, v_s)$ for $h > 0$,
- $f_h\,(\mathsf{T},\ v_1, \ldots, v_s)$ for $h < i$,

(see *attribute function ordering* for details).

Upon assigning to trace variable T the list of actions occurring in the system until present, trace variable T takes one of the following forms:

1. the empty trace [ ],
2. $\mathsf{T}'.\alpha_j(c_1, \ldots, c_p)$ for $j \in \{1, \ldots, q\},\ c_1, \ldots, c_p \in \mathcal{C}$.

Similarly, trace variable $front(\mathsf{T})$ takes one of the following forms:

1. the bottom (undefined) value $\perp_{\mathsf{T}}$ if and only if "$\mathsf{T} = [\ ]$",
2. T' if and only if "$\mathsf{T} = \mathsf{T}'.\alpha_j(c_1, \ldots, c_p)$".

The denotation of constant c under environment τ and trace T is provided by function $\mathcal{F}$, whereas the denotation of variable x under environment τ and trace T is provided by environment τ.

As a means to calculate the denotation of functional terms "$g\,(v_1, \ldots, v_l)$" under environment τ and trace T, we need to apply the standard denotation of function identifier g (as

provided by function $\mathcal{F}$) to the vector carrying the denotations of type (1) value expressions $v_1, \ldots, v_l$ computed inductively under environment τ and trace T.

As for the denotation of *attribute function* call "$f_h\,(\mathsf{T}, v_1, \ldots, v_s)$" for $h < i$ under environment τ and trace T, EB^3 type (1) value expression w_h^j is selected among EB^3 type (1) value expressions $w_h^1, \ldots, w_h^q$ appearing in *attribute function* definition A_h (see Definition 2.2.3 for details). Then, the calculation reduces to the denotation of type (1) value expression w_h^j under environment $\tau^\star(\mathsf{T}) \cup \tau'$ that assigns:

- value $\mathcal{F}(c_k)$ to formal parameter z_k of action label a_j for all $k \in \{1, \ldots, p\}$ (see Definition 2.3.4 for details on trace environment $\tau^\star$), and
- value $[\![v_k]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T})$ (computed inductively beforehand) to *attribute parameter* y_k of f_h for all $k \in \{1, \ldots, s\}$ (see definition of environment τ').

and trace T.

Notation $\perp^{\mathcal{F}}$ is syntactic sugar for $\mathcal{F}(\perp)$ and stands for the denotation of bottom (undefined) value $\perp$ under function $\mathcal{F}$. A more accurate definition would involve the introduction of symbol $\perp_T^{\mathcal{F}}$, where T stands for the return-type of *attribute function* name f_h (see Definition 2.2.3 above).

Let the trace variable T be equal to [ ]. Then, the denotation of *attribute function* call "$f_h\ (\mathit{front}(\mathsf{T}), v_1, \ldots, v_s)$" for $h > 0$ under given environment τ and the empty trace [ ] is equal to $\perp^{\mathcal{F}}$. Moreover, if "$\mathsf{T} = \mathsf{T}'.\ \alpha_j(c_1, \ldots, c_p)$" for some $j \in \{1, \ldots, q\}$ and some constants $c_1, \ldots, c_p \in \mathcal{C}$, the computation reduces to the denotation of "$f_h\,(\mathsf{T}, v_1, \ldots, v_s)$" under environment τ and trace T.

We recall that, as a means to mark the distinction between conditional value expressions and non-conditional value expressions, we assume that function identifier **if-then-else** is not generated by set $\mathcal{V}$. For this reason, the denotation of **if-then-else** value expressions and the denotation of function terms $g\,(v_1, \ldots, v_l)$ are provided separately in Definition 2.3.5. As a means to define the denotation of "**if** $g\,(v_1, \ldots, v_l)$ **then** v_{l+1} **else** v_{l+2} **end if**" under environment τ and system trace T, the classic interpretation of Boolean expressions is applied. Note that false refers to the standard denotation of Boolean constant *false*. Furthermore, the restriction of function "$[\![v_{l+2}]\!]_1^{\mathsf{T}} : \mathit{Env}_\pi \times \mathcal{T} \to D$" to "$[\![v_{l+2}]\!]_1^{\mathsf{T}} : \mathit{Env}_\pi \times \mathcal{T}' \to D$" is assumed, where $\mathcal{T}'$ is the subset of system traces, under which condition $g(v_1, \ldots, v_l)$ is evaluated to false. Hence, for those traces $\mathsf{T} \in \mathcal{T}'$, the computation is reduced to the denotation of v_{l+2} under trace T and environment τ. Otherwise, it is reduced to the denotation of v_{l+1} under trace T and environment τ.

In order to keep the presentation simple, case "$|\_ : w_i^{q+1}$" has not been treated in Definition 2.3.5. However, the power of this construct is not lost, as the user can embed the functionality of w_i^{q+1} in one of the preceding type (1) value expressions $w_i^0, \ldots, w_i^q$ (see Definition 2.2.3).

Another convention has also been made to simplify the definition. In particular, pattern matching constructs z_j are treated plainly as variables $x \in \mathcal{V}$, despite the fact that they may be reduced to value expressions v (see Figure 2.1). Again, the loss in expressiveness is minor, as the effect of v can be transferred to the corresponding w_i^j.

The standard approach of denotational semantics can be applied to EB^3 type (2) value expressions and guarded expressions too. In the following, we use notation $[\![\cdot]\!]_2^{\mathsf{T}}$ to refer to the denotation of EB^3 type (2) value expressions with respect to $\mathit{Sem}_{\mathsf{T}}$ as EB^3 type (2) value expressions constitute a subset of EB^3 type (1) value expressions.

Definition 2.3.6. *Let EB^3 be a well-formed* EB^3 *specification. The semantics (denotation) $[\![\cdot]\!]_2^{\mathsf{T}} : \lfloor T \rfloor_D$ for some type $T \in Typ$ of* EB^3 *type (2) value expressions $u \leftarrow_2 T$ appearing in EB^3 for some $i \in \{1, \ldots, n\}$ and some $j \in \{0, \ldots, q\}$ with respect to Sem_{T} is defined by way of structural induction over u as follows:*

$$\begin{aligned} [\![c]\!]_2^{\mathsf{T}} &= \mathcal{F}(c) \\ [\![g\,(u_1, \ldots, u_l)]\!]_2^{\mathsf{T}} &= \mathcal{F}(g)([\![u_1]\!]_2^{\mathsf{T}},\ \ldots,\ [\![u_l]\!]_2^{\mathsf{T}}) \end{aligned}$$

Function $[\![\cdot]\!]_2^{\mathsf{T}}$ does not depend on any given environment or any given trace, since EB^3 type (2) value expressions consist of constants $c \in \mathcal{C}$ and simple operations on constants. Note also that the set of EB^3 type (2) value expressions is a subset of EB^3 type (1) value expressions. This fact implies an alternative definition for $[\![\cdot]\!]_2^{\mathsf{T}}$ with the use of $[\![\cdot]\!]_1^{\mathsf{T}}$.

Definition 2.3.7. *Let EB^3 be a well-formed* EB^3 *specification and let T denote the current trace. The semantics (denotation) $[\![\cdot]\!]_3^{\mathsf{T}} : \mathcal{T} \to \lfloor T \rfloor_D$ for some type $T \in Typ$ of guards appearing in guarded expressions ge of EB^3 with respect to Sem_{T} under trace T is defined by way of structural induction over ge as follows:*

$$\begin{aligned} [\![c]\!]_3^{\mathsf{T}}(\mathsf{T}) &= \mathcal{F}(c) \\ [\![g\,(ge_1, \ldots, ge_l)]\!]_3^{\mathsf{T}}(\mathsf{T}) &= \mathcal{F}(g)([\![ge_1]\!]_3^{\mathsf{T}}(\mathsf{T}),\ \ldots,\ [\![ge_l]\!]_3^{\mathsf{T}}(\mathsf{T})) \\ [\![f_i\,(\mathsf{T},\, u_1, \ldots, u_s)]\!]_3^{\mathsf{T}}(\mathsf{T}) &= [\![f_i\,(\mathsf{T},\, u_1, \ldots, u_s)]\!]_1^{\mathsf{T}}([\,],\ \mathsf{T}) \end{aligned}$$

The definition of function $[\![\cdot]\!]_3^{\mathsf{T}}$ makes use of function $[\![\cdot]\!]_1^{\mathsf{T}}$ under the empty environment and the current trace T, since type (2) value expressions $u_1, \ldots, u_p$ are evaluated later on by $[\![\cdot]\!]_1^{\mathsf{T}}$ (see Definition 2.3.5 for details). For the same reason, function $[\![\cdot]\!]_2^{\mathsf{T}}$ is not applied at this point for $u_1, \ldots, u_p$. Note also that *attribute function* calls of the form "f_i ($front(\mathsf{T})$, $v_1, \ldots, v_s$)" cannot appear syntactically in *guards* ge for any $i > 0$ (see Figure 2.1 for details).

The operational semantics of EB^3 process expressions with respect to Sem_{T} is presented in Figure 2.3; it borrows heavily from the classic operational semantics of [FSt03]. The difference, here, lies in the addition of *attribute functions* and their effect on the reduction rules. Notice that the trace T is given explicitly in the system state.

- The system state with respect to Sem_{T} is represented by tuple (E, T), where E is the process expression describing the remaining system behaviour and T is the current trace.
- The initial system state is represented by process expression *main* and empty trace [ ], i.e. the tuple $(main, [\,])$.
- The evolution of system state modelled by a labelled transition system (LTS) must adhere to the reduction rules of Figure 2.3.

More details on the LTS construction of EB^3 specifications with respect to Sem_{T} can be found in Section 2.4.1, where the bisimulation equivalence of Sem_{T}, $Sem_{\mathsf{T/M}}$, and Sem_{M} is established.

Regarding Figure 2.3, the symbol $\surd$, which is not part of the user syntax, denotes successful execution. The *trace* T of an EB^3 specification at a given moment consists of the sequence of visible actions executed since the start of the system. As a result, idle action λ does not appear in the trace. At system start, the trace is empty. We also remark that if T denotes the current trace and action $\rho \in \{\alpha_j(c_1, \ldots, c_p) \mid j \in 1..q\} \cup \lambda$, where $c_1, \ldots, c_p \in \mathcal{C}$ are constants and $j \in \{1, \ldots, q\}$, can be executed, then $\mathsf{T}.\rho$ denotes the trace just after executing ρ.

$$\rho \in \{\alpha_j(c_1, \ldots, c_p) \mid j \in 1..q\} \cup \lambda$$

$$(\mathrm{T}_1)\ \frac{}{(\rho, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (\surd, \mathsf{T}.\ \rho)}\rho \neq \lambda \qquad (\mathrm{T}'_1)\ \frac{}{(\rho, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd, \mathsf{T})}$$

$$(\mathrm{T}_2)\ \frac{(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1, \mathsf{T}.\ \rho)}{(E_1.E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1.E_2, \mathsf{T}.\ \rho)} \qquad (\mathrm{T}_3)\ \frac{(E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_2, \mathsf{T}.\ \rho)}{(\surd.E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_2, \mathsf{T}.\ \rho)}$$

$$(\mathrm{T}_4)\ \frac{(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1, \mathsf{T}.\ \rho)}{(E_1 \mid E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1, \mathsf{T}.\ \rho)}$$

$$(\mathrm{T}_5)\ \frac{}{(E_0{}^*, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd, \mathsf{T})} \qquad (\mathrm{T}_6)\ \frac{(E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_0, \mathsf{T}.\ \rho)}{(E_0{}^*, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_0.E_0{}^*, \mathsf{T}.\ \rho)}$$

$$(\mathrm{T}_7)\ \frac{(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1, \mathsf{T}.\ \rho) \qquad (E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_2, \mathsf{T}.\ \rho)}{(E_1 \mid\![\Delta]\!\mid E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1 \mid\![\Delta]\!\mid E'_2, \mathsf{T}.\ \rho)}\ in\,(\rho, \Delta)$$

$$(\mathrm{T}_8)\ \frac{(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1, \mathsf{T}.\ \rho)}{(E_1 \mid\![\Delta]\!\mid E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1 \mid\![\Delta]\!\mid E_2, \mathsf{T}.\ \rho)}\ \neg in\,(\rho, \Delta)$$

$$(\mathrm{T}_9)\ \frac{}{(\surd \mid\![\Delta]\!\mid \surd, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd, \mathsf{T})}$$

$$(\mathrm{T}_{10})\ \frac{(E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_0, \mathsf{T}.\ \rho)}{(ge \Rightarrow E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_0, \mathsf{T}.\ \rho)}\ [\![ge]\!]_3^{\mathsf{T}}([\,],\ \mathsf{T}) = \mathsf{true}$$

$$(\mathrm{T}_{11})\ \frac{(E[\overline{x} := \overline{u}], \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E', \mathsf{T}.\ \rho)}{(P\,(\overline{u}), \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E', \mathsf{T}.\ \rho)}\ P\,(\overline{x}) = E$$

Figure 2.3: EB^3 Trace Semantics (Sem_{T})

- Rules $(\mathrm{T}_1), (\mathrm{T}'_1)$ refer to the execution of actions. Note that in the case of the idle action λ, λ is not inserted to the trace.
- Rules $(\mathrm{T}_2), (\mathrm{T}_3)$ refer to the execution of *sequential composition* "$E_1.E_2$" of process expressions E_1 and E_2. The execution of process expression E_1 precedes the execution of process expression E_2. Once process expression E_1 has been consumed, and, therefore, reduced to expression $\surd$, the system carries on with the execution of E_2.
- Rule (T_4) treats the *non-deterministic choice*. If process expression E_1 reduces to process expression E'_1, then process expression "$E_1 \mid E_2$" reduces to process expression E'_1. The symmetric case has been omitted for brevity.
- Rules $(\mathrm{T}_5), (\mathrm{T}_6)$ refer to the *Kleene* closure "$E_0{}^*$" of process expression E_0. Rule (T_5) refers to the idle choice. The system chooses not to execute process expression E_0 (idle

move). Moreover, the idle process λ is not added to the system trace. In rule (T_6), if process expression E_0 reduces to process expression E_0', then process expression "$E_0{}^*$" is reduced to process expression "$E_0'.E_0{}^*$". The system will carry on with the execution of E_0' in subsequent steps.

- Rules (T_7), (T_8) and (T_9) refer to the *parallel* composition "$E_1 \,|[\Delta]|\, E_2$" of process expressions E_1, E_2 with synchronization on action labels $\Delta \subseteq Lab$. Rule (T_7) treats the synchronisation on action ρ for process expressions E_1, E_2. The condition "$in\,(\rho, \Delta)$" is true if and only if the label of ρ belongs to Δ. Rule (T_8) treats the reduction of process expressions E_1 via the execution of action ρ that does not belong to Δ. The symmetric rule for process expressions E_2 has been omitted for brevity. Rule (T_9) corresponds to another idle reduction denoting that if both operands of the parallel composition have been consumed, process expression $\surd|[\Delta]|\surd$ is replaced by the termination process $\surd$.

- Rule (T_{10}) corresponds to the reduction of process expressions that are prefixed by *guards*. In particular, the reduction takes place on condition that the *guarded expressions* is evaluated to true.

  The *attribute functions* that participate syntactically in guard *ge* are evaluated using the inductive definition over *attribute functions* in Definition 2.3.7 under the current trace T. Notice that the evaluation of guard *ge* and the execution of the first action ρ in E_0 are simultaneous, i.e., no action is allowed in concurrent processes in the meantime. We call this property the *guard-action atomicity*. This property is essential for consistency as, by side effects, the occurrence of actions in concurrent processes could implicitly change the value of guard *ge* before the guarded action has been executed.

Execution. As an example, we consider a four-step simulation scenario for the library management system, whose EB^3 specification is given in Figure 2.2. We assume that the library may contain at most two books and at most two members, i.e. "$BID = \{b_1, b_2\}$", and "$MID = \{m_1, m_2\}$". We also set $NbLoans = 2$. The different values of trace variable T are depicted in Figure 2.4.

	T
A	[ ]
B	$Acquire\,(b_2).Acquire\,(b_1)$
C	$Acquire\,(b_2).Acquire\,(b_1).Register\,(m_2).Register\,(m_1)$
D	$Acquire\,(b_2).Acquire\,(b_1).Register\,(m_2).Register\,(m_1).Lend\,(b_1, m_1)$

Figure 2.4: Trace for the sample execution

In particular, the system is initially at state (A), where obviously $\mathsf{T}_A = [\,]$. The library acquires books b_2, b_1 (in this order) and transits to state (B). Then, members m_2, m_1 (in this order) are registered to the library and the system transits to state (C). Lastly, member m_1 lends book b_1 and the system transits to state (D). Transition (C)→(D) entails the evaluation of the following guard:

$$(borrower\,(\mathsf{T}, b_1) = \perp) \;\wedge\; (nbLoans\,(\mathsf{T}, m_1) < 2) \tag{2.30}$$

where T corresponds to the system trace at state (C), i.e. $\mathsf{T} = \mathsf{T}_C$. Condition (2.30) illustrates the conditions under which member m_1 can lend book b_1 (notably if the book is available and the number of loans carried out by m_1 is inferior to two). Following the *attribute function* definitions of Figure 2.2, evaluating "*borrower* (T_C, b_1)" triggers the complete traversal of the whole trace, which is not the case for *attribute function* call "*nbLoans* (T_C, m_1)". Applying successively the rules of Definition 2.3.5, the result is calculated as follows:

$$
\begin{array}{rcl}
[\![\mathit{borrower}\,(\mathsf{T},\ b_1)]\!]_1^{\mathsf{T}}([\,],\ \mathsf{T}_C) & = & [\![\mathit{borrower}\,(\mathit{front}(\mathsf{T}),\ bId)]\!]_1^{\mathsf{T}}(\tau_1,\ \mathsf{T}_C) \\
& = & [\![\mathit{borrower}\,(\mathsf{T},\ bId)]\!]_1^{\mathsf{T}}(\tau_1,\ \mathsf{T}_B.\mathit{Register}\,(m_2)) \\
& = & [\![\mathit{borrower}\,(\mathit{front}(\mathsf{T}),\ bId)]\!]_1^{\mathsf{T}}(\tau_2,\ \mathsf{T}_B.\mathit{Register}\,(m_2)) \\
& = & [\![\mathit{borrower}\,(\mathsf{T},\ bId)]\!]_1^{\mathsf{T}}(\tau_2,\ \mathsf{T}_B) \\
& = & [\![\mathit{borrower}\,(\mathit{front}(\mathsf{T}),\ bId)]\!]_1^{\mathsf{T}}(\tau_3,\ \mathit{Acquire}\,(b_2)) \\
& = & [\![\mathit{borrower}\,(\mathsf{T},\ bId)]\!]_1^{\mathsf{T}}(\tau_3,\ [\,]) \\
& = & [\![\bot]\!]_1^{\mathsf{T}}([\,],\ [\,]) \\
& = & \bot^{\mathcal{F}} \\
\big(\text{where } \tau_1 = \tau_2 & = & [\ \mathsf{mId} \leftarrow \mathsf{m_1},\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ],\quad \tau_3 = [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ]\ \big) \\
[\![\mathit{nbLoans}\,(\mathsf{T},\ m_1)]\!]_1^{\mathsf{T}}\ ([\,],\ \mathsf{T}_C) & = & [\![0]\!]_1^{\mathsf{T}}([\ \mathsf{mId} \leftarrow \mathsf{m_1}\ ],\ \mathsf{T}_C) \\
& = & \mathsf{0}
\end{array}
$$

In the previous calculations, the interpretation of constant $\bot$, i.e. $\mathcal{F}(\bot)$, has been replaced by $\bot^{\mathcal{F}}$. Similarly, $\mathcal{F}(0)$ has been replaced by $\mathsf{0}$. This notation style is adopted for all constants $c \in \mathcal{C}$ used throughout this manuscript. The same convention applies to any EB^3 variable $x \in \mathcal{V}$ referred to as x in the corresponding environment.

As for the evaluation of *attribute function* call "*borrower* (T, b_1)", one must refer to the corresponding definition of *borrower* in Figure 2.2 first. Owing to matching case expression:

$$\mathsf{I}\_ : \mathit{borrower}\,\big(\mathit{front}\,(\mathsf{T}),\ bId\big),$$

type (1) value expression w_h^{q+1} (see Definition 2.2.3) matches with *borrower* $\big(\mathit{front}\,(\mathsf{T}),\ bId\big)$, expression "*last* (T)" matches with action "*Register* (m_1)" and expression "*front*(T)" matches with trace "T_B. *Register* (m_2)" or equivalently "*Acquire* (b_2). *Acquire* (b_1). *Register* (m_2)". Then, applying Definition 2.3.5, the denotation "*borrower* (T, b_1)" under environment $[\,]$ and trace T_C reduces to the denotation "*borrower* $(\mathit{front}\ (\mathsf{T}),\ bId)$" under environment:

$$\tau_1 = \tau^\star(\mathsf{T}_C) \cup [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ] = [\ \mathsf{mId} \leftarrow \mathsf{m_1},\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ],$$

and trace T_C, since formal parameter mId of action prototype definition "*Register* $(mId : MID)$" (see Figure 2.2) is assigned value $\mathsf{m_1}$ by environment function $\tau^\star$ (see Definition 2.3.4 for details) and formal parameter bId of *attribute function* name is assigned value "$\mathsf{bId} \leftarrow [\![b_1]\!]_1^{\mathsf{T}}([\,],\ \mathsf{T}_C) = \mathsf{b_1}$".

Applying Definition 2.3.5 once more, the denotation of "*borrower* $(\mathit{front}\ (\mathsf{T}),\ bId)$" under environment τ_1 and trace T_C reduces to the denotation of "*borrower* $(\mathsf{T},\ bId)$" under environment τ_1 and trace "T_B. *Register* (m_2)".

Then, owing to matching case expression:

$$\mathsf{I}\_ : \mathit{borrower}\,\big(\mathit{front}\,(\mathsf{T}),\ bId\big),$$

type (1) value expression w_h^{q+1} matches with "*borrower* $\big($*front* $(\mathsf{T}),\ bId\big)$", expression "*last* (T)" matches with action "*Register* (m_2)" and expression "*front*(T)" matches with trace T_B. Hence, applying Definition 2.3.5, the denotation of "*borrower* (T, b_1)" under environment τ_1 and trace "T_B. *Register* (m_2)" reduces to the denotation of "*borrower* (*front* $(\mathsf{T}),\ bId$)" under environment:

$$\tau_2 = \tau^\star(\mathsf{T}_C) \cup [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ] = [\ \mathsf{mId} \leftarrow \mathsf{m_1},\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ],$$

and trace "T_B. *Register* (m_2)", since formal parameter mId of action prototype definition "*Register* $(mId : MID)$" (see Figure 2.2) is assigned value $\mathsf{m_2}$ by environment function $\tau^\star$ (see Definition 2.3.4) and formal parameter bId of *attribute function* name *borrower* is assigned value:

$$\mathsf{bId} \leftarrow [\![bId]\!]_1^{\mathsf{T}}([\ ],\ \mathsf{T}_C) = \tau_1(\mathsf{bId}) = \mathsf{b_1}.$$

Applying Definition 2.3.5 once more, the denotation of "*borrower* (*front* $(\mathsf{T}),\ bId$)" under environment τ_2 and trace "T_B. *Register* (m_2)" reduces to the denotation of "*borrower* $(\mathsf{T},\ bId)$" under environment τ_2 and trace "T_B".

Then, owing to matching case expression:

$$\mid\_ : \textit{borrower}\ \big(\textit{front}\ (\mathsf{T}),\ bId\big),$$

type (1) value expression w_h^{q+1} matches with "*borrower* $\big($*front* $(\mathsf{T}),\ bId\big)$", expression "*last* (T)" matches with action "*Acquire* (b_1)" and special expression "*front*(T)" matches with trace "*Acquire* (b_2)". Hence, applying Definition 2.3.5, the denotation of "*borrower* (T, b_1)" under environment τ_2 and trace "T_B" reduces to the denotation of "*borrower* (*front* $(\mathsf{T}),\ bId$)" under environment:

$$\tau_3 = \tau^\star(\mathsf{T}_C) \cup [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ] = [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ],$$

and trace "*Acquire* (b_2)", since parameter bId of action prototype definition "*Acquire* $(bId : MID)$" (see Figure 2.2) is assigned value $\mathsf{b_1}$ by environment function $\tau^\star$ (see Definition 2.3.4) and *formal parameter* bId of *attribute function* name *borrower* is assigned value "$\mathsf{bId} \leftarrow [\![bId]\!]_1^{\mathsf{T}}([\ ],\ \mathsf{T}_C) = \tau_2(\mathsf{bId}) = \mathsf{b_1}$".

Notice that variable identifier bId belonging to action "*Acquire*"'s *formal parameters* coincides with *formal parameter* bId of *attribute function* name "*borrower*" producing duplicate assignments for bId, which can be removed from the environment without complication. In general, the process of duplicate variable renaming is applied to resolve such issues.

Applying Definition 2.3.5 once more, the denotation of "*borrower* (*front* $(\mathsf{T}),\ bId$)" under environment τ_3 and trace "*Acquire*(b_2)" reduces to the denotation of "*borrower* $(\mathsf{T},\ bId)$" under environment τ_3 and empty trace [ ].

Finally, owing to matching case expression:

$$\mid \perp_{\mathsf{T}} : \perp$$

value expression w_h^0 matches with $\perp$. Hence, applying Definition 2.3.5, the denotation of "*borrower* (*front* $(\mathsf{T}),\ bId$)" under environment τ_3 and trace [ ] reduces to the denotation of $\perp$ under environment $\tau^\star([\ ]) \oplus [\ ] = [\ ]$ and empty trace [ ], which results in $\perp^{\mathcal{F}}$.

For the evaluation of *attribute function* call "*nbLoans* (T_C, m_1)", one may refer to the corresponding definition of *nbLoans* in Figure 2.2. Owing to the matching case expression:

$$\mid \textit{Register}\ (m_1)\ : 0, \tag{2.31}$$

expression "*last*(T)" matches with action "*Register* (m_1)", type (1) value expression w_h^2 matches with 0. Case expression (2.31) is the 3rd case expression of *attribute function* name *nbLoans* in Figure 2.2, which justifies the superscript of w_h^2. Hence, applying Definition 2.3.5, the denotation of "*nbLoans* (T_C, m_1)" under environment [ ] and trace T_C reduces to the denotation of 0 under environment:

$$\tau^\star(\mathsf{T}_C) \cup [\ \mathsf{mId} \leftarrow [\![m_1]\!]^1_{\mathsf{T}}\ ([\ ],\ \mathsf{T}_C)\ ] = [\ \mathsf{mId} \leftarrow \mathsf{m_1}\ ]$$

and trace T_C, since value "$[\![m_1]\!]^1_{\mathsf{T}}\ ([\ ],\ \mathsf{T}_C) = \mathsf{m_1}$" is assigned to formal parameter mId of *nbLoans* that coincides with formal parameter mId of action name *Register*. Remark that duplicate entries have been removed as previously.

Finally, by Definition 2.3.7, guard (2.30) is evaluated as follows:

$$\begin{gathered}
[\![(\mathit{borrower}\,(\mathsf{T}, b_1) = \bot)\ \wedge\ (\mathit{nbLoans}\,(\mathsf{T}, m_1) < 2)]\!]^{\mathsf{T}}_3(\mathsf{T}_C) = \\
[\![(\mathit{borrower}\,(\mathsf{T}, b_1) = \bot)]\!]^{\mathsf{T}}_3(\mathsf{T}_C)\ \wedge\ [\![(\mathit{nbLoans}\,(\mathsf{T}, m_1) < 2)]\!]^{\mathsf{T}}_3(\mathsf{T}_C) = \\
[\![(\mathit{borrower}\,(\mathsf{T}, b_1) = \bot)]\!]^{\mathsf{T}}_1([\ ],\ \mathsf{T}_C)\ \wedge\ [\![(\mathit{nbLoans}\,(\mathsf{T}, m_1) < 2)]\!]^{\mathsf{T}}_1([\ ],\ \mathsf{T}_C) = \\
(\bot^{\mathcal{F}} =^{\mathcal{F}} \bot^{\mathcal{F}})\ \wedge\ (0 <^{\mathcal{F}} 2) = \\
\mathsf{true}.
\end{gathered}$$

Notation $=^{\mathcal{F}}$ denotes the classical interpretation of equation as provided by function $\mathcal{F}$ in environment τ. Similar is the meaning of notation $<^{\mathcal{F}}$.

Figure 2.5 depicts the evolution of process expression *main* describing the library management system in states (A), (B), (C), and (D).

2.3.2 Trace/Memory Semantics $\boldsymbol{Sem}_{\mathsf{T/M}}$

Let EB^3 be a *well-formed* EB^3 specification, whose syntax conforms to the syntactic patterns of Figure 2.1. As a means to define Trace/Memory Semantics ($Sem_{\mathsf{T/M}}$), *attribute function* names f_i appearing in EB^3 are turned into state variables $\mathsf{f_i}$ for $i \in \{1, \ldots, n\}$, which we call *attribute variables*, carrying the effect of the system trace on their corresponding values. This avoids keeping the (ever-growing) trace, thus, leading to a finite state model.

Hence, if "$f_i\,(\mathsf{T}, x_1 : T_1, \ldots, x_s : T_s) : T$" is the prototype of *attribute function* f_i (see Figure 2.1 for details) and $|C_k|$ stands for the cardinality of set:

$$C_k = \{c \in \mathcal{C} \mid \sigma(c) = T_k\} \tag{2.32}$$

for index $k \in \{1, \ldots, s\}$, we construct in total:

$$|C_1| \times \ldots \times |C_s|$$

state variables "$\mathsf{f_i} : (T_1, \ldots, T_s) \rightarrow T$" for $i \in \{1, \ldots, n\}$.

The different sets of *attribute variables* and their associated values are treated as special environments that we call *attribute variable* environments. In the following, *attribute variable* environments are given separately from ordinary environments, i.e. environments of *attribute parameters* and formal parameters of action labels. In particular, an *attribute variable* environment M shall contain *attribute variables* $\mathsf{f_i}$ and their corresponding values denoted as $\mathsf{M}(\mathsf{f_i})$, as well as primed *attribute variables* $\mathsf{f_i'}$ and their corresponding values denoted as $\mathsf{M}(\mathsf{f_i'})$. The infinite domain of *attribute variable* environments is denoted as $\mathcal{M}$. In the following, we use the term EB^3 memory to refer to *attribute variable* environments.

$main$ (A)

$\xrightarrow{Acquire\,(b_2).\ Acquire\,(b_1)}$

$borrower\,(\mathsf{T}_B, b_1) = \bot \rightarrow Discard\,(b_1).\ book\,(b_1)^*\ |||$
$borrower\,(\mathsf{T}_B, b_2) = \bot \rightarrow Discard\,(b_2).\ book\,(b_2)^*\ |||$
$(\ ||| \ mId : MID : member\,(mId)^*\ )$ (B)

$\xrightarrow{Register\,(m_2).\ Register\,(m_1)}$

$borrower\,(\mathsf{T}_C, b_1) = \bot \rightarrow Discard\,(b_1).\ book\,(b_1)^*\ |||$
$borrower\,(\mathsf{T}_C, b_2) = \bot \rightarrow Discard\,(b_2).\ book\,(b_2)^*\ |||$
$(\ ||| \ bId : BID : loan\,(m_1, bId)^*\ ).\ Unregister\,(m_1).\ member\,(m_1)^*\ |||$
$(\ ||| \ bId : BID : loan\,(m_2, bId)^*\ ).\ Unregister\,(m_1).\ member\,(m_2)^*$ (C)

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

$borrower\,(\mathsf{T}_D, b_1) = \bot \rightarrow Discard\,(b_1).\ book\,(b_1)^*\ |||$
$borrower\,(\mathsf{T}_D, b_2) = \bot \rightarrow Discard\,(b_2).\ book\,(b_2)^*\ |||$
$(Return(b_1).\ loan(m_1, b_1)^*\ ||| \ loan(m_1, b_2)^*).\ Unregister\,(m_1).\ member\,(m_1)^*\ |||$
$(\ ||| \ bId : BID : loan\,(m_2, bId)^*).\ Unregister\,(m_1).\ member\,(m_2)^*$ (D)

Figure 2.5: Sample execution

As usual, we consider actions of the form $\alpha_j\,(\overline{c})$, where $\overline{c} = (c_1, \ldots, c_p)$ and constants $c_1, \ldots, c_p \in \mathcal{C}$. M' shall denote the EB^3 memory after action $\alpha_j\,(\overline{c})$ has been executed. By definition, the inert action λ has no effect on M, i.e. $\mathsf{M}' = \mathsf{M}$. We denote by M_0 the *attribute variables* at system start, i.e. the *attribute variables* corresponding to the empty trace, i.e. $\mathsf{T} = [\,]$.

Definition 2.3.8. *Let EB^3 be a well-formed EB^3 specification, let $\tau \in Env_\pi$ be a π-compatible environment, and let M be an EB^3 memory for EB^3, then the semantics $[\![\cdot]\!]_1^{\mathsf{M}} : Env_\pi \times \mathcal{M} \rightarrow \lfloor T \rfloor_D$ for some type $T \in Typ$ of EB^3 type (1) value expressions $w_i^j \leftharpoonup T$ for $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, q\}$ with respect to $Sem_{\mathsf{T}/\mathsf{M}}$ under environment τ, and EB^3 memory M is defined*

by way of structural induction over w_i^j *as follows:*

$$\begin{aligned}
[\![c]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) &= \mathcal{F}(c) && (2.33)\\
[\![x]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) &= \tau(x) && (2.34)\\
[\![g\,(v_1,\ldots,v_l)]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) &= \mathcal{F}(g)([\![v_1]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}),\ldots,[\![v_l]\!]_1^{\mathsf{M}}(\tau,\mathsf{M})) && (2.35)\\
[\![f_h\,(front\ (\mathsf{T}),v_1,\ldots,v_s)]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) &= \mathsf{M}(\mathsf{f_h})([\![v_1]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}),\ldots,[\![v_s]\!]_1^{\mathsf{M}}(\tau,\mathsf{M})) && (2.36)\\
[\![f_h\,(\mathsf{T},v_1,\ldots,v_s)]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) &= \mathsf{M}(\mathsf{f_h}')([\![v_1]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}),\ldots,[\![v_s]\!]_1^{\mathsf{M}}(\tau,\mathsf{M})) && (2.37)\\
[\![\textit{if}\ g(v_1,\ldots,v_l)\ \textit{then}\ v_{l+1} && && (2.38)\\
\quad \textit{else}\ v_{l+2}\ \textit{end if}]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) &= ([\![v_{l+1}]\!]_1^{\mathsf{M}} \oplus [\![v_{l+2}]\!]_1^{\mathsf{M}})(\tau,\ \mathsf{M}) &&\\
\quad \textit{where}\quad [\![v_{l+2}]\!]_1^{\mathsf{M}} &: Env_\pi \times \mathcal{M}' \to \lfloor T \rfloor &&
\end{aligned}$$

$$\textit{and}\quad \mathcal{M}' = \{\ \mathsf{M}\in\mathcal{M}\ \mid\ [\![g(v_1,\ldots,v_l)]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) = \mathsf{false}\}.$$

Only the interesting parts of Definition 2.3.8 are commented below. The denotation of *attribute function* call "$f_h\,(front\ (\mathsf{T}),\ v_1,\ldots,v_s)$" for $h>0$ (see *attribute function ordering* in Section 2.1) under environment τ and EB^3 memory M reduces to:

$$\mathsf{M}(\mathsf{f_h})([\![v_1]\!]_1^{\mathsf{M}}(\tau,\ \mathsf{M}),\ \ldots,\ [\![v_s]\!]_1^{\mathsf{M}}\ (\tau,\ \mathsf{M}))\quad \text{for } h>0,$$

denoting the value of *attribute variable* $\mathsf{f_h}$ in M, whose parameter vector is the vector carrying the assigned denotations to $v_1,\ldots,v_s$ computed inductively beforehand under environment τ and EB^3 memory M.

Furthermore, the denotation of *attribute function* call "$f_h\,(\mathsf{T},\ v_1,\ldots,v_s)$" for $h<i$ (see *attribute function ordering* in Section 2.1) under environment τ and EB^3 memory M reduces to:

$$\mathsf{M}(\mathsf{f_h'})(\ [\![v_1]\!]_1^{\mathsf{M}}\ (\tau,\ \mathsf{M}),\ \ldots,\ [\![v_s]\!]_1^{\mathsf{M}}\ (\tau,\ \mathsf{M}))\quad \text{for } h<i,$$

denoting the value of primed *attribute variable* $\mathsf{f_h}'$ that has been previously stocked in M, whose parameter vector is the vector carrying the assigned denotations to $v_1,\ldots,v_s$ computed inductively beforehand under environment τ and EB^3 memory M. For more details on the order in which *attribute variables* are modified, see Definition 2.3.10, which stipulates that *attribute variables* of lower order, i.e. for $h<i$, are modified first.

As a means to define the denotation of "**if** $g\,(v_1,\ldots,v_l)$ **then** v_{l+1} **else** v_{l+2} **end if**" under environment τ and *attribute variables* M, the restriction of function "$[\![v_{l+2}]\!]_1^{\mathsf{M}} : Env_\pi \times \mathcal{M} \to \lfloor T \rfloor$" to "$[\![v_{l+2}]\!]_1^{\mathsf{M}} : Env_\pi \times \mathcal{M}' \to \lfloor T \rfloor$" is assumed, where $\mathcal{M}'$ is the subset of *attribute variables* M, under which condition $g\,(v_1,\ldots,v_l)$ is evaluated to false. Hence, for *attribute variables* $\mathsf{M}\in\mathcal{M}'$, the calculation is reduced to the denotation of v_{l+2} under environment τ and EB^3 memory M. For EB^3 memory $\mathsf{M}\notin\mathcal{M}'$, it is reduced to the denotation of v_{l+1} under environment τ and *attribute variables* M.

The calculations involving the initialisation and modification of *attribute variables* are thoroughly discussed in Definitions 2.3.9 and 2.3.10. Remark also that as a means not to overcharge the formulas of Definition 2.3.8, notation $\mathsf{f_h}$ has been used to denote both the *attribute variable* and the corresponding interpretation. Similar conventions have been made for the primed *attribute variables* $\mathsf{f_h'}$.

Definition 2.3.9. *Let EB^3 be a well-formed* EB^3 *specification and let $\tau \in Env_\pi$ be a π-compatible environment. The computation of* EB^3 *memory $\mathsf{M_0} \in \mathcal{M}$ at system start, i.e. $\mathsf{T} = [\ ]$, is carried out as follows:*

function $upd_0 : \mathcal{M}$
  $\mathsf{M_0} = [\ \mathsf{f_i}\,[\mathsf{c_1},\ \ldots,\ \mathsf{c_s}] \leftarrow \perp^{\mathcal{F}} \quad | \quad \forall c_1 \in C_1, \ldots, \forall c_s \in C_s\ ];$
  for $i := 1$ ***to*** n ***do***
    for $(c_1, \ldots, c_s) \in (C_1 \times \ldots \times C_s)$ ***do***
      $\tau' = [\ \mathsf{y_1} \leftarrow \mathsf{c_1},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{c_s}\ ];$
      $\mathsf{w_i^0} = [\![w_i^0]\!]_1^{\mathsf{M}}\ (\tau',\ \mathsf{M_0});$
      $\mathsf{M_0} = \mathsf{M_0} \oplus \left[\ \mathsf{f_i'}\,[\mathsf{c_1},\ \ldots,\ \mathsf{c_s}] \leftarrow \mathsf{w_i^0}\ \right]$
    end for
  end for;
  return M_0;
end function

In particular, EB^3 memory $\mathsf{M_0}$ is created associating the bottom value $\perp_{\mathcal{F}}$ to every *attribute variable* $\mathsf{f_i}\,[\mathsf{c_1},\ \ldots,\ \mathsf{c_s}]$, where constant $c_k \in C_k$ for $k \in \{1, \ldots, s\}$. We recall that C_k is defined in (2.32). Index i is initially set to 1.

A tuple $(c_1, \ldots, c_s)$ belonging to $(C_1 \times \ldots \times C_s)$ is chosen. Environment τ' is created associating $\mathsf{c_k}$ to *attribute parameter* y_k of *attribute function* name f_i (see Definition 2.2.3 for details). Type (1) value expression w_i^0 referring to the empty trace is selected among value expressions $w_i^0, \ldots, w_i^q$ appearing in *attribute function* definition A_i. Then, the denotation of w_i^0 under environment τ' (see Definition 2.3.8) and EB^3 memory $\mathsf{M_0}$ is computed. The result is associated to *attribute variable* $\mathsf{f_i'}\,[\mathsf{c_1},\ \ldots,\ \mathsf{c_s}]$ and EB^3 memory $\mathsf{M_0}$ is updated accordingly. The procedure is repeated iteratively for the next tuple $(c_1, \ldots, c_s)$ in $(C_1 \times \ldots \times C_s)$ until all tuples $(c_1, \ldots, c_s)$ have been explored. Then, index i is incremented by one and the previous procedure is repeated iteratively until the point where i has reached value n.

In the previous calculations, we assume that the current values of *attribute variables* $\mathsf{f_i}$ co-exist in the corresponding instances of EB^3 memory $\mathsf{M_0}$ with the primed versions of *attribute variables* $\mathsf{f_i'}$. As soon as the computations are completed, primed values $\mathsf{f_i'}$ are turned into $\mathsf{f_i}$ and primed *attribute variables* $\mathsf{f_i'}$ of $\mathsf{M_0}$ are made redundant. Finally, function upd_0 returns $\mathsf{M_0}$, which is the result in question.

Definition 2.3.10. *Let EB^3 be a well-formed* EB^3 *specification and let $\tau \in Env_\pi$ denote a π-compatible environment. Let also "$\mathsf{T} = \mathsf{T}'.\rho$" denote a trace, where T' stands for a valid trace. Let M denote the* EB^3 *memory of EB^3 related to trace T' and let M' denote the* EB^3 *memory after action ρ has been executed.*

The computation of EB^3 *memory* M' *is carried out as follows:*

$$
\begin{array}{l}
\textbf{\textit{function}}\ upd\ (\rho : Act,\ \mathsf{M} : \mathcal{M}) : \mathcal{M} \\
\quad \mathsf{M}' = \mathsf{M}; \\
\quad \textbf{\textit{match}}\ \rho\ \textbf{\textit{with}} \\
\quad\quad \mid\ \alpha_j\ (d_1, \ldots, d_p) \Rightarrow \\
\quad\quad\quad \tau = [\ \mathsf{z_1} \leftarrow \mathsf{d_1},\ \ldots,\ z_p \leftarrow \mathsf{d_p}\ ]; \\
\quad\quad\quad \textbf{\textit{for}}\ i := 1\ \textbf{\textit{to}}\ n\ \textbf{\textit{do}} \\
\quad\quad\quad\quad \textbf{\textit{for}}\ (c_1, \ldots, c_s) \in (C_1 \times \ldots \times C_s)\ \textbf{\textit{do}} \\
\quad\quad\quad\quad\quad \tau' = [\ \mathsf{y_1} \leftarrow \mathsf{c_1},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{c_s}\ ]; \\
\quad\quad\quad\quad\quad \mathsf{w_i^j} = [\![w_i^j]\!]_1^{\mathsf{M}}\ (\tau \cup \tau',\ \mathsf{M}'); \\
\quad\quad\quad\quad\quad \mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{f_i'}\,[\mathsf{c_1},\ \ldots,\ \mathsf{c_s}] \leftarrow \mathsf{w_i^j}\ ] \\
\quad\quad\quad\quad \textbf{\textit{end for}} \\
\quad\quad\quad \textbf{\textit{end for}} \\
\quad\quad \textbf{\textit{end match}}; \\
\quad\quad \textbf{\textit{return}}\ \mathsf{M}'; \\
\textbf{\textit{end function}}
\end{array}
$$

In particular, action ρ and EB^3 memory M are passed to function *upd* as parameters. M' is initially set to M. Then, action ρ is matched (by pattern-matching) to action $\alpha_j(d_1, \ldots, d_p)$ for some $j \in \{1, \ldots, q\}$ and some constants $d_1, \ldots, d_p \in \mathcal{C}$. Environment τ is created associating $\mathsf{d_k}$ to formal parameter z_k belonging to action label α_j for $k \in \{1, \ldots, p\}$. Index i is initially set to 1 and environment τ' is created associating $\mathsf{c_k}$ to *attribute parameter* y_k referring to *attribute function* name f_i.

A tuple $(c_1, \ldots, c_s)$ belonging to $(C_1 \times \ldots \times C_s)$ is chosen. Then, type (1) value expression w_i^j is selected among type (1) value expressions $w_i^0, \ldots, w_i^q$ appearing in *attribute function* definition A_i (see Definition 2.2.3 for details). The denotation of w_i^j under environment $\tau \cup \tau'$ and EB^3 memory M' is calculated (see Definition 2.3.8 for details). The result is associated to primed *attribute variable* $\mathsf{f_i'}\,[\mathsf{c_1},\ \ldots,\ \mathsf{c_s}]$ and EB^3 memory M' is updated accordingly. The procedure is repeated iteratively for the next tuple $(c_1, \ldots, c_s)$ in $(C_1 \times \ldots \times C_s)$ until all tuples $(c_1, \ldots, c_s)$ have been explored. Then, index i is incremented by one and the previous procedure is repeated iteratively until the point where i has reached value n.

Last but not least, we assume that all primed values $\mathsf{f_i'}$ in M' are turned into $\mathsf{f_i}$ and that all primed *attribute variables* $\mathsf{f_i'}$ in M' are made redundant. Finally, function *upd* returns M', which is the result in question.

Notice that the previous calculations in Definition 2.3.10 entail no complex reasoning on the system trace. In particular, only the head of the current trace, i.e. action $\alpha(c_1, \ldots, c_p)$, is involved. Other objects taking part in the corresponding calculations include the current *attribute variables* $\mathsf{f_i}$ included in EB^3 memory M and the primed *attribute variables* $\mathsf{f_i'}$ included in M.

In the following, we use the symbol $[\![\cdot]\!]_1^{\mathsf{M}}$ to refer to the denotation of EB^3 type (2) value

expressions as EB^3 type (2) value expressions constitute a subset of EB^3 type (1) value expressions.

Definition 2.3.11. *Let EB^3 be a well-formed* EB^3 *specification. The semantics $[\![\cdot]\!]_1^{\mathsf{M}} : \lfloor T \rfloor_D$ of* EB^3 *type (2) value expressions u appearing in EB^3 such that $u \leftarrow_2 T$ for some type $T \in Typ$ with respect to $Sem_{\mathsf{T/M}}$ is defined with structural induction on u as follows:*

$$\begin{aligned} [\![c]\!]_2^{\mathsf{M}} &= \mathcal{F}(c) \\ [\![g\,(u_1, \ldots, u_l)]\!]_2^{\mathsf{M}} &= \mathcal{F}(g)([\![u_1]\!]_2^{\mathsf{M}},\ \ldots,\ [\![u_l]\!]_2^{\mathsf{M}}) \end{aligned}$$

Notice that the denotation of u depends neither on the current system trace nor on any EB^3 memory.

Definition 2.3.12. *Let EB^3 be a well-formed* EB^3 *specification and let* T *denote the current system trace. Let also* M *denote the* EB^3 *memory of EB^3 that is related to* T*. The semantics $[\![\cdot]\!]_3^{\mathsf{M}} : \mathcal{M} \to \lfloor T \rfloor_D$ of* EB^3 *guards $ge \leftarrow_3 T$ for some $T \in Typ$ appearing in* EB^3 *guarded expressions with respect to $Sem_{\mathsf{T/M}}$ under* M *is recursively defined as follows:*

$$\begin{aligned} [\![c]\!]_3^{\mathsf{M}}(\mathsf{M}) &= \mathcal{F}(c) \\ [\![g\,(ge_1, \ldots, ge_l)]\!]_3^{\mathsf{M}}(\mathsf{M}) &= \mathcal{F}(g)([\![ge_1]\!]_3^{\mathsf{M}}(\mathsf{M}), \ldots, [\![ge_l]\!]_3^{\mathsf{M}}(\mathsf{M})) \\ [\![f_h\,(\mathsf{T},\, u_1, \ldots, u_s)]\!]_3^{\mathsf{M}}(\mathsf{M}) &= \mathsf{M}(\mathsf{f_h})([\![u_1]\!]_1^{\mathsf{M}}, \ldots, [\![u_s]\!]_1^{\mathsf{M}}) \text{ for } h > 0 \end{aligned}$$

Notice that the denotation of $[\![f_h\,(\mathsf{T},\, u_1, \ldots, u_s)]\!]_3^{\mathsf{M}}$ reduces to *attribute variable* $\mathsf{f_h}$ in EB^3 memory M applied to the vector consisting of the denotations for EB^3 type (2) value expressions $u_1, \ldots, u_s$. The trace plays no role in the previous computations whatsoever.

The operational semantics of EB^3 process expressions with respect to $Sem_{\mathsf{T/M}}$ is defined in Figure 2.6.

- The system state with respect to $Sem_{\mathsf{T/M}}$ is represented by tuple $(E, \mathsf{T}, \mathsf{M})$, where E is the process expression describing the remaining system behaviour, T stands for the current trace and M is the current EB^3 memory.
- The initial system state is represented by tuple $(main, [\,], \mathsf{M_0} \doteq upd_0)$ (see Definition 2.3.9 for details on upd_0).
- The evolution of system state is modelled by a labelled transition system (LTS), which adheres to the reduction rules of Figure 2.6.

Regarding Figure 2.3, we recall that if T denotes the current trace and action $\rho \in \{\alpha_j(c_1, \ldots, c_p) \mid j \in 1..q\} \cup \lambda$, where $c_1, \ldots, c_p \in \mathcal{C}$ are constants and $j \in \{1, \ldots, q\}$ is executed, then $\mathsf{T}.\rho$ shall denote the trace after ρ has been executed.

- Rules $(\mathrm{M}_1), (\mathrm{M}_1')$ refer to the execution of actions. In rule (M_1), function *upd* returns modified M based on action ρ and M (see Definition 2.3.10). Notice that the idle action λ, λ has no effect on EB^3 memory M.
- In rule (M_{10}), guard ge is evaluated according to Definition 2.3.12 under EB^3 memory M.

More details on the LTS construction of EB^3 specifications with respect to $Sem_{\mathsf{T/M}}$ can be found in Section 2.4.1, where the bisimulation equivalence of Sem_{T}, $Sem_{\mathsf{T/M}}$, and Sem_{M} is established.

$\rho \in \{\alpha_j(c_1, \ldots, c_p) \mid j \in 1..q\} \cup \lambda$, where $c_1, \ldots, c_p \in \mathcal{C}$

$$(\mathrm{M}_1)\ \frac{}{(\rho, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho \neq \lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}.\ \rho, upd(\rho, \mathsf{M}))} \qquad (\mathrm{M}'_1)\ \frac{}{(\rho, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M})}$$

$$(\mathrm{M}_2)\ \frac{(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\rho, \mathsf{M}')}{(E_1.E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1.E_2, \mathsf{T}.\rho, \mathsf{M}')} \qquad (\mathrm{M}_3)\ \frac{(E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_2, \mathsf{T}.\ \rho, \mathsf{M}')}{(\surd.E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_2, \mathsf{T}.\ \rho, \mathsf{M}')}$$

$$(\mathrm{M}_4)\ \frac{(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\ \rho, \mathsf{M}')}{(E_1 \mid E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\ \rho, \mathsf{M}')}$$

$$(\mathrm{M}_5)\ \frac{}{(E_0{}^*, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M})} \qquad (\mathrm{M}_6)\ \frac{(E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_0, \mathsf{T}.\ \rho, \mathsf{M}')}{(E_0{}^*, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_0.E_0{}^*, \mathsf{T}.\ \rho, \mathsf{M}')}$$

$$(\mathrm{M}_7)\ \frac{(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\ \rho, \mathsf{M}') \qquad (E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_2, \mathsf{T}.\ \rho, \mathsf{M}')}{(E_1 \mid\![\Delta]\!\mid E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1 \mid\![\Delta]\!\mid E'_2, \mathsf{T}.\rho, \mathsf{M}')}\ in\,(\rho, \Delta)$$

$$(\mathrm{M}_8)\ \frac{(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\ \rho, \mathsf{M}')}{(E_1 \mid\![\Delta]\!\mid E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1 \mid\![\Delta]\!\mid E_2, \mathsf{T}.\ \rho, \mathsf{M}')}\ \neg in\,(\rho, \Delta)$$

$$(\mathrm{M}_9)\ \frac{}{(\surd \mid\![\Delta]\!\mid \surd, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M})}$$

$$(\mathrm{M}_{10})\ \frac{(E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_0, \mathsf{T}.\ \rho, \mathsf{M}')}{(ge \Rightarrow E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_0, \mathsf{T}.\ \rho, \mathsf{M}')}\ [\![ge]\!]^{\mathsf{M}}_3(\mathsf{M}) = \mathsf{true}$$

$$(\mathrm{M}_{11})\ \frac{(E[\overline{x} := \overline{u}], \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E', \mathsf{T}.\ \rho, \mathsf{M}')}{(P\,(\overline{u}), \mathsf{T}, \mathsf{M}') \xrightarrow{\rho}_{\mathsf{T/M}} (E', \mathsf{T}.\ \rho, \mathsf{M}')}\ P\,(\overline{x}) = E$$

Figure 2.6: EB^3 Trace/Memory Semantics ($Sem_{\mathsf{T/M}}$)

Execution. We illustrate how the EB^3 specification of Figure 2.2 describing the library management system is executed with respect to Sem_{M}. As in Section 2.3.1, we set $m = p = NbLoans = 2$, i.e. we consider two books b_1 and b_2, and two members m_1 and m_2.

In this context, an *attribute variable* memory consists of four cells namely $\mathsf{borrower}[\mathsf{b_1}]$, $\mathsf{borrower}[\mathsf{b_2}]$, $\mathsf{nbLoans}[\mathsf{m_1}]$ and $\mathsf{nbLoans}[\mathsf{m_2}]$. The first two cells keep the two values relating to *attribute function borrower* $(\mathsf{T}, \bullet)$ for a given trace T, and the last two keep the values relating to *nbLoans* $(\mathsf{T}, \bullet)$.

We turn our attention to the execution scenario of Figure 2.5 and we consider all intermediate steps leading from state (A) to (B). The purpose is to depict the step-by-step evolution of *attribute variables*. Hence, Figure 2.4 is expanded to Figure 2.7.

- The EB^3 memory at state (A) corresponding to EB^3 memory $\mathsf{M_0}$ at system start can be calculated by way of function upd_0 of Definition 2.3.9. In particular, upon substituting

	T
A	[ ]
A'	*Acquire* (b_2)
B	*Acquire* (b_2). *Acquire* (b_1)
B'	*Acquire* (b_2). *Acquire* (b_1). *Register* (m_2)
C	*Acquire* (b_2). *Acquire* (b_1). *Register* (m_2). *Register* (m_1)
D	*Acquire* (b_2). *Acquire* (b_1). *Register* (m_2). *Register* (m_1). *Lend* (b_1, m_1)

Figure 2.7: Trace for the sample execution

$\mathsf{f_1}$ with borrower and $\mathsf{f_2}$ with nbLoans in Definition 2.3.9, EB^3 memory $\mathsf{M_0}$ is (initially) set to:

$$\mathsf{M_0} = [\, \mathsf{borrower[b_1]} \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower[b_2]} \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans[m_1]} \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans[m_2]} \leftarrow \perp^{\mathcal{F}}\,].$$

By setting $i = 1$, we may carry on with *attribute variable* borrower. Comparing the standard syntax of *attribute function* definitions (see Definition 2.2.3 for details) with the syntax of borrower in Figure 2.2, w_i^0 of Definition 2.3.9 can be matched with $\perp$. Then, applying Definition 2.3.9 and function $[\![\cdot]\!]_1^{\mathsf{M}}$ of Definition 2.3.8, the primed *attribute variable* $\mathsf{borrower'\ [b_1]}$ referring to $\mathsf{borrower\ [b_1]}$ at state (A) can be calculated as follows:

$$\mathsf{borrower'[b_1]} = [\![\perp]\!]_1^{\mathsf{M}}(\tau',\ \mathsf{M_0}) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ]$" modifying EB^3 memory $\mathsf{M_0}$ as follows:

$$\mathsf{M_0} = \mathsf{M_0} \oplus [\ \mathsf{borrower'[b_1]} \leftarrow \perp^{\mathcal{F}}\ ]$$

The computation continues with *attribute variable* borrower $[b_2]$ at state (A):

$$\mathsf{borrower'[b_2]} = [\![\perp]\!]_1^{\mathsf{M}}(\tau',\ \mathsf{M_0}) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b_2}\ ]$" modifying EB^3 memory $\mathsf{M_0}$ as follows:

$$\mathsf{M_0} = \mathsf{M_0} \oplus [\ \mathsf{borrower'[b_2]} \leftarrow \perp^{\mathcal{F}}\ ].$$

Hence, the present value of EB^3 memory $\mathsf{M_0}$ is the following:

$$\begin{aligned}\mathsf{M_0} = [\ &\mathsf{borrower[b_1]} \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower[b_2]} \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans[m_1]} \leftarrow \perp^{\mathcal{F}},\\ &\mathsf{nbLoans[m_2]} \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower'[b_1]} \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower'[b_2]} \leftarrow \perp^{\mathcal{F}}\ ].\end{aligned}$$

By setting $i = 2$, we pass to *attribute variable* nbLoans. The primed *attribute variable* $\mathsf{nbLoans'[m_1]}$ at state (A) is the following:

$$\mathsf{nbLoans'[m_1]} = [\![\perp]\!]_1^{\mathsf{M}}\ (\tau',\ \mathsf{M_0}) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m_1}\ ]$" modifying EB^3 memory $\mathsf{M_0}$ as follows:

$$\mathsf{M_0} = \mathsf{M_0} \oplus [\ \mathsf{nbLoans'[m_1]} \leftarrow \perp^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{nbLoans'[m_2]}$ at state (A):

$$\mathsf{nbLoans[m_2]} = [\![\bot]\!]^{\mathsf{M}}_1\ (\tau',\ \mathsf{M_0}) = \bot^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m_2}\ ]$" modifying EB^3 memory $\mathsf{M_0}$ as follows:

$$\mathsf{M_0} = \mathsf{M_0} \oplus [\ \mathsf{nbLoans[m_2]} \leftarrow \bot^{\mathcal{F}}\ ].$$

Combining the previous results, the present value of $\mathsf{M_0}$ is the following:

$$\begin{aligned}\mathsf{M_0} = [\ &\mathsf{borrower[b_1]} \leftarrow \bot^{\mathcal{F}},\ \mathsf{borrower[b_2]} \leftarrow \bot^{\mathcal{F}},\ \mathsf{nbLoans[m_1]} \leftarrow \bot^{\mathcal{F}},\ \mathsf{nbLoans[m_2]} \leftarrow \bot^{\mathcal{F}},\\ &\mathsf{borrower'[b_1]} \leftarrow \bot^{\mathcal{F}},\ \mathsf{borrower'[b_2]} \leftarrow \bot^{\mathcal{F}},\ \mathsf{nbLoans'[m_1]} \leftarrow \bot^{\mathcal{F}},\ \mathsf{nbLoans'[m_2]} \leftarrow \bot^{\mathcal{F}}\ ],\end{aligned}$$

then replacing each *attribute variable* in $\mathsf{M_0}$ by the corresponding primed variable, and removing all primed *attribute variables* from $\mathsf{M_0}$, $\mathsf{M_0}$ is transformed as follows:

$$\mathsf{M_A} = [\ \mathsf{borrower}\ [\mathsf{b_1}] \leftarrow \bot^{\mathcal{F}},\ \mathsf{borrower}\ [\mathsf{b_2}] \leftarrow \bot^{\mathcal{F}},\ \mathsf{nbLoans}\ [\mathsf{m_1}] \leftarrow \bot^{\mathcal{F}},\ \mathsf{nbLoans}\ [\mathsf{m_2}] \leftarrow \bot^{\mathcal{F}}$$

denoting the EB^3 memory at state (A).

- Transition (A)→(A') describes the execution of action "*Acquire* (b_2)". Before proceeding further with the calculations, we need to resolve the duplicate name issue regarding *formal parameter* bId of action prototype definition:

$$D_1 ::= \mathit{Acquire}\ (\mathit{bId} : \mathit{BID})$$

and *formal parameter* bId of *attribute function* definition:

$$A_1 ::= \mathit{borrower}\ (\mathsf{T} : \mathcal{T}, \mathit{bId} : \mathit{BID}) : \mathit{MID}_{\bot}$$

(see Figure 2.2 for details). To this end, D_1 is modified as follows:

$$D_1 ::= \mathit{Acquire}\ (\mathit{bId}' : \mathit{BID}).$$

In the following, whenever similar issues arise, we turn *formal parameter* identifiers of action prototype definitions into their corresponding primed versions.

The EB^3 memory at state (A') can be calculated by way of function "$\mathit{upd}(\mathit{Acquire}\ (b_2), \mathsf{M_A})$" of Definition 2.3.10. Following Definition 2.3.10, $\mathsf{M'}$ corresponding to the EB^3 memory at state (A') is set to $\mathsf{M_A}$. Then, environment τ is found equal to "$\tau \doteq [\ \mathsf{bId'} \leftarrow \mathsf{b_2}\ ]$".

By setting $i = 1$, we may carry on with *attribute variable* borrower. Comparing the standard syntax of *attribute function* definitions (see Definition 2.2.3 for details) with the syntax of borrower in Figure 2.2, w_i^j of Definition 2.3.9 can match with "$\mathit{borrower}\ (\mathit{front}\ (\mathsf{T}), \mathit{bId})$". Then, applying Definition 2.3.10 and function $[\![\cdot]\!]^{\mathsf{M}}_1$ of Definition 2.3.8, the primed *attribute variable* $\mathsf{borrower'}[b_1]$ referring to $\mathsf{borrower}[b_1]$ at state (A') can be calculated as follows:

$$\mathsf{borrower'[b_1]} = [\![\mathit{borrower}\ (\mathit{front}\ (\mathsf{T}), \mathit{bId})]\!]^{\mathsf{M}}_1(\tau \cup \tau',\ \mathsf{M'}) = \mathsf{M'}(\mathsf{borrower})(\mathsf{b_1}) = \bot^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ]$" modifying EB^3 memory $\mathsf{M'}$ as follows:

$$\mathsf{M'} = \mathsf{M'} \oplus [\ \mathsf{borrower'[b_1]} \leftarrow \bot^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{borrower}[\mathsf{b}_2]$ at state (A'):

$$\mathsf{borrower}'[\mathsf{b}_2] = [\![borrower\,(front\ (\mathsf{T}), bId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})(\mathsf{b}_2) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus \big[\ \mathsf{borrower}'\ [\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}}\ \big].$$

Hence, the present value of EB^3 memory M' is the following:

$$\begin{aligned}\mathsf{M}' = [\ &\mathsf{borrower}[\mathsf{b}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower}[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}[\mathsf{m}_1] \leftarrow \perp^{\mathcal{F}},\\ &\mathsf{nbLoans}[\mathsf{m}_2] \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower}'[\mathsf{b}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower}'[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}}\ ].\end{aligned}$$

By setting $i = 2$, we pass to *attribute variable* $\mathsf{nbLoans}$. Hence, the primed *attribute variable* $\mathsf{nbLoans}'[\mathsf{m}_1]$ at state (A') is the following:

$$\mathsf{nbLoans}'[\mathsf{m}_1] = [\![nbLoans\,(front\,(\mathsf{T}), mId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{nbLoans})(\mathsf{m}_1) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m}_1\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'[\mathsf{m}_1] \leftarrow \perp^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{nbLoans}'[m_2]$ at state (A'):

$$\mathsf{nbLoans}'[\mathsf{m}_2] = [\![nbLoans\,(front\,(\mathsf{T}), mId)]\!]_1^{\mathsf{M}}\ (\tau \cup \tau',\ \mathsf{M}_0) = \mathsf{M}_0(\mathsf{nbLoans})(\mathsf{m}_2) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'[\mathsf{m}_2] \leftarrow \perp^{\mathcal{F}}\ ].$$

Combining the previous results, the present value of M' is the following:

$$\begin{aligned}\mathsf{M}' = [\ &\mathsf{borrower}[\mathsf{b}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower}[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}[\mathsf{m}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}[\mathsf{m}_2] \leftarrow \perp^{\mathcal{F}},\\ &\mathsf{borrower}'[\mathsf{b}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower}'[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}'[\mathsf{m}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}'[\mathsf{m}_2] \leftarrow \perp^{\mathcal{F}}\ ],\end{aligned}$$

then replacing each *attribute variable* in M' by the corresponding primed variable, and removing all primed *attribute variables* from M', M' is transformed as follows:

$$\mathsf{M}_{\mathsf{A}'} = [\ \mathsf{borrower}[\mathsf{b}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower}[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}\,[\mathsf{m}_1] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}\,[\mathsf{m}_2] \leftarrow \perp^{\mathcal{F}}\ ]$$

denoting the EB^3 memory at state (A').

- Transition (A')→(B) describes the execution of action "*Acquire* (b_1)". Applying similar reasoning, we show directly how *attribute variables* are modified.

  The EB^3 memory at state (B) can be calculated by way of function "$upd(Acquire\,(b_1), \mathsf{M}'_{\mathsf{A}})$" of Definition 2.3.10. Following Definition 2.3.10, M' corresponding to the EB^3 memory at state (B) is set to $\mathsf{M}_{\mathsf{A}'}$. Then, environment τ is found equal to "$\tau \doteq [\ \mathsf{bId}' \leftarrow \mathsf{b}_1\ ]$".

  The primed *attribute variable* $\mathsf{borrower}'[\mathsf{b}_1]$ referring to $\mathsf{borrower}\ [b_1]$ at state (B) can be calculated as follows:

$$\mathsf{borrower}'[\mathsf{b}_1] = [\![borrower\,(front\ (\mathsf{T}), bId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})(\mathsf{b}_1) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b}_1\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b}_1] \leftarrow \perp^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{borrower}[b_2]$ at state (B):

$$\mathsf{borrower}'[\mathsf{b}_2] = [\![borrower\,(front\ (\mathsf{T}), bId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})(\mathsf{b}_2) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}}\ ].$$

The primed *attribute variable* $\mathsf{nbLoans}'[\mathsf{m}_1]$ at state (B) is the following:

$$\mathsf{nbLoans}'\ [\mathsf{m}_1] = [\![nbLoans\,(front\,(\mathsf{T}), mId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{nbLoans})(\mathsf{m}_1) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m}_1\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'[\mathsf{m}_1] \leftarrow \perp^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{nbLoans}'[\mathsf{m}_2]$ at state (B):

$$\mathsf{nbLoans}'[\mathsf{m}_2] = [\![nbLoans\,(front\,(\mathsf{T}), mId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{nbLoans})(\mathsf{m}_2) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ mId \leftarrow \mathsf{m}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'[\mathsf{m}_2] \leftarrow \perp^{\mathcal{F}}\ ].$$

Combining the previous results, replacing each *attribute variable* in M' by the corresponding primed variable, and removing all primed *attribute variables* from M', EB^3 memory M' is finally transformed as follows:

$$\mathsf{M}_\mathsf{B} = [\ \mathsf{borrower}[\mathsf{b}_1] \leftarrow \perp,\ \mathsf{borrower}[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}[\mathsf{m}_1] \leftarrow \perp,\ \mathsf{nbLoans}[\mathsf{m}_2] \leftarrow \perp^{\mathcal{F}}\ ]$$

denoting the EB^3 memory at state (B).

- Transition (B)→(B') describes the execution of action "*Register* (m_2)". Hence, the EB^3 memory at state (B') can be calculated by way of function "$upd\big(Register\,(m_2), \mathsf{M}_\mathsf{B}\big)$" of Definition 2.3.10. Following Definition 2.3.10, M' corresponding to the EB^3 memory at state (B') is set to M_B.

  Then, environment τ is found equal to "$\tau \doteq [\ \mathsf{mId}' \leftarrow \mathsf{m}_2\ ]$", as the duplicate renaming issue regarding formal parameter mId of action label *Register* and *attribute parameter* mId of *nbLoans* has been resolved by changing mId into mId'.

  The primed *attribute variable* $\mathsf{borrower}'[\mathsf{b}_1]$ referring to $\mathsf{borrower}[b_1]$ at state (B) can be calculated as follows:

$$\mathsf{borrower}'[\mathsf{b}_1] = [\![borrower\,(front\ (\mathsf{T}), bId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})\ (\mathsf{b}_1) = \perp^{\mathcal{F}},$$

  where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b}_1\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b}_1] \leftarrow \perp^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{borrower}[b_2]$ at state (B'):

$$\mathsf{borrower}'[\mathsf{b_2}] = [\![\mathit{borrower}\,(\mathit{front}\ (\mathsf{T}), \mathit{bId})]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})(\mathsf{b_2}) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b_2}\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b_2}] \leftarrow \perp^{\mathcal{F}}\ ].$$

The primed *attribute variable* $\mathsf{nbLoans}'[\mathsf{m_1}]$ at state (B) is the following:

$$\mathsf{nbLoans}'[\mathsf{m_1}] = [\![\mathit{nbLoans}\,(\mathit{front}\,(\mathsf{T}), \mathit{mId})]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{nbLoans})(\mathsf{m_1}) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m_1}\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'(\mathsf{m_1}) \leftarrow \perp^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{nbLoans}'[m_2]$ at state (B'):

$$\mathsf{nbLoans}'[\mathsf{m_2}] = [\![0]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = 0,$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m_2}\ ]$" and, owing to case expression "$|\ \mathit{Register}\ (\mathit{mId})\ : 0$" (see Figure 2.2), "*Register* (mId)" matches with action "*Register* (m_2)", hence, modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus \big[\ \mathsf{nbLoans}'\ [\mathsf{m_2}] \leftarrow 0\ \big].$$

Combining the previous results, replacing each *attribute variable* in M' by the corresponding primed variable, and removing all primed *attribute variables* from M', EB^3 memory M' is transformed as follows:

$$\mathsf{M_{B'}} = [\ \mathsf{borrower}[\mathsf{b_1}] \leftarrow \perp^{\mathcal{F}},\ \mathsf{borrower}\ [\mathsf{b_2}] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}\ [\mathsf{m_1}] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}\ [\mathsf{m_2}] \leftarrow 0\ ]$$

denoting the EB^3 memory at state (B').

- Transition (B')→(C) describes the execution of action "*Register* (m_1)". As previously, the EB^3 memory at state (C) can be calculated by way of function "$\mathit{upd}\big(\mathit{Register}\,(m_1), \mathsf{M_{B'}}\big)$" of Definition 2.3.10. Following Definition 2.3.10, M' referring to the EB^3 memory at state (C) is set to $\mathsf{M_{B'}}$.

  Then, environment τ is found equal to "$\tau \doteq [\ \mathsf{mId}' \leftarrow \mathsf{m_1}\ ]$", as the duplicate renaming issue regarding formal parameter mId of action label *Register* and *attribute parameter* mId of *nbLoans* has been resolved by turning mId into mId'.

  The primed *attribute variable* $\mathsf{borrower}'[\mathsf{b_1}]$ referring to $\mathsf{borrower}[\mathsf{b_1}]$ at state (C) can be calculated as follows:

$$\mathsf{borrower}'[\mathsf{b_1}] = [\![\mathit{borrower}\,(\mathit{front}\ (\mathsf{T}), \mathit{bId})]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})(\mathsf{b_1}) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b_1}\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b_1}] \leftarrow \perp^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{borrower}[\mathsf{b}_2]$ at state (C):

$$\mathsf{borrower}'[\mathsf{b}_2] = [\![\mathit{borrower}\,(\mathit{front}\ (\mathsf{T}), \mathit{bId})]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})(\mathsf{b}_2) = \bot,$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b}_2] \leftarrow \bot^{\mathcal{F}}\ ].$$

The primed *attribute variable* $\mathsf{nbLoans}'[\mathsf{m}_1]$ at state (C) is the following:

$$\mathsf{nbLoans}'[\mathsf{m}_1] = [\![0]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = 0,$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m}_1\ ]$" and, owing to case expression "$|\ \mathit{Register}\ (\mathit{mId}) : 0$" (see Figure 2.2), "$\mathit{Register}\ (\mathit{mId})$" matches with action "$\mathit{Register}\,(m_1)$", hence, modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'[\mathsf{m}_1] \leftarrow \bot^{\mathcal{F}}\ ].$$

The computation continues with *attribute variable* $\mathsf{nbLoans}'[\mathsf{m}_2]$ at state (C):

$$\mathsf{nbLoans}'[\mathsf{m}_2] = [\![\mathit{nbLoans}\,(\mathit{front}\,(\mathsf{T}), \mathit{mId})]\!]_1^{\mathsf{M}}\ (\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{nbLoans})(\mathsf{m}_2) = 0,$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus \big[\ \mathsf{nbLoans}'[\mathsf{m}_2] \leftarrow 0\ \big].$$

Combining the previous results, replacing each *attribute variable* in M' by the corresponding primed variable, and removing all primed *attribute variables* from M', EB^3 memory M' is transformed as follows:

$$\mathsf{M_C} = [\ \mathsf{borrower}[\mathsf{b}_1] \leftarrow \bot^{\mathcal{F}},\ \mathsf{borrower}[\mathsf{b}_2] \leftarrow \bot^{\mathcal{F}},\ \mathsf{nbLoans}\ [\mathsf{m}_1] \leftarrow 0,\ \mathsf{nbLoans}\ [\mathsf{m}_2] \leftarrow 0\ ]$$

denoting the EB^3 memory at state (C).

- Transition (C)→(D) describes the execution of action "$\mathit{Lend}\,(b_1, m_1)$". As previously, the EB^3 memory at state (C) can be calculated by way of function "$\mathit{upd}\big(\mathit{Lend}\,(b_1, m_1), \mathsf{M_C}\big)$" of Definition 2.3.10. Following Definition 2.3.10, M' relating to the EB^3 memory at state (D) is set to $\mathsf{M_C}$.

  Then, environment τ is found equal to "$\tau \doteq [\ \mathsf{bId}' \leftarrow \mathsf{b}_1,\ \mathsf{mId}' \leftarrow \mathsf{m}_1\ ]$", where the duplicate renaming issues regarding bId and mId have been dealt with as previously.

  The primed *attribute variable* $\mathsf{borrower}'[\mathsf{b}_1]$ referring to $\mathsf{borrower}[\mathsf{b}_1]$ at state (D) can be calculated as follows:

$$\mathsf{borrower}'[\mathsf{b}_1] = [\![\mathit{mId}]\!]_1^{\mathsf{M}}\ (\tau \cup \tau',\ \mathsf{M}') = \mathsf{m}_1,$$

  where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b}_1\ ]$" and, owing to case expression "$|\ \mathit{Lend}\ (\mathit{bId}, \mathit{mId}) : \mathit{mId}$" (see Figure 2.2), "$\mathit{Lend}\ (\mathit{bId}, \mathit{mId})$" is matched with action "$\mathit{Lend}\,(b_1, m_1)$", hence, modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b}_1] \leftarrow \mathsf{m}_1\ ].$$

The computation continues with *attribute variable* $\mathsf{borrower}[\mathsf{b}_2]$ at state (D):

$$\mathsf{borrower}'[\mathsf{b}_2] = [\![borrower\,(front\ (\mathsf{T}), bId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{borrower})(\mathsf{b}_2) = \perp^{\mathcal{F}},$$

where "$\tau' \doteq [\ \mathsf{bId} \leftarrow \mathsf{b}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{borrower}'[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}}\ ].$$

The primed *attribute variable* $\mathsf{nbLoans}'[\mathsf{m}_1]$ at state (D) is the following:

$$\mathsf{nbLoans}'[\mathsf{m}_1] = [\![nbLoans\,(front\,(\mathsf{T}), mId) + 1]\!]_1^{\mathsf{M}}\ (\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{nbLoans})(\mathsf{m}_1)+1 = 1,$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m}_1\ ]$" and, owing to case expression:

$$\mid Lend\ (bId, mId)\ : nbLoans\,(front\,(\mathsf{T}), mId) + 1$$

(see Figure 2.2), "$Lend\ (bId, mId)$" is matched with action "$Lend\,(b_1, m_1)$", thus, modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'[\mathsf{m}_1] \leftarrow 1\ ].$$

The computation continues with *attribute variable* $\mathsf{nbLoans}'\ [\mathsf{m}_2]$ at state (D):

$$\mathsf{nbLoans}'[\mathsf{m}_2] = [\![nbLoans\,(front\,(\mathsf{T}), mId)]\!]_1^{\mathsf{M}}(\tau \cup \tau',\ \mathsf{M}') = \mathsf{M}'(\mathsf{nbLoans})(\mathsf{m}_2) = 0,$$

where "$\tau' \doteq [\ \mathsf{mId} \leftarrow \mathsf{m}_2\ ]$" modifying EB^3 memory M' as follows:

$$\mathsf{M}' = \mathsf{M}' \oplus [\ \mathsf{nbLoans}'[\mathsf{m}_2] \leftarrow 0\ ].$$

Combining the previous results, replacing each *attribute variable* in M' by the corresponding primed variable and removing all primed *attribute variables* from M', EB^3 memory M' is transformed as follows:

$$\mathsf{M}_\mathsf{D} = [\ \mathsf{borrower}[\mathsf{b}_1] \leftarrow \mathsf{m}_1,\ \mathsf{borrower}[\mathsf{b}_2] \leftarrow \perp^{\mathcal{F}},\ \mathsf{nbLoans}[\mathsf{m}_1] \leftarrow 1,\ \mathsf{nbLoans}[\mathsf{m}_2] \leftarrow 0\ ]$$

denoting the EB^3 memory at state (D).

The principal difference between Sem_T and $Sem_{\mathsf{T/M}}$ lies in the way guards are evaluated. Revisiting transition (C)→(D), the evaluation of guard (2.30) with respect to $Sem_{\mathsf{T/M}}$ depends solely on the EB^3 memory at state (C), i.e. M_C. In particular, by Definition 2.3.12, guard (2.30) is evaluated as follows:

$$\begin{gathered}
[\![(borrower\,(\mathsf{T}, b_1) = \perp)\ \wedge\ (nbLoans\,(\mathsf{T}, m_1) < 2)]\!]_3^{\mathsf{T}}\ (\mathsf{M}_C) = \\
[\![(borrower\,(\mathsf{T}, b_1) = \perp)]\!]_3^{\mathsf{T}}\ (\mathsf{M}_C)\ \wedge\ [\![(nbLoans\,(\mathsf{T}, m_1) < 2)]\!]_3^{\mathsf{T}}\ (\mathsf{M}_C) = \\
\mathsf{M}_C\ (borrower)\ ([\![b_1]\!]_2^{\mathsf{M}}) =^{\mathcal{F}} \perp\ \wedge\ \mathsf{M}_C\ (nbLoans)\ ([\![m_1]\!]_2^{\mathsf{M}}) <^{\mathcal{F}} 2) = \\
(\perp =^{\mathcal{F}} \perp)\ \wedge\ (0 <^{\mathcal{F}} 2^{\mathcal{F}}) = \mathsf{true},
\end{gathered}$$

where $[\![b_1]\!]_2^{\mathsf{M}}$ evaluates to b_1 and $[\![m_1]\!]_2^{\mathsf{M}}$ evaluates to m_1 according to Definition 2.3.11.

The evolution of *attribute variables* is depicted in Figure 2.8:

Memory				
State	$\mathsf{borrower[b_1]}$	$\mathsf{borrower[b_2]}$	$\mathsf{nbLoans[m_1]}$	$\mathsf{nbLoans[m_2]}$
A	$\perp$	$\perp$	$\perp$	$\perp$
A'	$\perp$	$\perp$	$\perp$	$\perp$
B	$\perp$	$\perp$	$\perp$	$\perp$
B'	$\perp$	$\perp$	$\perp$	0
C	$\perp$	$\perp$	0	0
D	m_1	$\perp$	1	0

Figure 2.8: Evolution of *Atrribute variables*

2.3.3 Memory Semantics Sem_{M}

Sem_{M} is given in Figure 2.9 as a set of rules named (S_1) to (S_{11}). Sem_{M} derives from $Sem_{\mathsf{T/M}}$ by simple elimination of the *trace* variable T from each tuple $(E, \mathsf{T}, \mathsf{M})$ in rules (S_1) upto (S_{11}). It gives a finite state system. Intuitively, this means that the information on the history of executions is kept in the EB^3 memory M, thus rendering the presence of *trace* variable redundant. In particular,

- The system state with respect to Sem_{T} is represented by tuple (E, M), where E is the process expression describing the remaining system behaviour and M is the current EB^3 memory.
- The initial system state is represented by process expression *main* and the initial EB^3 memory $\mathsf{M_0}$, i.e. the tuple $(main, \mathsf{M_0} \doteq upd_0)$.
- The evolution of system state modelled by a labelled transition system (LTS) must adhere to the reduction rules of Figure 2.9.

2.4 Bisimulation Equivalence of Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M}

We recall that the three semantics Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M} of EB^3 are based on the notion of labelled transition systems (LTSs). Besides, the equivalence of Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M} relies on the notion of (action-based) equivalence between systems and, in particular, LTSs.

(Action-based) bisimulation [Par81, Mil80] 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. In this section, we present the bisimulation equivalence proof for Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M}.

2.4.1 Useful definitions

The three semantics Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M} of EB^3 presented in Section 2.3 are structured operational semantics based on labelled transition system (LTS) models as interpretation models for EB^3 process algebra. LTSs are particularly suitable for action-based description formalisms such as EB^3.

$$\rho \in \{\alpha_j(c_1,\ldots,c_p) \mid j \in 1..q\} \cup \lambda, \text{ where } c_1,\ldots,c_p \in \mathcal{C}$$

$$(\mathrm{S}_1)\ \frac{}{(\rho,\mathsf{M}) \xrightarrow{\rho\neq\lambda}_{\mathsf{M}} \left(\surd,\ upd(\rho,\mathsf{M})\right)} \qquad (\mathrm{S}'_1)\ \frac{}{(\rho,\mathsf{M}) \xrightarrow{\lambda}_{\mathsf{M}} (\surd,\mathsf{M})}$$

$$(\mathrm{S}_2)\ \frac{(E_1,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1,\mathsf{M}')}{(E_1.E_2,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1.E_2,\mathsf{M}')} \qquad (\mathrm{S}_3)\ \frac{(E_2,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_2,\mathsf{M}')}{(\surd.E_2,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_2,\mathsf{M}')}$$

$$(\mathrm{S}_4)\ \frac{(E_1,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1,\mathsf{M}')}{(E_1 \mid E_2,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1,\mathsf{M}')}$$

$$(\mathrm{S}_5)\ \frac{}{({E_0}^*,\mathsf{M}) \xrightarrow{\lambda}_{\mathsf{M}} (\surd,\mathsf{M})} \qquad (\mathrm{S}_6)\ \frac{(E_0,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_0,\mathsf{M}')}{({E_0}^*,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_0.{E_0}^*,\mathsf{M}')}$$

$$(\mathrm{S}_7)\ \frac{(E_1,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1,\mathsf{M}') \qquad (E_2,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_2,\mathsf{M}')}{(E_1 \mid\![\Delta]\!\mid E_2,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1 \mid\![\Delta]\!\mid E'_2,\mathsf{M}')}\ in\,(\rho,\Delta)$$

$$(\mathrm{S}_8)\ \frac{(E_1,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1,\mathsf{M}')}{(E_1 \mid\![\Delta]\!\mid E_2,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_1 \mid\![\Delta]\!\mid E_2,\mathsf{M}')}\ \neg in\,(\rho,\Delta)$$

$$(\mathrm{S}_9)\ \frac{}{(\surd \mid\![\Delta]\!\mid \surd,\mathsf{M}) \xrightarrow{\lambda}_{\mathsf{M}} (\surd,\mathsf{M})}$$

$$(\mathrm{S}_{10})\ \frac{(E_0,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_0,\mathsf{M}')}{(ge \Rightarrow E_0,\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E'_0,\mathsf{M}')}\ [\![ge]\!]^{\mathsf{M}}_3(\mathsf{M}) = \mathsf{true}$$

$$(\mathrm{S}_{11})\ \frac{(E[\overline{x} := \overline{u}],\mathsf{M}) \xrightarrow{\rho}_{\mathsf{M}} (E',\mathsf{M}')}{(P\,(\overline{u}),\mathsf{M}') \xrightarrow{\rho}_{\mathsf{M}} (E',\mathsf{M}')}\ P\,(\overline{x}) = E$$

Figure 2.9: EB^3 Memory Semantics (Sem_{M})

Definition 2.4.1. *A labelled transition system (LTS) is a triple*

$$(S,\ \delta \doteq \{\xrightarrow{a}\}_{a\in Act},\ s^0),$$

where:

1. *S is a set of states,*
2. *Act is a set of actions,*
3. *$\xrightarrow{a} \subseteq S \times S$, for all $a \in Act$,*
4. *$s^0 \in S$ is the initial state.*

The proof of equivalence for Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M} relies on the standard notion of bisimulation.

Definition 2.4.2. ***Bisimulation.***
Let $TS_i = (S_i, \rightarrow_i, s_i^0)$, $i = 1, 2$ be two TSs and let $R \subseteq S_1 \times S_2$ be a relation. We say that R is a bisimulation relation for TS_1 and TS_2 if and only if:

- $(s_1^0, s_2^0) \in R$,
- *for all $(s_1, s_2) \in R$, then:*
  - *if $s_1 \xrightarrow{a}_1 s_1'$, then there exists $s_2' \in S_2$ such that $s_2 \xrightarrow{a}_2 s_2'$ and $(s_1', s_2') \in R$,*
  - *if $s_2 \xrightarrow{a}_2 s_2'$, then there exists $s_1' \in S_1$ such that $s_1 \xrightarrow{a}_1 s_1'$ and $(s_1', s_2') \in R$.*

We say that TS_1 and TS_2 are equivalent with respect to bisimulation denoted as $TS_1 \sim TS_2$ if and only if there is a bisimulation relation R for TS_1 and TS_2.

2.4.2 LTS Construction

Let *well-formed* EB^3 specification EB^3 with $main = E$, where E stands for *well-formed* EB^3 process expression. We will construct three LTSs with respect to Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M} respectively. These correspond to the LTSs generated inductively by the rules depicted in Figure 2.3, Figure 2.6 and Figure 2.9. The whole process mimics the construction of a transition system associated with a *transition system specification*, as in [MR05].

We recall that the EB^3 memory at system start is denoted by $\mathsf{M}_0 \in \mathcal{M}$, where $\mathcal{M}$ stands for the set of EB^3 memorys in the IS defined upon the fixed body of *attribute function* definitions in EB^3. The construction is carried out by structural induction over EB^3 process expression E. We assume that each sub-expression of E is part of an EB^3 specification endowed with the fixed body of *attribute function* definitions in EB^3.

We demonstrate how to construct a) the LTS related to process expression E with respect to Sem_{M} denoted as TS_E^{M}, b) the LTS related to process expression E with respect to Sem_{T} denoted as TS_E^{T}, and c) the LTS related to process expression E with respect to $Sem_{\mathsf{T/M}}$ denoted as $TS_E^{\mathsf{T/M}}$.

- Let $E = \rho \neq \lambda$.

  1. TS_ρ^{M} is constructed as follows:

$$\begin{array}{rcl} TS_\rho^{\mathsf{M}} & = & (S_\rho^{\mathsf{M}},\ \delta_\rho,\ s_\rho^0), \\ \text{where} \quad S_\rho & = & \{(\rho, \mathsf{M_0})\} \bigcup \{(\surd,\ upd(\rho, \mathsf{M_0}))\} \\ \delta_\rho & = & \{(\rho, \mathsf{M_0}) \xrightarrow{\rho}_{\mathsf{M}} (\surd,\ upd(\rho, \mathsf{M_0}))\} \\ s_\rho^0 & = & (\rho, \mathsf{M_0}) \end{array}$$

     Note that statespace S_ρ consists of the initial state $(\rho, \mathsf{M_0})$ and the terminal state $(\surd,\ upd(\rho, \mathsf{M_0}))$. We recall that $\mathsf{M_0} \doteq upd_0$ (see Definition 2.3.9 for details on upd_0) and that $upd(\rho, \mathsf{M_0})$ stands for the updated EB^3 memory upon execution of action ρ (see Definition 2.3.10 for details on upd).

  2. TS_ρ^{T} is constructed as follows:

$$\begin{array}{rcl} TS_\rho^{\mathsf{T}} & = & (S_\rho,\ \delta_\rho,\ s_\rho^0), \\ \text{where} \quad S_\rho & = & \{(\rho, [\,])\} \bigcup \{(\surd, [\rho])\} \\ \delta_\rho & = & \{(\rho, [\,]) \xrightarrow{\rho}_{\mathsf{T}} (\surd, [\rho])\} \\ s_\rho^0 & = & (\rho, [\,]), \end{array}$$

We recall that [ ] stands for the empty trace and $[\rho]$ is the trace that contains action ρ.

3. Combining the previous results, $TS_{\rho}^{\mathsf{T/M}}$ is constructed as follows:

$$\begin{array}{rrcl} & TS_{\rho}^{\mathsf{T/M}} & = & (S_{\rho},\ \delta_{\rho},\ s_{\rho}^{0}), \\ \text{where} & S_{\rho} & = & \big\{(\rho,[\,],\mathsf{M_0})\big\} \bigcup \big\{(\surd,[\rho],upd\big(\rho,\mathsf{M_0})\big)\big\} \\ & \delta_{\rho} & = & \big\{(\rho,[\,],\mathsf{M_0}) \xrightarrow{\rho}_{\mathsf{T}} \big(\surd,[\rho],upd(\rho,\mathsf{M_0})\big)\big\} \\ & s_{\rho}^{0} & = & (\rho,[\,],\mathsf{M_0}), \end{array}$$

- Let $E = \lambda$.

1. $TS_{\lambda}^{\mathsf{M}}$ is constructed as follows:

$$\begin{array}{rrcl} & TS_{\lambda} & = & (S_{\lambda},\ \delta_{\lambda},\ s_{\lambda}^{0}), \\ \text{where} & S_{\lambda} & = & \big\{(\lambda,\mathsf{M_0})\big\} \\ & \delta_{\rho} & = & \big\{(\lambda,\mathsf{M_0}) \xrightarrow{\lambda}_{\mathsf{M}} (\surd,\mathsf{M_0})\big\} \\ & s_{\lambda}^{0} & = & (\lambda,\mathsf{M_0}) \end{array}$$

Note that statespace S_{ρ} consists of the initial state $(\rho,\mathsf{M_0})$ and the terminal state $(\surd,\mathsf{M_0})$ and that the inert action λ has no effect on EB^3 memory $\mathsf{M_0}$.

2. $TS_{\lambda}^{\mathsf{T}}$ is constructed as follows:

$$\begin{array}{rrcl} & TS_{\lambda}^{\mathsf{T}} & = & (S_{\lambda},\ \delta_{\lambda},\ s_{\lambda}^{0}), \\ \text{where} & S_{\lambda} & = & \big\{(\lambda,[\,])\big\} \\ & \delta_{\rho} & = & \big\{(\lambda,[\,]) \xrightarrow{\lambda}_{\mathsf{M}} (\surd,[\,])\big\} \\ & s_{\lambda}^{0} & = & (\lambda,[\,]) \end{array}$$

Note that the inert action λ is not inserted to the trace.

3. Combining the previous results, $TS_{\lambda}^{\mathsf{T/M}}$ is constructed as follows:

$$\begin{array}{rrcl} & TS_{\lambda}^{\mathsf{T/M}} & = & (S_{\lambda},\ \delta_{\lambda},\ s_{\lambda}^{0}), \\ \text{where} & S_{\lambda} & = & \big\{(\lambda,[\,],\mathsf{M_0})\big\} \\ & \delta_{\rho} & = & \big\{(\lambda,[\,],\mathsf{M_0}) \xrightarrow{\lambda}_{\mathsf{M}} (\surd,[\,],\mathsf{M_0})\big\} \\ & s_{\lambda}^{0} & = & (\lambda,[\,],\mathsf{M_0}) \end{array}$$

- Let $E = E_1 \,|\, E_2$.

We aim at a generic formula $TS\ (X,Y)$ denoting the LTS that describes $E_1 \,|\, E_2$ with respect to Sem_{M}, Sem_{T} and $Sem_{\mathsf{T/M}}$ for suitable values of variable $X \in \{\mathsf{M_0},[\,],([\,],\mathsf{M_0})\}$ and $Y \in \{\mathsf{M},\mathsf{T},\mathsf{T/M}\}$.

To this end, we apply compositional reasoning relying on $TS_{E_1}^{Y}$ and $TS_{E_2}^{Y}$. In particular, we consider LTS $TS\ (X,Y)$ defined as follows:

$$\begin{array}{rrcl} & TS(X,Y) & = & \big(S(X,Y),\ \delta(X,Y),\ s^{0}(X,Y)\big), \\ \text{where} & S(X,Y) & = & S_{E_1}^{Y} \setminus \big\{(E_1,X)\big\} \bigcup S_{E_2}^{Y} \setminus \big\{(E_2,X)\big\} \bigcup \big\{(E_1 \,|\, E_2,X)\big\} \\ & \delta(X,Y) & = & \delta_{E_1}^{Y}\big[E_1 \leftarrow E_1 \,|\, E_2\big] \bigcup \delta_{E_2}^{Y}\big[E_2 \leftarrow E_1 \,|\, E_2\big] \\ & s^{0}(X,Y) & = & (E_1 \,|\, E_2,X) \end{array}$$

Notice that statespace $S\ (X, Y)$ comprises the union of state space $S^Y_{E_1}$ and state space $S^Y_{E_2}$ from which all references to initial state $(E_1, X) \in S^Y_{E_1}$ and initial state $(E_2, X) \in S^Y_{E_2}$ have been substituted by initial state $(E_1 \,|\, E_2, X) \in S(X, Y)$.

Similarly, transition relation $\delta_{E_1 \,|\, E_2}$ comprises the union of transition relation δ_{E_1} and transition relation δ_{E_2} for which a) all references to process expression E_1 in tuples of $\delta_{E_1 \,|\, E_2}$ have been substituted by $E_1 \,|\, E_2$ denoted as $[E_1 \leftarrow E_1 \,|\, E_2]$, and b) all references to process expression E_2 in elements of $\delta_{E_1 \,|\, E_2}$ have been substituted by $E_1 \,|\, E_2$ denoted as $[E_1 \leftarrow E_1 \,|\, E_2]$. It follows easily that:

1. $TS^{\mathsf{M}}_{E_1 \,|\, E_2} \doteq TS\ (\mathsf{M_0}, \mathsf{M})$ is the LNT describing $E_1 \,|\, E_2$ w.r.t. Sem_{M}.
2. $TS^{\mathsf{T}}_{E_1 \,|\, E_2} \doteq TS\ ([\,], \mathsf{T})$ is the LNT describing $E_1 \,|\, E_2$ w.r.t. Sem_{T}.
3. $TS^{\mathsf{T/M}}_{E_1 \,|\, E_2} \doteq TS\ \big(([\,], \mathsf{M_0}), \mathsf{T/M}\big)$ is the LNT describing $E_1 \,|\, E_2$ w.r.t. $Sem_{\mathsf{T/M}}$.

- Let $E = E_1.E_2$.

  As a means to construct $TS^Y_{E_1.E_2}$ for $Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$, we adopt a compositional approach relying on $TS^Y_{E_1}$ and $TS^Y_{E_2}$.

  First, we remark that if state (E_1', X) belongs to $S^Y_{E_1}$ for an arbitrary derivation E_1' of E_1 and $X \in \{\mathsf{M'}, \mathsf{T'}, (\mathsf{T'}, \mathsf{M'})\}$, where $\mathsf{T'}$ is a trace and $\mathsf{M'}$ is an EB^3 memory, then state $(E_1'.E_2, X)$ shall denote a valid state of $S^Y_{E_1.E_2}$. Then, we recall that as soon as the execution of process expression E_1 is completed, the system may carry on with the execution of process expression E_2. Hence, $S^Y_{E_1.E_2}$ shall contain the union of state spaces $S^Y_{E_2,Z}$ for $Z \in \{\mathsf{M''}, \mathsf{T''}, (\mathsf{T''}, \mathsf{M''})\}$, a $\mathsf{T''}$ is a trace and $\mathsf{M''}$ is an EB^3 memory, such that $(\surd, Z) \in S^Y_{E_1}$ and $S^Y_{E_2,Z}$ refers to the statespace of EB^3 specification $EB^{3\text{'}}$, for which "*main* $= E_2$" and the set of *attribute function* definitions is specified as described below:

  For $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, q\}$, *attribute function definition* f_i of EB^3 (see Definition 2.2.3 for details) is uniquely defined in the following manner:

$$
\begin{aligned}
&f_i\,(\mathsf{T} : \mathcal{T}, \overline{y} : \overline{T}) : T = \textbf{match}\ \mathit{last}\,(\mathsf{T})\ \textbf{with}\\
&\qquad \bot_{\mathsf{T}} : w_i^0\\
&\qquad |\, \alpha_1\,(\overline{z_1}) : w_i^1 \,|\, \ldots \,|\, \alpha_q\,(\overline{z_q}) : w_i^q\\
&\qquad [\,|_{\text{-}} : w_i^{q+1}\,]\\
&\textbf{end match},
\end{aligned}
$$

  then *attribute function definition* f_i' of $EB^{3\text{'}}$ shall be specified as follows:

$$
\begin{aligned}
&f_i'\,(\mathsf{T} : \mathcal{T}, \overline{y} : \overline{T}) : T = \textbf{match}\ \mathit{last}\,(\mathsf{T})\ \textbf{with}\\
&\qquad \bot_{\mathsf{T}} : \textbf{if}\ \overline{y} = \overline{c}_1\ \textbf{then}\ \mathsf{f_i}^{\overline{c}_1}\ \textbf{else}\\
&\qquad\qquad \ldots\\
&\qquad\qquad \textbf{else if}\ \overline{y} = \overline{c}_o\ \textbf{then}\ \mathsf{f_i}^{\overline{c}_o}\\
&\qquad\qquad \textbf{end if}\\
&\qquad |\, \alpha_1\,(\overline{z_1}) : w_i^1 \,|\, \ldots \,|\, \alpha_q\,(\overline{z_q}) : w_i^q\\
&\qquad [\,|_{\text{-}} : w_i^{q+1}\,]\\
&\textbf{end match},
\end{aligned}
$$

where

$$\mathsf{f}_i^{\overline{c}_k} = \begin{cases} \mathsf{M}''(\mathsf{f_i})[\overline{c_k}], & \text{if } Y = \mathsf{M},\ Z = \mathsf{M}'' \quad \text{or} \quad Y = \mathsf{T/M},\ Z = (\mathsf{T}'', \mathsf{M}'')\} \\ f_i\,(\mathsf{T}'', \overline{c_k}), & \text{if } Y = \mathsf{T},\ Z = \mathsf{T}'' \\ \text{undefined}, & \text{otherwise} \end{cases}$$

for $k \in \{1, \ldots, o\}$ and notation "$\overline{y} : \overline{T}$" is an abbreviation for "$y_1 : T_1, \ldots, y_s : T_s$", $o \in \mathbb{N}$ is equal to the cartesian product "$|T_1| \times \ldots \times |T_s|$", and notation "$\overline{y} = \overline{c}_i$" is an abbreviation for syntactic expression "$y_1 = c_{i,1} \wedge \ldots y_s = c_{i,s}$", where $c_{i,1}, \ldots, c_{i,s} \in \mathcal{C}$ are constants and $c^{\mathcal{F}}_{i,k} \in \lfloor T_k \rfloor$ for $i \in \{1, \ldots, o\}$ and $k \in \{1, \ldots, s\}$. Transition relation $\delta^Y_{E_1.E_2}$ is obtained with similar reasoning.

We consider LTS $TS(W, Y)$ for $W \in \{\mathsf{M_0}, [\,], ([\,], \mathsf{M_0})\}$ and $Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$ defined as follows:

$$\begin{array}{rrcl} & TS(W,Y) & = & \big(S(W,Y),\ \delta(W,Y),\ s^0(W,Y)\big), \\ \text{where} & S(W,Y) & = & \big\{(E_1'.E_2, X) \mid (E_1', X) \in S^Y_{E_1}\big\} \bigcup \ \big(\bigcup_{Z \in A} S^Y_{E_2,Z}\big) \\ & A & \doteq & \big\{Z \mid (\surd, Z) \in S^Y_{E_1}\big\} \quad \text{and} \quad X \in \{\mathsf{M}', \mathsf{T}', (\mathsf{T}', \mathsf{M}')\} \\ & \delta(W,Y) & = & \big\{(E_1'.E_2, X) \xrightarrow{\rho}_Y (E_1''.E_2, X') \mid (E_1', X) \xrightarrow{\rho}_Y (E_1'', X') \in \delta^Y_{E_1}\big\} \\ & & & \bigcup \ \big(\bigcup_{Z \in B} \delta^Y_{E_2,Z}\big) \\ & B & \doteq & \big\{Z \mid (\surd, Z) \in S^Y_{E_1}\big\} \quad \text{and} \quad X' \in \{\mathsf{M}'', \mathsf{T}'', (\mathsf{T}'', \mathsf{M}'')\} \\ & s^0(W,Y) & = & (E_1.E_2, W) \end{array}$$

Remark that T'' is a valid trace and M'' is a valid EB^3 memory.

It follows easily that:

1. $TS^{\mathsf{M}}_{E_1.E_2} \doteq TS(\mathsf{M_0}, \mathsf{M})$ is the LNT describing $E_1.E_2$ w.r.t. Sem_{M}.
2. $TS^{\mathsf{T}}_{E_1.E_2} \doteq TS([\,], \mathsf{T})$ is the LNT describing $E_1.E_2$ w.r.t. Sem_{T}.
3. $TS^{\mathsf{T/M}}_{E_1.E_2} \doteq TS\big(([\,], \mathsf{M_0}), \mathsf{T/M}\big)$ is the LNT describing $E_1.E_2$ w.r.t. $Sem_{\mathsf{T/M}}$.

- Let $E = E_1 |[\Delta]| E_2$.

  We aim at a generic formula:

$$TS(X,Y) = \big(S(X,Y), \delta(X,Y), s^0(X,Y)\big)$$

denoting the LTS that describes $E_1|[\Delta]|E_2$ with respect to Sem_{M}, Sem_{T} and $Sem_{\mathsf{T/M}}$ for suitable values of variable $X \in \{\mathsf{M_0}, [\,], ([\,], \mathsf{M_0})\}$ and $Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$.

The construction of $S(X,Y)$ is carried out as depicted below:

$$
\begin{array}{rcl}
S'(X,Y) & = & \{(E_1|[\Delta]|E_2, X)\}; \\
\textbf{do} & \{ & \\
S(X,Y) & = & S'(X,Y); \\
S'(X,Y) & = & S(X,Y) \\
 & \bigcup & \{(E_1''|[\Delta]|E_2', X'') \mid \exists E_1', \exists \rho, \exists X' : \ (E_1'|[\Delta]|E_2', X') \in S(X,Y) \\
 & & \qquad \wedge \ \neg in(\rho, \Delta) \\
 & & \qquad \wedge \ (E_1', X') \xrightarrow{\rho}_Y (E_1', X'') \in \delta_{E_1}\} \\
 & \bigcup & \{(E_1'|[\Delta]|E_2'', X'') \mid \exists E_2', \exists \rho, \exists X' : \ (E_1'|[\Delta]|E_2', X') \in S(X,Y) \\
 & & \qquad \wedge \ \neg in(\rho, \Delta) \\
 & & \qquad \wedge \ (E_2', X') \xrightarrow{\rho}_Y (E_2'', X'') \in \delta_{E_2}\} \\
 & \bigcup & \{(E_1''|[\Delta]|E_2'', X'') \mid \exists E_1', \exists E_2', \exists \rho, \exists X', \ (E_1'|[\Delta]|E_2', X') \in S(X,Y) \\
 & & \qquad \wedge \ in(\rho, \Delta) \\
 & & \qquad \wedge \ (E_1', X') \xrightarrow{\rho}_Y (E_1'', X'') \in \delta_{E_1} \\
 & & \qquad \wedge \ (E_2', X') \xrightarrow{\rho}_Y (E_2'', X'') \in \delta_{E_2}\} \\
 & \bigcup & \{(\surd, X') \mid (\surd|[\Delta]|\surd, X') \in S(X,Y)\}; \\
 & \} & \\
\textbf{while} & & \big(S'(X,Y) \neq S(X,Y)\big)
\end{array}
$$

Auxiliary variable $S'(X,Y)$ is initially set to singleton set $\{(E_1|[\Delta]|E_2, X)\}$.

$S(X,Y)$ is set to $S'(X,Y)$. Statespace $S(X,Y)$ is iteratively updated to $S'(X,Y)$ that constitutes the union of $S(X,Y)$ and

1. the states $(E_1''|[\Delta]|E_2', X'')$, for which there exist EB^3 process expression $E_1' \in S_{E_1}^Y$, variable X', and action ρ such that $(E_1'|[\Delta]|E_2', X') \in S(X,Y)$ and $(E_1', X') \xrightarrow{\rho}_Y (E_1'', X'') \in \delta_{E_1}^Y$ on condition that the label of action ρ does not belong to synchronization set Δ, i.e. $\neg in(\rho, \Delta)$,
2. the states described by the symmetric case of 1, and
3. the states $(E_1''|[\Delta]|E_2'', X'')$, for which there exist EB^3 process expressions $E_1' \in S_{E_1}^Y$, $E_2' \in S_{E_2}^Y$, variable X', and action ρ such that a) $(E_1'|[\Delta]|E_2', X') \in S\ (X,Y)$ and $(E_1', X') \xrightarrow{\rho}_Y (E_1'', X'') \in \delta_{E_1}^Y$, $(E_2', X') \xrightarrow{\rho}_Y (E_2'', X'') \in \delta_{E_2}^Y$ on condition that the label of action ρ belongs to synchronization set Δ, i.e. $in(\rho, \Delta)$;
4. state $\{(\surd, X')$ is inserted into $S_{E_1|[\Delta]|E_2}$ provided that $(\surd|[\Delta]|\surd, X') \in S_{E_1|[\Delta]|E_2}$.

The procedure described in the previous paragraph is repeated as long as "$S'(X,Y) \neq S(X,Y)$". In other words, the iterative procedure terminates as soon as the corresponding computation reaches a fixed-point.

Notice also that the previous procedure creates a state space, whose upper bound is inferior to the product of state spaces S_{E_1} and S_{E_2}. Furthermore, state spaces S_{E_1} and S_{E_2} are finite, from which we deduce that termination is guaranteed.

The construction of $\delta_{E_1|[\Delta]|E_2}$ follows similar lines:

$$
\begin{array}{rcl}
\delta(X,Y) & = & \emptyset \\
\textbf{do} & \{ & \\
\delta(X,Y) & = & \delta'(X,Y); \\
\delta'(X,Y) & = & \delta\ (X,Y) \\
 & \bigcup & \big\{(E_1'|[\Delta]|E_2', X') \xrightarrow{\rho}_Y (E_1''|[\Delta]|E_2', X'') \mid (E_1'|[\Delta]|E_2', X') \in S(X,Y) \\
 & & \qquad \wedge\ \neg in(\rho,\Delta) \\
 & & \qquad \wedge\ (E_1', X') \xrightarrow{\rho}_Y (E_1'', X'') \in \delta^Y_{E_1}\big\} \\
 & \bigcup & \big\{(E_1'|[\Delta]|E_2', X') \xrightarrow{\rho}_Y (E_1'|[\Delta]|E_2'', X'') \mid (E_1'|[\Delta]|E_2', X') \in S(X,Y) \\
 & & \qquad \wedge\ \neg in(\rho,\Delta) \\
 & & \qquad \wedge\ (E_2', X') \xrightarrow{\rho}_Y (E_2'', X'') \in \delta^Y_{E_2}\big\} \\
 & \bigcup & \big\{(E_1'|[\Delta]|E_2', X') \xrightarrow{\rho}_Y (E_1''|[\Delta]|E_2'', X'') \mid (E_1'|[\Delta]|E_2', X') \in S(X,Y) \\
 & & \qquad \wedge\ in(\rho,\Delta) \\
 & & \qquad \wedge\ (E_1', X') \xrightarrow{\rho}_Y (E_1'', X'') \in \delta^Y_{E_1} \\
 & & \qquad \wedge\ (E_2', X') \xrightarrow{\rho}_Y (E_2'', X'') \in \delta^Y_{E_2}\big\} \\
 & \bigcup & \big\{(\surd|[\Delta]|\surd, X') \xrightarrow{\rho}_Y (\surd, X') \mid (\surd|[\Delta]|\surd, X') \in S(X,Y)\big\} \\
\} & & \\
\textbf{while} & & \big(\delta'(X,Y) \neq \delta(X,Y)\big)
\end{array}
$$

The initial state $s^0(X,Y)$ of $TS(X,Y)$ is $(E_1|[\Delta]|E_2, X)$.

It follows easily that:

1. $TS^{\mathsf{M}}_{E_1|[\Delta]|E_2} \doteq TS(\mathsf{M_0}, \mathsf{M})$ is the LNT describing $E_1|[\Delta]|E_2$ w.r.t. Sem_{M}.
2. $TS^{\mathsf{T}}_{E_1|[\Delta]|E_2} \doteq TS([\,], \mathsf{T})$ is the LNT describing $E_1|[\Delta]|E_2$ w.r.t. Sem_{T}.
3. $TS^{\mathsf{T/M}}_{E_1|[\Delta]|E_2} \doteq TS\big(([\,], \mathsf{M_0}), \mathsf{T/M}\big)$ is the LNT describing $E_1|[\Delta]|E_2$ w.r.t. $Sem_{\mathsf{T/M}}$.

- Let $E = E_0^*$.

  Let TS^Y denote the possibly infinite set of LTSs that simulate EB^3 specifications with respect to Sem_{M}, Sem_{T} and $Sem_{\mathsf{T/M}}$ depending on the value of $Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$ and let also $TS^Y_{E_x}$ denote the LTS of EB^3 process expression E_x with respect to Y.

  We define $F : TS^Y \to TS^Y$ function that receives $TS^Y_{E_x}$ as parameter and returns the union of $TS^Y_{E_0 \cdot E_x}$ and TS^Y_{λ}. Based on the meaning of identity formula "$E^* = \lambda \mid E.E^*$", $F\ (TS^Y_{E_x})$ shall express the possibility of executing E_0 followed by the eventual re-execution of $TS^Y_{E_x}$ or skipping process expression E completely.

  Hence, in order to obtain $TS_{E_0^*}$, we need to compute the least fix-point [Tar55] of function $F : TS^Y \to TS^Y$ with respect to the *lattice* $\mathcal{TS} = (TS^Y, \subseteq)$.

We consider LTS $TS(X,Y)$ defined as follows:

$$\begin{array}{rrcl} & TS(X,Y) & = & \big(S(X,Y),\ \delta(X,Y),\ s^0(X,Y)\big), \\ \text{where} & TS(X,Y) & = & lfp_F \\ & F(TS^Y_{E_x}) & = & TS^Y_{E_0 \cdot E_x} \bigcup TS^Y_\lambda \\ & s\ (X,Y) & = & (E_0^*, X) \end{array}$$

It follows easily that:

1. $TS^{\mathsf{M}}_{E_0^*} \doteq TS(\mathsf{M_0}, \mathsf{M})$ is the LNT describing E_0^* w.r.t. Sem_{M}.
2. $TS^{\mathsf{T}}_{E_0^*} \doteq TS([\,], \mathsf{T})$ is the LNT describing E_0^* w.r.t. Sem_{T}.
3. $TS^{\mathsf{T/M}}_{E_0^*} \doteq TS\big(([\,], \mathsf{M_0}), \mathsf{T/M}\big)$ is the LNT describing E_0^* w.r.t. $Sem_{\mathsf{T/M}}$.

- Let "$E = ge \Rightarrow E_0$".

  As usual, we aim at a generic formula:

$$TS\ (X,Y) = \big(S(X,Y),\ \delta(X,Y),\ s^0(X,Y)\big)$$

denoting the LTS that describes $ge \Rightarrow E_0$ with respect to Sem_{M}, Sem_{T} and $Sem_{\mathsf{T/M}}$ for suitable values of variable $X \in \{\mathsf{M_0}, [\,], ([\,], \mathsf{M_0})\}$ and $Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$.

The construction $TS(X,Y)$ relies on the possible values of expression $[\![ge]\!]^Y_3(X)$. Hence,

1. if $[\![ge]\!]^Y_3(X)$ evaluates to true, we apply similar reasoning as in the case of process expression $E_1 \,|\, E_2$.
2. if $[\![ge]\!]^Y_3(X)$ evaluates to false, $TS(X)$ reduces to the trivial LTS that contains state $(ge \Rightarrow E_0,\ X)$ and has no transitions.

More precisely, $TS\ (X,Y)$ is constructed as follows:

$$\begin{array}{rrcl} & TS(X,Y) & = & \big(S(X,Y),\ \delta(X,Y),\ s^0(X,Y)\big), \\ \text{where} & S(X,Y) & = & \begin{cases} \big\{(ge \Rightarrow E_0, X)\big\} \bigcup S^Y_{E_0} \setminus \big\{(E_0, X)\big\}, & \text{if } [\![ge]\!]^Y_3(X) = \mathsf{true} \\ \big\{(ge \Rightarrow E_0, X)\big\}, & \text{otherwise} \end{cases} \\ & \delta(X,Y) & = & \begin{cases} \delta^Y_{E_0}\big[(E_0, X) \leftarrow (ge \Rightarrow E_0, X)\big], & \text{if } [\![ge]\!]^Y_3(X) = \mathsf{true} \\ \emptyset, & \text{otherwise} \end{cases} \\ & s^0(X,Y) & = & (ge \Rightarrow E_0, X) \end{array}$$

It follows easily that:

1. $TS^{\mathsf{M}}_{ge \Rightarrow E_0} \doteq TS(\mathsf{M_0}, \mathsf{M})$ is the LNT describing $ge \Rightarrow E_0$ w.r.t. Sem_{M}.
2. $TS^{\mathsf{T}}_{ge \Rightarrow E_0} \doteq TS([\,], \mathsf{T})$ is the LNT describing $ge \Rightarrow E_0$ w.r.t. Sem_{T}.
3. $TS^{\mathsf{T/M}}_{ge \Rightarrow E_0} \doteq TS\big(([\,], \mathsf{M_0}), \mathsf{T/M}\big)$ is the LNT describing $ge \Rightarrow E_0$ w.r.t. $Sem_{\mathsf{T/M}}$.

- Let $E = P(\bar{t})$.

  Assuming the existence of *attribute function* definition:

  $$S ::= P(\overline{x} : \overline{T}) = E_0,$$

  (see Figure 2.1 for details), then, by simple substitution, the construction of $TS^{Y}_{P(\bar{t})}$ for $Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$ is reduced to:

  $$TS^{Y}_{P(\bar{t})} = TS^{Y}_{E_0\ [\overline{x}:=\overline{t}]}.$$

- Let "$E = \mathsf{|}\, x : V : E_0$".

  We extend the construction for $TS(E_1 \mathsf{|}\, E_2)$ considering all possible instantiations for EB^3 process expression E_0, i.e. $E_0[x := c]$, for all $c \in V \subseteq \mathcal{C}$.

  The generic formula denoting the LTS that describes $\mathsf{|}\, x : V : E_0$ with respect to Sem_{M}, Sem_{T} and $Sem_{\mathsf{T/M}}$ for variable $X \in \{\mathsf{M_0}, [\,], ([\,], \mathsf{M_0})\}$, $Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$ and set V is the following:

  $$\begin{array}{rrcl} & TS(X,Y,V) & = & \big(S(X,Y,V),\ \delta(X,Y,V),\ s^0(X,Y,V)\big) \\ \text{where} & S(X,Y,V) & = & \bigcup\limits_{c \in V} S^{Y}_{E_0[x:=c]} \setminus \{(E_0[x := c],\ X)\} \bigcup \big\{(\mathsf{|}\, x : V : E_0,\ X)\big\} \\ & \delta(X,Y,V) & = & \bigcup\limits_{c \in V} \delta^{Y}_{E_0[x:=c]}\big[E_0[x := c] \leftarrow \mathsf{|}\, x : V : E_0\big] \\ & s^0(X,Y,V) & = & (\mathsf{|}\, x : V : E_0,\ X) \end{array}$$

  if $V \neq \emptyset$. Moreover, $S(X,Y,\emptyset) = \{(\surd,\ X)\}$, $\delta(X,Y,\emptyset) = \emptyset$, and $s^0(X,Y,\emptyset) = (\surd,\ X)$.

  It follows easily that:

  1. $TS^{\mathsf{M}}_{\mathsf{|}\, x:V:E_0} \doteq TS(\mathsf{M_0}, \mathsf{M}, V)$ is the LNT describing $\mathsf{|}\, x : V : E_0$ w.r.t. Sem_{M}.
  2. $TS^{\mathsf{T}}_{\mathsf{|}\, x:V:E_0} \doteq TS([\,], \mathsf{T}, V)$ is the LNT describing $\mathsf{|}\, x : V : E_0$ w.r.t. Sem_{T}.
  3. $TS^{\mathsf{T/M}}_{\mathsf{|}\, x:V:E_0} \doteq TS\big(([\,], \mathsf{M_0}, V), \mathsf{T/M}\big)$ is the LNT describing $\mathsf{|}\, x : V : E_0$ w.r.t. $Sem_{\mathsf{T/M}}$.

- Let "$E = \mathsf{|[\Delta]|}\, x : V : E_0$".

  $$TS^{Y}_{\mathsf{|[\Delta]|}\, x:V:E_0} = (S^{Y}_{\mathsf{|[\Delta]|}\, x:V:E_0},\ \delta^{Y}_{\mathsf{|[\Delta]|}\, x:V:E_0},\ s^{0,Y}_{\mathsf{|[\Delta]|}\, x:V:E_0}) \quad \text{for} \quad Y \in \{\mathsf{M}, \mathsf{T}, \mathsf{T/M}\}$$

  Based on the values of V's cardinality, i.e. $|V|$ and the following formula:

  $$\mathsf{|[\Delta]|}\, x{:}V{:}E \quad = \quad \begin{cases} E[x := t]\ \mathsf{|[\Delta]|}\ (\mathsf{|[\Delta]|}\, x : V \setminus \{t\} : E_0)), & \text{if } |V| > 1 \\ E[x := t], & \text{if } |V| = 1 \\ \surd, & \text{otherwise} \end{cases}$$

  transition system $TS^{Y}_{\mathsf{|[\Delta]|}\, x:V:E_0}$ is constructed compositionally following the construction of $TS^{Y}_{E_1 \mathsf{|[\Delta]|}\, E_2}$.

2.4.3 Proof of Bisimulation Equivalence of Sem_{T}, $Sem_{\mathsf{T/M}}$ and Sem_{M}

First, we define the notion of EB^3 memory M *compatible* to given trace T. This definition is pivotal in establishing the equivalence between the three semantics.

Definition 2.4.3. *Let EB^3 be a well-formed* EB^3 *specification, let T be the trace variable and let M be an* EB^3 *memory. We say that memory M is compatible with trace T in EB^3 if and only if:*

$$\mathsf{M} = \mathit{compatible\_memory}(\mathsf{T}, \mathsf{M_0}),$$

where function compatible_memory is defined as follows:

$$\begin{array}{l}
\textbf{\textit{function}}\ \mathit{compatible_memory}(\mathsf{T} : \mathcal{T},\ \mathsf{M} : \mathcal{M}) \\
\textbf{\textit{begin}} \\
\quad \textbf{\textit{match}}\ \mathsf{T}\ \textbf{\textit{with}} \\
\qquad [\,] \Rightarrow \textbf{\textit{return}}\ \mathsf{M} \\
\qquad \mid \mathsf{T}'.\rho \Rightarrow \textbf{\textit{return}}\ \mathit{compatible_memory}(\mathsf{T}', \mathit{upd}(\rho, \mathsf{M})) \\
\quad \textbf{\textit{end match}} \\
\textbf{\textit{end}}
\end{array}$$

Let EB^3 be *well-formed* EB^3 specification. In the following, we refer to the LTS describing EB^3 with respect to Sem_{T} as TS_{T}. Moreover, notation S_{T} refers to the corresponding state space of TS_{T}, δ_{T} refers to the corresponding transition relation, and s^0_{T} refers to the initial state of the system.

Similarly, we refer to the LTS describing EB^3 with respect to $Sem_{\mathsf{T/M}}$ (Sem_{M}) as $TS_{\mathsf{T/M}}$ (TS_{M}). $S_{\mathsf{T/M}}$ (S_{M}) refers to the corresponding state space of $TS_{\mathsf{T/M}}$ (TS_{M}), $\delta_{\mathsf{T/M}}$ (δ_{M}) refers to the corresponding transition relation, and $s^0_{\mathsf{T/M}}$ (s^0_{M}) refers to the initial state of the system.

Corollary 2.4.1. *Let EB^3 be a well-formed* EB^3 *specification. For every state $(E, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$,* EB^3 *memory M is compatible to trace T.*

Proof.

- **Base Case:** By way of $(E_0, [\,], \mathsf{M_0}) \in S_{\mathsf{T/M}}$ and Definition 2.4.3, EB^3 memory $\mathsf{M_0}$ is found to be compatible with trace $[\,]$.
- **Inductive Hypothesis:** Assuming that for state $(E, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$, EB^3 memory M is compatible with trace T, then for any transition $(E, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$, it is "$\mathsf{M}' = \mathit{upd}(\rho, \mathsf{M})$" (see Figure 2.6 for details), from which follows that M' is compatible with $\mathsf{T}.\rho$. With similar reasoning, we visit all states in $S_{\mathsf{T/M}}$, which completes the proof. □

Theorem 2.4.1. *Let EB^3 be a well-formed* EB^3 *specification, $\tau \in Env_\pi$ be an environment, T be the trace variable, and M be a* EB^3 *memory compatible to T. Let also $v_1, \ldots, v_s$ denote a sequence of* EB^3 *type (1) value expressions.*

The denotation of EB^3 *type (1) value expression w_i^j corresponding to attribute function f_i appearing in EB^3 for $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, q\}$ with respect to Sem_{T} under environment τ and trace T is equal to the denotation of w_i^j with respect to $Sem_{\mathsf{T/M}}$ under environment τ and* EB^3 *memory M, i.e.*

$$[\![w_i^j]\!]_1^{\mathsf{T}}(\tau,\ \mathsf{T}) = [\![w_i^j]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}) \tag{2.39}$$

Proof. **Base case:** We set "$\mathsf{T} = [\,]$" and "$\mathsf{M} = \mathsf{M_0}$" in (2.39). Hence, it suffices to prove that:

$$[\![w_i^0]\!]_1^{\mathsf{T}}(\tau, [\,]) = [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \tag{2.40}$$

We proceed with structural induction on type (1) value expressions w_i^0.

- w_i^0 reduces to constant $c \in \mathcal{C}$:

$$\begin{aligned} [\![w_i^0]\!]_1^{\mathsf{T}}(\tau, [\,]) &= [\![c]\!]_1^{\mathsf{T}}(\tau, [\,]) \\ &= \mathcal{F}(c) \quad \text{(Definition 2.3.5)} \\ &= [\![c]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \quad \text{(Definition 2.3.8)} \\ &= [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \end{aligned}$$

  By (2.23) of Definition 2.3.5, $[\![f_i\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_1^{\mathsf{T}}(\tau, [\,])$ reduces to $\mathcal{F}(c)$. Then, by (2.33) of Definition 2.3.8), $\mathcal{F}(c)$ reduces to $[\![c]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})$, which completes the proof.

- w_i^0 reduces to variable $x \in \mathcal{V}$:

$$\begin{aligned} [\![w_i^0]\!]_1^{\mathsf{T}}(\tau,\ [\,]) &= [\![x]\!]_1^{\mathsf{T}}(\tau, [\,]) \\ &= \tau(x) \quad \text{(Definition 2.3.5)} \\ &= [\![x]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \quad \text{(Definition 2.3.8)} \\ &= [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \end{aligned}$$

  This case is similar to the previous case.

- w_i^0 reduces to non-conditional functional term $g(v_1, \ldots, v_l)$ for some EB^3 type (1) value expressions $v_1, \ldots, v_l$:

$$\begin{aligned} [\![w_i^0]\!]_1^{\mathsf{T}}(\tau, [\,]) &= [\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{T}}(\tau, [\,]) \\ &= \mathcal{F}(g)([\![v_1]\!]_1^{\mathsf{T}}(\tau, [\,]), \ldots, [\![v_l]\!]_1^{\mathsf{T}}(\tau, [\,])) \quad \text{(Definition 2.3.5)} \\ &= \mathcal{F}(g)([\![v_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}), \ldots, [\![v_l]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})) \quad \text{(Induction Hypothesis)} \\ &= [\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \quad \text{(Definition 2.3.8)} \\ &= [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \end{aligned}$$

  Remark that the inductive principle regarding the base case and (2.40) is applied on type (1) value expressions $v_1, \ldots, v_l$ that are "structurally smaller" than w_i^0. The rest of the proof is straightforward.

- w_i^0 reduces to *attribute function* call $f_h\,(\mathit{front}(\mathsf{T}), u_1, \ldots, u_s)$ for some $h \in \{1, \ldots, n\}$ and some EB^3 type (1) value expressions $u_1,\ \ldots,\ u_s$:

$$\begin{aligned} [\![w_i^0]\!]_1^{\mathsf{T}}(\tau, [\,]) &= [\![f_h\,(\mathit{front}(\mathsf{T}), u_1, \ldots, u_s)]\!]_1^{\mathsf{T}}(\tau, [\,]) \\ &= \perp^{\mathcal{F}} \quad \text{(Definition 2.3.5)} \\ &= \mathsf{M_0}(\mathsf{f_h})(\mathsf{e_1}, \ldots, \mathsf{e_s}) \quad \text{(Definition 2.3.9)} \\ &= [\![f_h\,(\mathit{front}(\mathsf{T}), v_1, \ldots, v_s)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \quad \text{(Definition 2.3.8)} \\ &= [\![w_i^0]\!]_1^{\mathsf{T}}(\tau, \mathsf{M_0}) \end{aligned}$$

where "$\mathsf{e_1} = [\![u_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})$", ..., and "$\mathsf{e_s} = [\![u_s]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})$" for some constants $e_1, \ldots, e_s \in \mathcal{C}$. Note that $\mathsf{M_0}(\mathsf{f_h})(\mathsf{e_1}, \ldots, \mathsf{e_s})$ refers to the initial value assigned to *attribute variable* $\mathsf{f_h}[e_1, \ldots, e_s]$ by $\mathsf{M_0}$ (see Definition 2.3.9 for details on upd_0). Then, by force of (2.36), $\mathsf{M_0}(\mathsf{f_h})(\mathsf{e_1}, \ldots, \mathsf{e_s})$ reduces to $[\![f_h\,(\mathit{front}(\mathsf{T}), v_1, \ldots, v_s)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})$, which completes the proof.

- w_i^0 reduces to *attribute function* call $f_h\,(\mathsf{T}, u_1, \ldots, u_s)$ for $h < i$ and some EB^3 type (1) value expressions $u_1, \ldots, u_s$:

  $$[\![w_i^0]\!]_1^{\mathsf{T}}(\tau, [\,]) = [\![f_h\,(\mathsf{T}, u_1, \ldots, u_s)]\!]_1^{\mathsf{T}}(\tau, [\,]) = [\![f_h\,(\mathsf{T}, u_1, \ldots, u_s)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) = [\![w_i^0]\!]_1^{\mathsf{T}}(\tau, \mathsf{M_0}),$$

  which is justified as follows:

  $$\begin{aligned}
  [\![f_h\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_1^{\mathsf{T}}(\tau, [\,]) &= [\![w_h^0]\!]_1^{\mathsf{T}}(\tau^\star([\,]) \cup \tau', [\,]) \quad \text{(Definition 2.3.5)} \\
  &= [\![w_h^0]\!]_1^{\mathsf{M}}(\tau', \mathsf{M_0}) \\
  &= \mathsf{M_0}(\mathsf{f'_h})(\mathsf{e_1}, \ldots, \mathsf{e_s}) \quad \text{(Definition 2.3.9)} \\
  &= [\![f_h\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \quad \text{(Definition 2.3.8)},
  \end{aligned}$$

  where, according to Definition 2.3.5, it is "$\tau' = [y_1 \leftarrow [\![v_1]\!]_1^{\mathsf{T}}(\tau, [\,])$", ..., "$y_s \leftarrow [\![v_s]\!]_1^{\mathsf{T}}(\tau, [\,])]$" and "$\mathsf{e_1} = [\![u_1]\!]_1^{\mathsf{M}}(\tau,\ \mathsf{M_0})$", ..., "$\mathsf{e_s} = [\![u_s]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})$" for some constants e_1, ..., $e_s \in \mathcal{C}$. The following proposition:

  $$[\![w_h^0]\!]_1^{\mathsf{T}}(\tau', [\,]) = [\![w_h^0]\!]_1^{\mathsf{M}}(\tau', \mathsf{M_0}) \tag{2.41}$$

  is obtained by replacing each occurrence of index i in (2.40) with h, for which $h < i$. Based on the observation that h is an integer, repeating the previous calculations and applying inductive reasoning, the correctness of (2.41) boils down to the correctness of the following proposition:

  $$[\![w_1^0]\!]_1^{\mathsf{T}}(\tau', [\,]) = [\![w_1^0]\!]_1^{\mathsf{M}}(\tau', \mathsf{M_0}). \tag{2.42}$$

  However, according to *attribute function* ordering, w_1^0 cannot reduce further to any *attribute function* call $f_k(\mathsf{T}, \ldots)$ for $k > 0$, which means that the proof steps taken so far to establish (2.40) suffice to establish (2.41).

- w_i^0 reduces to conditional term "**if** $g(v_1, \ldots, v_l)$ **then** v_{l+1} **else** v_{l+2}".

  1. Let "$[\![g\,(v_1, \ldots, v_l)]\!]_1^{\mathsf{T}}\ (\tau,\ [\,]) = \mathsf{true}$". The inductive principle applies to the "syntactically smaller" value expression $g\,(v_1, \ldots, v_l)$, which allows us to write:

     $$[\![g\,(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau, [\,]) = [\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{T}}(\tau, \mathsf{M_0}).$$

     or, equivalently, "$[\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) = \mathsf{true}$".

     $$\begin{aligned}
     [\![w_i^0]\!]_1^{\mathsf{T}}(\tau, [\,]) &= [\![\textbf{if } g(v_1, \ldots, v_l) \textbf{ then } v_{l+1} \textbf{ else } v_{l+2}]\!]_1^{\mathsf{T}}(\tau, [\,]) \\
     &= [\![v_{l+1}]\!]_1^{\mathsf{T}}(\tau, [\,]) \quad \text{(Definition 2.3.5)} \\
     &= [\![v_{l+1}]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \quad \text{(Induction Hypothesis)} \\
     &= [\![\textbf{if } g(v_1, \ldots, v_l) \textbf{ then } v_{l+1} \textbf{ else } v_{l+2}]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) \quad \text{(Definition 2.3.8)} \\
     &= [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})
     \end{aligned}$$

Notice also how the inductive hypothesis applies previously on v_{l+1}:

$$[\![v_{l+1}]\!]_1^{\mathsf{T}}(\tau, [\,]) = [\![v_{l+1}]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0})$$

2. Let "$[\![g\,(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau,\ [\,]) = \mathsf{false}$". Then, the proof is similar.

Induction Hypothesis: Let trace $\mathsf{T_1}$ such that $\mathsf{T_1}$ is a (strict) prefix of trace T denoted as $\mathsf{T_1} \prec \mathsf{T}$, and $\mathsf{M_1}$ is an EB^3 memory compatible to $\mathsf{T_1}$. We assume the following induction hypothesis:

$$[\![w_i^j]\!]_1^{\mathsf{T}}(\tau, \mathsf{T_1}) = [\![w_i^j]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_1}) \tag{2.43}$$

Now, let trace T be equal to "$\mathsf{T_1}.\alpha_j(c_1, \ldots, c_p)$" for some $j \in \{1, \ldots, q\}$ and some constants $c_1, \ldots, c_s \in \mathcal{C}$. Let also $\mathsf{M_1}$ be the compatible EB^3 memory to $\mathsf{T_1}$. We proceed with structural induction on type (1) value expressions w_i^j (see Definition 2.2.3 for details). It is fairly easy to see that the proofs regarding c, x, $g(v_1, \ldots, v_n)$, and "**if** $g(v_1, \ldots, v_l)$ **then** v_{l+1} **else** v_{l+2}" are similar to the corresponding proofs for the base case. In the following, we treat the remaining cases namely $f_h\,(\mathsf{T}, u_1, \ldots, u_s)$ and $f_h\,(\mathit{front}\,(\mathsf{T}), v_1, \ldots, v_s)$.

- w_i^j reduces to *attribute function* call "$f_h\,(\mathsf{T},\, u_1, \ldots, u_s)$" for $h < i$.

  $$[\![w_i^j]\!]_1^{\mathsf{T}}(\tau, \mathsf{T}) = [\![f_h\,(\mathsf{T}, u_1, \ldots, u_s)]\!]_1^{\mathsf{T}}(\tau, \mathsf{T}) = [\![f_h\,(\mathsf{T}, u_1, \ldots, u_s)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}) = [\![w_i^j]\!]_1^{\mathsf{T}}(\tau, \mathsf{M})$$

  which is justified as follows:

  $$\begin{aligned}
  [\![f_h\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_1^{\mathsf{T}}(\tau, \mathsf{T}) &= [\![w_h^j]\!]_1^{\mathsf{T}}(\tau^\star(\mathsf{T}) \cup \tau', \mathsf{T}) \quad \text{(Definition 2.3.5)} \\
  &= [\![w_h^j]\!]_1^{\mathsf{M}}(\tau^\star(\mathsf{T}) \cup \tau', \mathsf{M}) \\
  &= \mathsf{M}(\mathsf{f_h'})(\mathsf{e_1}, \ldots, \mathsf{e_s}) \quad \text{(Definition 2.3.10)} \\
  &= [\![f_h\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}) \quad \text{(Definition 2.3.8)}
  \end{aligned}$$

  where, according to Definition 2.3.5, it is "$\tau' = [y_1 \leftarrow [\![v_1]\!]_1^{\mathsf{T}}(\tau, \mathsf{T}),\ \ldots,\ "y_s \leftarrow [\![v_s]\!]_1^{\mathsf{T}}(\tau, \mathsf{T})]$" and "$\mathsf{e_1} = [\![u_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M})$", ..., "$\mathsf{e_s} = [\![u_s]\!]_1^{\mathsf{M}}(\tau, \mathsf{M})$" for some constants $e_1, \ldots, e_s \in \mathcal{C}$. The following proposition:

  $$[\![w_h^j]\!]_1^{\mathsf{T}}(\tau', \mathsf{T}) = [\![w_h^j]\!]_1^{\mathsf{M}}(\tau', \mathsf{M}) \tag{2.44}$$

  is obtained by replacing each occurrence of index i in (2.40) with h, for which $h < i$. Based on the observation that h is an integer, repeating the previous calculations and applying inductive reasoning, the correctness of (2.41) boils down to the correctness of the following proposition:

  $$[\![w_1^j]\!]_1^{\mathsf{T}}(\tau', \mathsf{T}) = [\![w_1^j]\!]_1^{\mathsf{M}}(\tau', \mathsf{M}) \tag{2.45}$$

  However, according to *attribute function* ordering, w_1^0 cannot reduce further to any *attribute function* call $f_k(\mathsf{T}, \ldots)$ for $k > 0$, which means that the proof steps taken so far to establish (2.40) suffice to establish (2.41).

- w_i^j reduces to *attribute function* call $f_h\,(\mathit{front}\ (\mathsf{T}),\ u_1,\ \ldots,\ u_s)$ for some $h < i$ and some EB^3 type (1) value expressions $u_1,\ \ldots,\ u_s$:

$$\begin{aligned}
[\![w_i^j]\!]_1^{\mathsf{T}}(\tau,\mathsf{T}) &= [\![f_h\,(\mathit{front}\ (\mathsf{T}),u_1,\ldots,u_s)]\!]_1^{\mathsf{T}}(\tau,\mathsf{T}) \quad \text{(Definition 2.3.5)}\\
&= [\![f_h\,(\mathsf{T},u_1,\ldots,u_s)]\!]_1^{\mathsf{T}}(\tau,\mathsf{T}_1) \quad \text{(Definition 2.3.5)}\\
&= [\![f_h\,(\mathsf{T},v_1,\ldots,v_s)]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}_1) \quad \text{(Inductive Hypothesis)}\\
&= \mathsf{M}_1(\mathsf{f}'_{\mathsf{h}})(\mathsf{e}_1,\ldots,\mathsf{e}_s) \quad \text{(Definition 2.3.8)}\\
&= \mathsf{M}(\mathsf{f}_{\mathsf{h}})(\mathsf{e}_1,\ldots,\mathsf{e}_s)\\
&= [\![f_h\,(\mathit{front}\,(\mathsf{T}),v_1,\ldots,v_s)]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) \quad \text{(Definition 2.3.8)}\\
&= [\![w_i^j]\!]_1^{\mathsf{M}}(\tau,\mathsf{M})
\end{aligned}$$

  where, for some constants $e_1,\ldots,e_s \in \mathcal{C}$, it is "$\mathsf{e}_1 = [\![u_1]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}_1)$", ..., and "$\mathsf{e}_s = [\![u_s]\!]_1^{\mathsf{M}}\,(\tau,\mathsf{M}_1)$". Note that the denotation of $f_h\,(\mathit{front}\,(\mathsf{T}),u_1,\ldots,u_s)$ under environment τ and trace T reduces to the denotation of $f_h\,(\mathsf{T},u_1,\ldots,u_s)$ under environment τ and trace T_1, which is a strict prefix of T. This is justified by (2.27) of Definition 2.3.8. The inductive principle (2.43) is then applied. Note also the replacement of $\mathsf{M}_1(\mathsf{f}'_{\mathsf{h}})$ by $\mathsf{M}(\mathsf{f}_{\mathsf{h}})$ in the previous calculations, since "$\mathsf{M} = \mathit{upd}(\alpha_j(c_1,\ldots,c_p),\mathsf{M}_1)$", which means that the *attribute variables* f_{i} of M are equal to the primed *attribute variables* f'_{i} of M_1 (see Definition 2.3.10 for details)[1].

□

Corollary 2.4.2. *Let EB^3 be a well-formed* EB^3 *specification, $\tau \in \mathit{Env}_\pi$ be an environment,* T *be the trace variable, and* M *be a* EB^3 *memory compatible to* T*. For* EB^3 *type (1) value expressions $v_1,\ldots,v_s$ appearing in EB^3, it follows that:*

$$[\![f_i\,(\mathsf{T},u_1,\ldots,u_s)]\!]_1^{\mathsf{T}}(\tau,\mathsf{T}) = \mathsf{M}(\mathsf{f}'_{\mathsf{i}})(\mathsf{e}_1,\ldots,\mathsf{e}_s) \tag{2.46}$$

where, for some constants $e_1,\ldots,e_s \in \mathcal{C}$, it is "$\mathsf{e}_1 = [\![u_1]\!]_1^{\mathsf{M}}(\tau,\mathsf{M})$", ..., and "$\mathsf{e}_s = [\![u_s]\!]_1^{\mathsf{M}}(\tau,\mathsf{M})$".

Proof. We set "$w_i^j = f_i\,(\mathsf{T},u_1,\ldots,u_s)$". From Theorem 2.4.1, it follows directly that:

$$[\![f_i\,(\mathsf{T},u_1,\ldots,u_s)]\!]_1^{\mathsf{T}}(\tau,\mathsf{T}) = [\![f_i\,(\mathsf{T},u_1,\ldots,u_s)]\!]_1^{\mathsf{M}}(\tau,\mathsf{M}) \tag{2.47}$$

By means of (2.37) of Definition 2.3.8, (2.47) and transitivity, we establish (2.46). □

Theorem 2.4.2. *Let EB^3 be a well-formed* EB^3 *specification,* T *be the trace variable, and* M *be a* EB^3 *memory compatible to* T*. For* EB^3 *guard ge appearing in EB^3, it follows that:*

$$[\![ge]\!]_3^{\mathsf{T}}(\mathsf{T}) = [\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}).$$

Proof. We proceed with structural induction on guard *ge*:

[1] the use of *attribute variables* f_{i} and primed *attribute variables* f'_{i} in the previous formulas is justified by the fact that the denotation "$[\![\cdot]\!]_1^{\mathsf{M}}(\tau,\mathsf{M})$" of EB^3 type (1) value expressions under τ and M is implicated mainly in the evaluation of *attribute variables*, where f_{i} and f'_{i} necessarily co-exist in M (see corresponding Definition 2.3.9 and Definition 2.3.10 for details).

- ge reduces to constant $c \in \mathcal{C}$. By way of Definitions 2.3.12, 2.3.7, we complete the proof:

$$[\![ge]\!]_3^{\mathsf{T}}(\mathsf{T}) = [\![c]\!]_3^{\mathsf{T}}(\mathsf{T}) = \mathcal{F}(c) = [\![c]\!]_3^{\mathsf{M}}(\mathsf{M}) = [\![ge]\!]_3^{\mathsf{M}}(\mathsf{M})$$

- ge reduces to functional term of the form $g(ge_1, \ldots, ge_n)$.

$$\begin{aligned} [\![g\,(ge_1, \ldots, ge_l)]\!]_3^{\mathsf{T}}(\mathsf{T}) &= \mathcal{F}(g)([\![ge_1]\!]_3^{\mathsf{T}}(\mathsf{T}), \ldots, [\![ge_l]\!]_3^{\mathsf{T}}(\mathsf{T})) \quad \text{(Definition 2.3.7)} \\ &= \mathcal{F}(g)([\![ge_1]\!]_3^{\mathsf{M}}(\mathsf{M}), \ldots, [\![ge_l]\!]_3^{\mathsf{M}}(\mathsf{M})) \quad \text{(Induction Hypothesis)} \\ &= [\![g\,(ge_1, \ldots, ge_l)]\!]_3^{\mathsf{M}}(\mathsf{M}) \quad \text{(Definition 2.3.12)} \end{aligned}$$

- ge reduces to an *attribute function* call of the form "$f_i\,(\mathsf{T}, v_1, \ldots, v_s)$". The result follows from Corollary 2.4.2.

□

Theorem 2.4.3. *Let EB^3 be a well-formed* EB^3 *specification. Let TS_{T} denote the LTS describing EB^3 with respect to Sem_{T} and $TS_{\mathsf{T/M}}$ denote the LTS describing EB^3 with respect to $Sem_{\mathsf{T/M}}$, then TS_{T} and $TS_{\mathsf{T/M}}$ are equivalent with respect to bisimulation.*

Proof. We define the candidate bisimulation relation as follows:

$$\begin{aligned} R = \big\{\big\langle (E, \mathsf{T}, \mathsf{M}), (E, \mathsf{T})\big\rangle \;\mid\; & E \text{ is a process expression, } \mathsf{T} \text{ is a } \textit{trace}, \\ & \mathsf{M} \text{ is an EB}^3 \text{ memory and } \mathsf{M} \text{ is compatible to } \mathsf{T}\big\}. \end{aligned} \tag{2.48}$$

Note that the definition of relation R is not restricted to tuples $(E, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and $(E, \mathsf{T}) \in S_{\mathsf{T}}$. We recall that $s^0_{\mathsf{T/M}} = (E, [\,], \mathsf{M_0})$ and $s^0_{\mathsf{T}} = (E, [\,])$. As EB^3 memory $\mathsf{M_0}$ is compatible with trace $[\,]$, it follows that $\big\langle (E, [\,], \mathsf{M_0}), (E, [\,])\big\rangle \in R$.

Let $(E, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and $(E, \mathsf{T}) \in S_{\mathsf{T}}$ such that $\langle (E, \mathsf{T}, \mathsf{M}), (E, \mathsf{T})\rangle \in R$, where E is an EB^3 process expression describing the remaining behaviour of system EB^3, T is the *trace* related to E and M is the EB^3 memory related to E. Let also $(E, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$ for action $\rho \in \{\alpha_j(c_1, \ldots, c_p) \mid j \in 1..q\} \cup \lambda$ with $c_1, \ldots, c_p \in \mathcal{C}$, where E' is an EB^3 process expression and M' is the EB^3 memory related to E'. We need to prove that there is transition $(E, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ with $\langle (E', \mathsf{T}.\rho, \mathsf{M}'), (E', \mathsf{T}.\rho)\rangle \in R$ and the converse.

The construction of $S_{\mathsf{T/M}}$ and S_{T} is carried out as described in Section 2.4.2. We proceed with structural induction on process expression E:

- Let $E = \rho \in \{\alpha_j(c_1, \ldots, c_p) \mid j \in 1..q\}$ for constants $c_1, \ldots, c_p \in \mathcal{C}$. We consider state $(\rho, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(\rho, \mathsf{T}) \in S_{\mathsf{T}}$. Note that M is compatible with T (see Corollary 2.4.1 for details), from which follows that $\langle (\rho, \mathsf{T}, \mathsf{M}), (\rho, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(\rho, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (\surd, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. By Rule ($\mathrm{M}_1$) (see Figure 2.6), it follows that "$\mathsf{M}' = \mathit{upd}(\rho, \mathsf{M})$". By Rule ($\mathrm{T}_1$) (see Figure 2.3), there should be transition $(\rho, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (\surd, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Moreover, M is compatible with T, from which follows that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3). Hence, $\langle (\surd, \mathsf{T}.\rho, \mathsf{M}'), (\surd, \mathsf{T}.\rho)\rangle \in R$.
  2. ($\Leftarrow$) Conversely, let transition $(\rho, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (\surd, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. By Rule ($\mathrm{M}_1$) (see Figure 2.6) there should be transition $(\rho, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (\surd, \mathsf{T}.\rho, \mathit{upd}(\rho, \mathsf{M})) \in \delta_{\mathsf{T}}$. Moreover, M is compatible with T, from which follows that $\mathit{upd}(\rho, \mathsf{M})$ is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3). Hence, $\langle (\surd,\ \mathsf{T}.\rho, \mathit{upd}(\rho, \mathsf{M})), (\surd, \mathsf{T}.\rho)\rangle \in R$.

- Let $E = \lambda$. We consider state $(\lambda, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(\lambda, \mathsf{T}) \in S_{\mathsf{T}}$. Again, M is compatible with T (see Corollary 2.4.1 for details), from which follows that $\langle(\lambda, \mathsf{T}, \mathsf{M}), (\lambda, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(\lambda, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M}) \in \delta_{\mathsf{T/M}}$. By Rule ($\mathrm{T}'_1$) (see Figure 2.3), there should be transition $(\rho, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd, \mathsf{T}) \in \delta_{\mathsf{T}}$. Moreover, M is compatible with T, from which follows that $\langle(\surd, \mathsf{T}, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$.
  2. ($\Leftarrow$) Conversely, let transition $(\lambda, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (\surd, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. By Rule ($\mathrm{M}'_1$) (see Figure 2.6), there should be transition $(\rho, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M}) \in \delta_{\mathsf{T/M}}$. Moreover, M is compatible with T, from which follows that $\langle(\surd, \mathsf{T}, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$.

- Let $E = E_1.E_2$. We consider state $(E_1.E_2, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(E_1.E_2, \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle(E_1.E_2, \mathsf{T}, \mathsf{M}), (E_1.E_2, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(E_1.E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1.E_2, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_2$) of Figure 2.6 implies premise $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\rho, \mathsf{M}')$[2]. Recall at this point that M is compatible with T, from which follows that $\langle(E_1, \mathsf{T}, \mathsf{M}), (E_1, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there should be transition $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1, \mathsf{T}.\rho)$ with $\langle(E'_1, \mathsf{T}.\rho, \mathsf{M}'), (E'_1, \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{T}_2$) of Figure 2.3, it follows that there should be transition $(E_1.E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1.E_2, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle(E'_1.E_2, \mathsf{T}.\rho, \mathsf{M}'), (E'_1.E_2, \mathsf{T}.\rho)\rangle \in R$, which completes the proof.
  2. ($\Leftarrow$) $\square$ Conversely, let transition $(E_1.E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1.E_2, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_2$) of Figure 2.3 implies premise $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_1, \mathsf{T}.\rho)$. Recall at this point that M is compatible with T, from which follows that $\langle(E_1, \mathsf{T}, \mathsf{M}), (E_1, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there should be transition $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\rho, \mathsf{M}')$ with $\langle(E'_1, \mathsf{T}.\rho, \mathsf{M}'), (E'_1, \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{M}_2$) of Figure 2.6, it follows that there should be transition $(E_1.E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1.E_2, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$ and, therefore, it should be $\langle(E'_1.E_2, \mathsf{T}.\rho, \mathsf{M}'), (E'_1.E_2, \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

  Now, we consider state $(\surd.E_2, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(\surd.E_2, \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle(\surd.E_2, \mathsf{T}, \mathsf{M}), (\surd.E_2, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(\surd.E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_2, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_3$) of Figure 2.6 implies premise $(E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_2, \mathsf{T}.\rho, \mathsf{M}')$. Recall at this point that M is compatible with T, from which follows that $\langle(E_2, \mathsf{T}, \mathsf{M}), (E_2, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there should be transition $(E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_2, \mathsf{T}.\rho)$ with $\langle(E'_2, \mathsf{T}.\rho, \mathsf{M}'), (E'_1, \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{T}_3$) of Figure 2.3, it follows that there should be transition $(\surd.E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E'_2, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle(\surd.E_2, \mathsf{T}.\rho, \mathsf{M}'), (E'_2, \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

[2] note that $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E'_1, \mathsf{T}.\rho, \mathsf{M}') \notin \delta_{\mathsf{T/M}}$

2. ($\Leftarrow$) Conversely, let transition $(\surd.E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_2', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_3$) of Figure 2.3 implies premise $(E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_2', \mathsf{T}.\rho)$. Recall at this point that M is compatible with T, from which follows that $\langle (E_2, \mathsf{T}, \mathsf{M}), (E_2, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

   Applying the induction hypothesis, there should be transition $(E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_2', \mathsf{T}.\rho, \mathsf{M}')$ with $\langle (E_2', \mathsf{T}.\rho, \mathsf{M}'), (E_2', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{M}_3$) of Figure 2.6, it follows that there should be transition $(\surd.E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_2', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$ and, therefore, it should be $\langle (E_2', \mathsf{T}.\rho, \mathsf{M}'), (E_2', \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

- Let $E = E_1 \,|\, E_2$. We consider state $(E_1 \,|\, E_2, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(E_1 \,|\, E_2, \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle (E_1 \,|\, E_2, \mathsf{T}, \mathsf{M}), (E_1 \,|\, E_2, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(E_1 \,|\, E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_4$) of Figure 2.6 implies premise $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}')$. Recall at this point that M is compatible with T, from which follows that $\langle (E_1, \mathsf{T}, \mathsf{M}), (E_1, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there should be transition $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1', \mathsf{T}.\rho)$ with $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'),\ (E_1',\ \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{T}_4$) of Figure 2.3, it follows that there should be transition $(E_1 \,|\, E_2,\ \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'), (E_1', \mathsf{T}.\rho)\rangle \in R$, which completes the proof.
  2. ($\Leftarrow$) Conversely, let transition $(E_1 \,|\, E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_2$) of Figure 2.3 implies premise $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1',\ \mathsf{T}.\rho)$. Recall at this point that M is compatible with T, from which follows that $\langle (E_1, \mathsf{T}, \mathsf{M}), (E_1, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there should be transition $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}')$ with $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'), (E_1', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{M}_4$) of Figure 2.6, it follows that there should be transition $(E_1 \,|\, E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$ and, therefore, it should be $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'), (E_1', \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

  The symmetric case is omitted for brevity.

- Let $E = E_0^*$. We consider state $(E_0^*, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(E_0^*, \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle (E_0^*, \mathsf{T}, \mathsf{M}), (E_0^*, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(E_0^*, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M}) \in \delta_{\mathsf{T/M}}$. Recall at this point that M is compatible with T, from which follows that $\langle (\surd, \mathsf{T}, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$.

     By way of rule (T_5) of Figure 2.3, there should be transition $(E_0^*, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd, \mathsf{T})$ with $\langle (\surd, \mathsf{T}, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$, which completes the proof.
  2. ($\Leftarrow$) Conversely, let transition $(E_0^*, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd,\ \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Recall at this point that M is compatible with T, from which follows that $\langle (\surd, \mathsf{T}, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$.

     By way of rule (M_5) of Figure 2.6, there should be transition $(E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M})$ with $\langle (\surd,\ \mathsf{T}.\rho, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$, which completes the proof.

1. ($\Rightarrow$) Let transition $(E_0^*, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0'.E_0^*, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_6$) of Figure 2.6 implies premise $(E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0', \mathsf{T}.\rho, \mathsf{M}')$. Recall at this point that M is compatible with T, from which follows that $\langle (E_0, \mathsf{T}, \mathsf{M}), (E_0, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

   Applying the induction hypothesis, there should be transition $(E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_0', \mathsf{T}.\rho)$ with $\langle (E_0', \mathsf{T}.\rho, \mathsf{M}'), (E_0', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{T}_6$) of Figure 2.3, it follows that there should be transition $(E_0^*, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_0'.E_0^*, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle (E_0'.E_0^*, \mathsf{T}.\rho, \mathsf{M}'), (E_0'.E_0^*, \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

2. ($\Leftarrow$) Conversely, let transition $(E_0^*, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_0'.E_0^*, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_6$) of Figure 2.3 implies premise $(E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_0', \mathsf{T}.\rho)$. Recall at this point that M is compatible with T, from which follows that $\langle (E_0, \mathsf{T}, \mathsf{M}), (E_0, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

   Applying the induction hypothesis, there should be transition $(E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0', \mathsf{T}.\rho, \mathsf{M}')$ with $\langle (E_0', \mathsf{T}.\rho, \mathsf{M}'), (E_0', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{M}_6$) of Figure 2.6, it follows that there should be transition $(E_0^*, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0'.E_0^*, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$ and, therefore, it should be $\langle (E_0'.E_0^*, \mathsf{T}.\rho, \mathsf{M}'), (E_0'.E_0^*, \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

- Let $E = E_1|[\Delta]|E_2$. We consider states $(E_1|[\Delta]|E_2, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and $(E_1|[\Delta]|E_2, \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle (E_1|[\Delta]|E_2, \mathsf{T}, \mathsf{M}), (E_1|[\Delta]|E_2, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(E_1|[\Delta]|E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1'|[\Delta]|E_2', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_7$) of Figure 2.6 implies $in(\rho, \Delta)$ and premises $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}')$, $(E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_2', \mathsf{T}.\rho, \mathsf{M}')$. Recall at this point that M is compatible with T, from which follows that $\langle (E_1, \mathsf{T}, \mathsf{M}), (E_1, \mathsf{T})\rangle \in R$, that $\langle (E_2, \mathsf{T}, \mathsf{M}), (E_2, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis twice, there should be transition $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1', \mathsf{T}.\rho)$ with $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'), (E_1', \mathsf{T}.\rho)\rangle \in R$ and transition $(E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_2', \mathsf{T}.\rho)$ with $\langle (E_2', \mathsf{T}.\rho, \mathsf{M}'), (E_2', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{T}_7$) of Figure 2.3 and $in(\rho, \Delta)$, it follows that there should be transition $(E_1|[\Delta]|E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1'|[\Delta]|E_2', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle (E_1'|[\Delta]|E_2', \mathsf{T}.\rho, \mathsf{M}'), (E_1'|[\Delta]|E_2',\ \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

  2. ($\Leftarrow$) Conversely, let transition $(E_1|[\Delta]|E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1'|[\Delta]|E_2', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_7$) of Figure 2.3 implies $in(\rho, \Delta)$ and premises $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1', \mathsf{T}.\rho)$, $(E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_2', \mathsf{T}.\rho)$. Recall at this point that M is compatible with T, from which follows that $\langle (E_1, \mathsf{T}, \mathsf{M}), (E_1, \mathsf{T})\rangle \in R$, that $\langle (E_2, \mathsf{T}, \mathsf{M}), (E_2, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there should be transition $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}')$ with $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'), (E_1', \mathsf{T}.\rho)\rangle \in R$ and $(E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_2', \mathsf{T}.\rho, \mathsf{M}')$ with $\langle (E_2', \mathsf{T}.\rho, \mathsf{M}'), (E_2', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{M}_7$) of Figure 2.6 and $in(\rho, \Delta)$, it follows that there should be transition $(E_1|[\Delta]|E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$ and, therefore, $\langle (E_1'|[\Delta]|E_2', \mathsf{T}.\rho, \mathsf{M}'), (E_1'|[\Delta]|E_2', \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

1. ($\Rightarrow$) Let transition $(E_1|[\Delta]|E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1'|[\Delta]|E_2, \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_8$) of Figure 2.6 implies $\neg in(\rho, \Delta)$ and premise $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}')$. M is compatible with T, from which follows that $\langle (E_1, \mathsf{T}, \mathsf{M}), (E_1, \ \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

   Applying the induction hypothesis, there should be transition $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1', \mathsf{T}.\rho)$ with $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'), (E_1', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{T}_8$) of Figure 2.3 and $\neg in(\rho, \Delta)$, it follows that there should be transition $(E_1|[\Delta]|E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1'|[\Delta]|E_2, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle (E_1'|[\Delta]|E_2, \mathsf{T}.\rho, \mathsf{M}'), (E_1'|[\Delta]|E_2, \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

2. ($\Leftarrow$) Conversely, let transition $(E_1|[\Delta]|E_2, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1'|[\Delta]|E_2, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_8$) of Figure 2.3 implies $in(\rho, \Delta)$ and premise $(E_1, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_1', \mathsf{T}.\rho)$. M is compatible with T, from which follows that $\langle (E_1, \mathsf{T}, \mathsf{M}), (E_1, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

   Applying the induction hypothesis, there should be transition $(E_1, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}')$ with $\langle (E_1', \mathsf{T}.\rho, \mathsf{M}'), (E_1', \mathsf{T}.\rho)\rangle \in R$. By Rule ($\mathrm{M}_8$) of Figure 2.6 and $\neg in(\rho, \Delta)$, it follows that there should be transition $(E_1|[\Delta]|E_2, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_1', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$ and, therefore, $\langle (E_1'|[\Delta]|E_2, \mathsf{T}.\rho, \mathsf{M}'), (E_1'|[\Delta]|E_2, \mathsf{T}.\rho)\rangle \in R$, which completes the proof.

The symmetric case is omitted for brevity.

Now, we consider state $(\surd|[\Delta]|\surd, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(\surd|[\Delta]|\surd, \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle (\surd|[\Delta]|\surd, \mathsf{T}, \mathsf{M}), (\surd|[\Delta]|\surd, \mathsf{T})\rangle \in R$.

1. ($\Rightarrow$) Let transition $(\surd|[\Delta]|\surd, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M}) \in \delta_{\mathsf{T/M}}$. Recall at this point that M is compatible with T, from which follows that $\langle (\surd, \mathsf{T}, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$.

   By way of Rule (T_9) of Figure 2.3, there should be transition $(\surd|[\Delta]|\surd, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd, \mathsf{T})$ with $\langle (\surd, \mathsf{T}, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$, which completes the proof.

2. ($\Leftarrow$) Conversely, let transition $(\surd|[\Delta]|\surd, \mathsf{T}) \xrightarrow{\lambda}_{\mathsf{T}} (\surd, \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Recall at this point that M is compatible with T, from which follows that $\langle (\surd, \mathsf{T}, \mathsf{M}), (\surd, \ \mathsf{T})\rangle \in R$.

   By way of rule (S_9) of Figure 2.6, there should be transition $(\surd|[\Delta]|\surd, \mathsf{T}, \mathsf{M}) \xrightarrow{\lambda}_{\mathsf{T/M}} (\surd, \mathsf{T}, \mathsf{M})$ with $\langle (\surd, \ \mathsf{T}.\rho, \mathsf{M}), (\surd, \mathsf{T})\rangle \in R$, which completes the proof.

- Let "$E = ge \Rightarrow E_0$". We consider state $(ge \Rightarrow E_0, \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(ge \Rightarrow E_0, \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle (ge \Rightarrow E_0, \mathsf{T}, \mathsf{M}), (ge \Rightarrow E_0, \mathsf{T})\rangle \in R$.

  1. ($\Rightarrow$) Let transition $(ge \Rightarrow E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_{10}$) of Figure 2.6 implies "$[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true}$" and premise $(E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0', \mathsf{T}.\rho, \mathsf{M}')$. M is compatible with T, from which follows that $\langle (E_0, \mathsf{T}, \mathsf{M}), (E_0, \mathsf{T})\rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there should be transition $(E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_0', \mathsf{T}.\rho)$ with $\langle (E_0', \mathsf{T}.\rho, \mathsf{M}'), (E_0', \mathsf{T}.\rho)\rangle \in R$. From Theorem 2.4.2 and "$[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true}$", we derive that "$[\![ge]\!]_3^{\mathsf{T}}(\mathsf{T}) = \mathsf{true}$". Then, by Rule ($\mathrm{T}_{10}$) of Figure 2.3 and "$[\![ge]\!]_3^{\mathsf{T}}(\mathsf{M}) = \mathsf{true}$", it follows that there should be transition $(ge \Rightarrow E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}}$

$(E_0', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle (E_0', \mathsf{T}.\rho, \mathsf{M}'), (E_0', \mathsf{T}.\rho) \rangle \in R$, which completes the proof.

2. ($\Leftarrow$) Let transition $(ge \Rightarrow E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_0', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_{10}$) of Figure 2.3 implies "$[\![ge]\!]_3^{\mathsf{T}}(\mathsf{T}) = \mathsf{true}$" and premise $(E_0, \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E_0', \mathsf{T}.\rho)$. M is compatible with T, from which follows that $\langle (E_0, \mathsf{T}, \mathsf{M}), (E_0, \mathsf{T}) \rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

   Applying the induction hypothesis, there should be transition $(E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0', \mathsf{T}.\rho, \mathsf{M})$ with $\langle (E_0', \mathsf{T}.\rho, \mathsf{M}'), (E_0', \mathsf{T}.\rho) \rangle \in R$. From Theorem 2.4.2 and "$[\![ge]\!]_3^{\mathsf{T}}(\mathsf{T}) = \mathsf{true}$", we derive that "$[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true}$". Then, by Rule ($\mathrm{M}_{10}$) of Figure 2.6 and "$[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true}$", it follows that there should be transition $(ge \Rightarrow E_0, \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E_0', \mathsf{T}.\rho, \mathsf{M}) \in \delta_{\mathsf{T/M}}$ and, therefore, it should be $\langle (E_0', \mathsf{T}.\rho, \mathsf{M}'), (E_0', \mathsf{T}.\rho) \rangle \in R$, which completes the proof.

- Let $E = P(\overline{u})$. We consider state $(P(\overline{u}), \mathsf{T}, \mathsf{M}) \in S_{\mathsf{T/M}}$ and state $(P(\overline{u}), \mathsf{T}) \in S_{\mathsf{T}}$, from which follows that $\langle (P(\overline{u}), \mathsf{T}, \mathsf{M}), (P(\overline{u}), \mathsf{T}) \rangle \in R$.

  1. ($\Rightarrow$) Let transition $(P(\overline{u}), \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E', \mathsf{T}.\rho, \mathsf{M}') \in \delta_{\mathsf{T/M}}$. Rule ($\mathrm{M}_{11}$) of Figure 2.6 implies the existence of process name definition "$P(\overline{x}) = E$" (see Figure 2.1 for details) and premise $(E[\overline{x} := \overline{u}], \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E', \mathsf{T}.\rho, \mathsf{M}')$. M is compatible with T, from which follows that $\langle (E[\overline{x} := \overline{u}], \mathsf{T}, \mathsf{M}), (E[\overline{x} := \overline{u}], \mathsf{T}) \rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there is transition $(E[\overline{x} := \overline{u}], \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E', \mathsf{T}.\rho)$ with $\langle (E', \mathsf{T}.\rho, \mathsf{M}'), (E', \mathsf{T}.\rho) \rangle \in R$. Then, by Rule ($\mathrm{T}_{11}$) of Figure 2.3 and process definition $P(\overline{x}) = E$, it follows that there should be transition $(P(\overline{u}), \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$ and, therefore, it should be $\langle (E', \mathsf{T}.\rho, \mathsf{M}'), (E', \mathsf{T}.\rho) \rangle \in R$, which completes the proof.
  2. ($\Rightarrow$) Let transition $(P(\overline{u}), \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E', \mathsf{T}.\rho) \in \delta_{\mathsf{T}}$. Rule ($\mathrm{T}_{11}$) of Figure 2.3 implies the existence of process name definition "$P(\overline{x}) = E$" and premise $(E[\overline{x} := \overline{u}], \mathsf{T}) \xrightarrow{\rho}_{\mathsf{T}} (E', \mathsf{T}.\rho)$. M is compatible with T, from which follows that $\langle (E[\overline{x} := \overline{u}], \mathsf{T}, \mathsf{M}), (E[\overline{x} := \overline{u}],\ \mathsf{T}) \rangle \in R$ and that M' is compatible with $\mathsf{T}.\rho$ (see Definition 2.4.3 for details).

     Applying the induction hypothesis, there is transition $(E[\overline{x} := \overline{u}], \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E', \mathsf{T}.\rho, \mathsf{M})$ with $\langle (E', \mathsf{T}.\rho, \mathsf{M}'), (E', \mathsf{T}.\rho) \rangle \in R$. Then, by Rule ($\mathrm{M}_{11}$) of Figure 2.6 and process definition "$P(\overline{x}) = E$", it follows that there should be transition $(P(\overline{u}), \mathsf{T}, \mathsf{M}) \xrightarrow{\rho}_{\mathsf{T/M}} (E', \mathsf{T}.\rho, \mathsf{M}) \in \delta_{\mathsf{T/M}}$ and, therefore, it also should be $\langle (E', \mathsf{T}.\rho, \mathsf{M}'), (E', \mathsf{T}.\rho) \rangle \in R$, which completes the proof.

□

Theorem 2.4.4. *$TS_{T/M}$ and TS_M are equivalent with respect to bisimulation.*

Proof. The proof is straightforward, because the effect of the trace on the attribute functions and the program execution is coded in the EB^3 memory M. Hence, intuitively the trace is redundant. □

Corollary 2.4.3. *TS_T and TS_M are equivalent with respect to bisimulation.*

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

2.5 Conclusion

We presented an alternative, traceless semantics Sem_{M} that we proved equivalent to the standard semantics Sem_{T}. In Chapter 4, we show 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.

Chapter 3

Experimentation with Petri Nets

3.1 Introduction

We recall that the goal of this thesis is to develop efficient techniques for the model-checking of temporal properties over EB^3 specifications. To this end, EB^3 specifications could be translated to LTSs that mimic the system behaviour (see Section 2.4.2 for details on the construction of bisimilar LTSs for EB^3 process expressions). Then, an appropriate model-checker could be used to verify the properties on the LTS level. Following another approach, the abstract syntax tree of EB^3 specification could be generated combining the EB^3 specification and Sem_{M}'s rewriting rules. Then, the model-checking would be done on the systax tree.

In practice, most of the times process algebra specifications are cast to intermediate representations such as Petri Nets (P/Ns) [Pet75], since direct translations of process expressions to LTSs or abstract syntax trees give usually complex and inefficient in terms of memory representations. Thanks to their operational semantics, P/Ns can be used to model systems composed of multiple parallel processes leading to efficient process representations eventually enhancing the verification process. P/Ns come in very handy if one needs to simulate process algebra features such as the non-deterministic closure of EB^3 expressions, i.e. E^*, and the guarded expressions containing attribute function calls. For these reasons, we prone for the translation of EB^3 process expressions to equivalent P/Ns.

However, the absence of efficient model-checking tools for the verification of P/Ns makes necessary the translation of derived P/N specifications to equivalent model-checking specifications. We turn our attention to the Nu-SMV model checker [CCG$^+$02] and consider the translation of intermediate P/N specifications to equivalent Nu-SMV specifications.

In this Chapter, we discuss the difficulties we encountered when translating EB^3 specifications to the corresponding P/N definitions and explain why the project of casting EB^3 to Nu-SMV was abandoned.

3.2 Petri Nets Basics

We adapt the theory on P/Ns to the EB^3 context.

Definition 3.2.1. *$N=(S,T,I,L)$ is an EB^3 Petri Net (denoted as EB^3 P/N) if and only if*

1. *S is a set of places denoting the state*
2. *$T \subseteq 2^S_+ \times \mathcal{G}E \times Act \times 2^S_+$ is a set of transitions*

3. $I \in S$ is the set of initial places,

where 2^S denotes the power set *over S and 2^S_+ denotes the* power set *over S from which $\emptyset$ is excluded. Note that $\mathcal{GE}$ is the set of* EB^3 guarded expressions *and Act is the set of actions occurring in the system.*

Let place $s \in S$. The *pre-set* ${}^\bullet s$ and *post-set* $s^\bullet$ of s are defined as follows:

$$ {}^\bullet s \doteq \{t \in T \mid tFs\} \text{ and } s^\bullet \doteq \{t \in T \mid sFt\}. $$

Let transition $t \in T$. The *pre-set* ${}^\bullet t$ and the *post-set* $t^\bullet$ of t are defined as follows:

$$ {}^\bullet t \doteq \{s \in S \mid sFt\} \text{ and } t^\bullet \doteq \{s \in S \mid tFs\}. $$

For $t \in T$, let "${}^\bullet t \doteq pr_1(t)$" be the *pre-set* of t, let "$ge(t) \doteq pr_2(t)$" be an EB^3 guard (trivially true in the absence of guard) and let "$l(t) \doteq pr_3(t)$" be the label of the current action and let "$t^\bullet \doteq pr_4(t)$" be the *post-set* of t. We define the *flow relation* F for given P/N N as follows:

$$ (x,y) \in F \doteq x \in S,\ y \in T \text{ and } {}^\bullet y(x) = 1 \text{ or } x \in T,\ y \in S \text{ and } x^\bullet(y) = 1 $$

In general, P/Ns are represented graphically with circles for P, boxes for T and arcs for F. Initial places I are indicated by placing a token on the corresponding circle to $s \in I$. We restrict our attention to *one-safe* P/Ns, i.e. places will never carry more than one token.

We consider *markings* $M \in 2^S$ that represent the places in the P/N that contain tokens. Let N be an EB^3 P/N. We denote by ${}^\circ N \doteq \{s \in S \mid {}^\bullet s = \emptyset\}$ and by $N^\circ \doteq \{s \in S \mid s^\bullet = \emptyset\}$ the set of places without *ingoing* arcs and the set of places without *outgoing* arcs respectively. $S_1 \rceil S_2$ denotes the *difference* between sets S_1 and S_2.

Definition 3.2.2. *Let $N = (S,T,I)$, $G : T \to \{0,1\}$ and $M, M' \in \mathbb{N}^S_+$. G is called a* step *from M to M' ($M[G\rangle M'$) iff*

1. $\forall s \in S,\ M(s) = \sum_{t \in T} {}^\bullet t(s).G(t)$ *(M enables G)*
2. $\forall s \in S,\ M'(s) = M(s) - \sum_{t \in T} {}^\bullet t(s).G(t) + \sum_{t \in T} t^\bullet(s).G(t)$

We stipulate that G is a *singleton* set, i.e. "$G = \{\{t\}\}$" for some $t \in T$. The P/N semantics above, which considers only occurences of single transitions, is called *interleaving semantics*. This remark allows us to write:

for all $\rho \in Act$, $M[\rho\rangle M'$ if and only if
$\exists t \in T$, $G : T \to \{0,1\}$ such that $G(t) = 1$ and $[\![ge(t)]\!]^{\mathsf{M}}_3(\mathsf{M}) = \mathsf{true}$,

where $[\![.]\!]^{\mathsf{M}}_3$ refers to the standard interpretation of EB^3 guards with respect to Sem_{M} and $\mathsf{M} \in \mathcal{M}$ is the current EB^3 memory that has been considered implicit in the previous definitions for brevity. In practice, the EB^3 P/N must keep track of the current EB^3 memory M for the evaluation of $ge(t)$. In particular, the Nu-SMV specification that simulates N must cater for stocking M somewhere, as well as for updating M every time transition t (see Definition 2.3.10 for details). Hence, the *interleaving semantics* of N is given by the LTS:

$$ \mathcal{A}(N) \doteq (\mathbb{N}^S_+, \to, I), \text{ where for } M, M' \in \mathbb{N}^S_+, \text{ it is } M \xrightarrow{\rho} M' \doteq M[\rho\rangle M' $$

3.3 Process Algebraic Operators and Petri Nets

The casting of *process algebra* operators into P/Ns appers to be a classic process algebraic problem. [GM84] provides P/N constructions for CCS with *guarded choice*, a subset of CCS expressions such that their corresponding P/N constructions preserve the operational semantics of CCS. The same issue is treated in [Gla87] within the *Algebra of Communicating Processes* (ACP) [BK84] setting.

We adapt these constructions to the EB^3 context. In the following, let N_1, N_2 be EB^3 P/Ns. Let also *op* be a binary process algebraic operator between EB^3 P/Ns. Then, "$N_1 op N_2$" will denote the corresponding P/N operation.
$\surd$ is represented by a single place s.

Definition 3.3.1. $N_{\surd} \doteq (\{s\},\ true,\ \emptyset,\ \{s\})$.

$\rho \in Act$ is represented as a box ρ connected to circles s_1 and s_2.

Definition 3.3.2. *Let* $\rho \in Act$. $N_\rho \doteq (\{s_1,\ s_2\},\ \{(\{s_1\},\ true,\ \rho, \{s_2\})\},\ \{s_1\})$.

The sequence of EB^3 P/Ns N_1, N_2 is obtained by taking the disjoint union of transitions in N_1 that do not belong to the *pre-set* of N_1°, i.e. ${}^\bullet N_1^\circ$ and transitions in N_2 that do not belong to $I_2^\bullet$, while combining places N_1° and I_2 in the *cartesian product* $N_1^\circ \times I_2$. More concretely:

Definition 3.3.3. *Let* $N_i = (S_i, T_i, I_i)$, $i = 1, 2$ *and* $S_1 \cap S_2 = \emptyset$.
$N_1.N_2 \doteq (S_1 \rceil N_1^\circ \ \cup\ N_1^\circ \times I_2 \ \cup\ S_2 \rceil I_2,\ T_1' \cup T_2' \cup T,\ I_1 \cup I_2)$, *where*

$$\begin{aligned} T = \ & \{{}^\bullet t,\ ge(t),\ l(t),\ \{t\} \times I_2) \mid t \in {}^\bullet N_1^\circ\} \ \cup\ T_2 \rceil I_2^\bullet \ \cup \\ & \{N_1^\circ \times \{t\},\ ge(t),\ l(t), t^\bullet) \mid t \in {}^\bullet N_1^\circ\} \ \cup\ T_1 \rceil {}^\bullet N_1^\circ. \end{aligned}$$

The *choice* $N_1 | N_2$ behaves either as N_1 or N_2. We merge I_1 and I_2 by use of the *cartesian product* $I_1 \times I_2$. We then disable all initial transitions in N_2, whenever any transition in N_1 takes place and vice versa.

Definition 3.3.4. *Let* $N_i = (S_i, T_i, I_i)$, $i = 1, 2$ *and* $S_1 \cap S_2 = \emptyset$.
$N_1 | N_2 \doteq (S_1 \rceil I_1 \ \cup\ S_2 \rceil I_2 \ \cup\ I_1 \times I_2,\ T_1' \cup T_2',\ I_1 \times I_2)$, *where*

$$\begin{aligned} T_1' &= \{(({}^\bullet t \cap I_1 \times I_2) \ \cup\ {}^\bullet t \rceil I_1,\ ge(t),\ l(t),\ t^\bullet) \mid t \in T_1\} \\ T_2' &= \{((I_1 \times\ {}^\bullet t \cap I_2) \ \cup\ {}^\bullet t \rceil I_2,\ ge(t),\ l(t),\ t^\bullet) \mid t \in T_2\}. \end{aligned}$$

The *parallel composition* of the EB^3 P/Ns N_1, N_2 with synchronisation on Δ is modelled as the disjoint union of transitions, whose labels do not belong to Δ enlarged by transitions t with $l(t) \in \Delta$ representing all possible synchronisations on $t_1 \in T_1$, $t_2 \in T_2$ with $l(t) = l(t_1) = l(t_2) \in \Delta$. $ge(t)$ will be equal, as expected, to the conjunction of $ge(t_1)$ and $ge(t_2)$, i.e. $ge(t_1) \wedge ge(t_2)$.

Definition 3.3.5. *Let* $N_i = (S_i, T_i, I_i)$, $i = 1, 2$ *and* $S_1 \cap S_2 = \emptyset$.
$N_1 | [\Delta] | N_2 \doteq (S_1 \cup S_2,\ T_1' \cup T_2' \cup T,\ I_1 \cup I_2)$, *where*

$$\begin{aligned} T_i' &= T_i \rceil \{t \mid t \in T_i,\ l(t) \in \Delta\},\ i = 1, 2 \\ T &= \{({}^\bullet t_1 \cup {}^\bullet t_2,\ ge(t_1) \wedge ge(t_2),\ l(t_1),\ t_1^\bullet \cup t_2^\bullet) \mid t_1 \in T_1,\ t_2 \in T_2,\ l(t_1) = l(t_2) \in \Delta\}. \end{aligned}$$

The *Kleene* Closure of the P/N N is obtained from N by removing all places N° and replacing each ingoing arc to N° by arcs to the initial places of the net. By virtue of the *expansion law* "$E^* = \mu x(E.x|\lambda)$" (μ is the standard *fix-point* operator), a new box λ, with *pre-set* the initial places I and *post-set* the *singleton* set $\{s\}$ where "$s^\bullet = \emptyset$", must be added to transitions T.

Definition 3.3.6. *Let* $N=(S,T,I)$.
$N^* \doteq (S \rceil N^\circ \biguplus \{s\},\ T \rceil {}^\bullet N^\circ \cup T' \cup \{(I, true, \lambda, \{s\})\},\ I)$, *where*

$T' = \{({}^\bullet t,\ ge(t),\ l(t),\ I) \mid t \in {}^\bullet N^\circ\}$.

The *guarded expression* P/N $ge \Rightarrow N$ is obtained from N by replacing for each $t \in I^\bullet$, $pr_2(t)$ with $ge \wedge ge(t)$.

Definition 3.3.7. *Let* $N=(S,T,I)$.
$ge \Rightarrow N \doteq (S,\ T \rceil I^\bullet \cup T',\ I)$, *where* $T' = \{({}^\bullet t,\ ge \wedge ge(t),\ l(t),\ I) \mid t \in I^\bullet\}$.

Let "$P(\overline{x}) = E$" be an EB^3 process definition. The EB^3 P/N N corresponding to E, the construction of the EB^3 P/N N' corresponding to $E[\overline{x} := \overline{t}]$ is obtained by relabelling transition labels according to the function $f : Act \to Act$, where "$f(\rho) \doteq \rho[x_i \leftarrow t_i \mid x_i \in \overline{x},\ t_i \in \overline{t}]$".

Definition 3.3.8. *Let* $N=(S,T,I)$.
$N' = (S, T', I)$, *where* $T' = \{({}^\bullet t,\ ge(t),\ f(l(t)),\ t^\bullet) \mid t \in T\}$.

Examples of the EB^3 P/N constructions above can be found in Figure 3.1 and Figure 3.2. For cases i) to iv), we consider that "$ge(t) = true$" for all transitions in the P/Ns involved. Whenever $ge(t)$ is not equal to *true*, it is indicated explicitly.

However, these constructions do not always produce correct P/Ns. Some cases that lead to erroneous EB^3 P/N constructions are depicted in Figure 3.3. For instance in Figure 3.3.vi), the step $M[a\rangle M'$ implies that EB^3 guard ge evaluates to true. Then executing action b involves re-evaluating ge, since the P/N construction algorithm propagates ge to all transitions $t \in I^\bullet$, i.e. transition t_1 of "$a \parallel b$" with "$l(t_1) = a$" and transition t_2 with "$l(t_2) = b$". This is the so-called EB^3 *guard-action atomicity problem*, which is elaborated and solved satisfactorily in Chapter 4. In Figure 3.3.vii), the EB^3 P/N corresponding to "$(a \parallel b)^* | c$" takes the step $M[b\rangle M'$ that fires action b and enables action c. The problem lies in the initial parallelism of the recursive component $(a \parallel b)^*$ with c, which creates ingoing arcs to initially marked places, i.e. s_1 and s_2. This problem is detected in [Win84].

Unfolding the EB^3 P/N, as is demonstrated in Figure 3.4.viii), is a possible solution to the problem of Figure 3.3.vii). The authors of [Gla87] use the term *cyclic roots* to designate initial places with ingoing arcs and propose an efficient *root unwinding* operation that relates to every P/N an equivalent P/N with *acyclic roots*.

Another problem results from Figure 3.3.vi) and Figure 3.3.vii). In the later case, $M[b\rangle M'$ gives "$M'(s) = 1$" and "$M'(s') = 2$", which violates the *one-safe* assumption of the previous section. A solution would be to restrict the domain of EB^3 expressions to a subset that gives EB^3 P/Ns with *acyclic roots*, i.e. $I = {}^\circ N$ and treats the problem of Figure 3.3.vi) correctly, but this would restrict EB^3's expressiveness considerably.

3.4 Nu-SMV Specific Difficulties

Apart from the difficulties discussed above, model-checking EB^3 specifications by means of the Nu-SMV model checker is hampered by a series of obstacles:

- First of all, Nu-SMV verifies models against branching time properties in (state-based) CTL [CEF$^+$86]. Hence, The need to verify an action-based process algebra by nature such as EB^3 against action-based properties, entails some syntactic acrobatics, because

classical action-based properties (for example, the requirements of the library management system) need to be translated to their equivalent state-based versions. However, the problem can be addressed and solved as in [DV90].

- Furthermore, Nu-SMV's close resemblance to low-level programming languages makes the automatic translation a strenuous and time-consuming task. The lack of recursive type definitions within the Nu-SMV specification language forces us to consider binary encodings for complex data structures in EB^3 such as lists and sets.
- Finally, the translation algorithm gives us complex Nu-SMV specifications for simple EB^3 specifications and poor verification performance.

3.5 Conclusion

We demonstrated that, the casting of EB^3 specifications into Nu-SMV involves a considerable amount of P/N unfoldings even for the most common process algebra operators like *. This observation is deemed to have a negative effect on the model-checking problem. We also argumented that the nature of the Nu-SMV model checker does not facilitate the translation of EB^3 specifications to the Nu-SMV specification language. For all the above reasons, the project of model-checking EB^3 specifications with the use of Nu-SMV was abandoned.

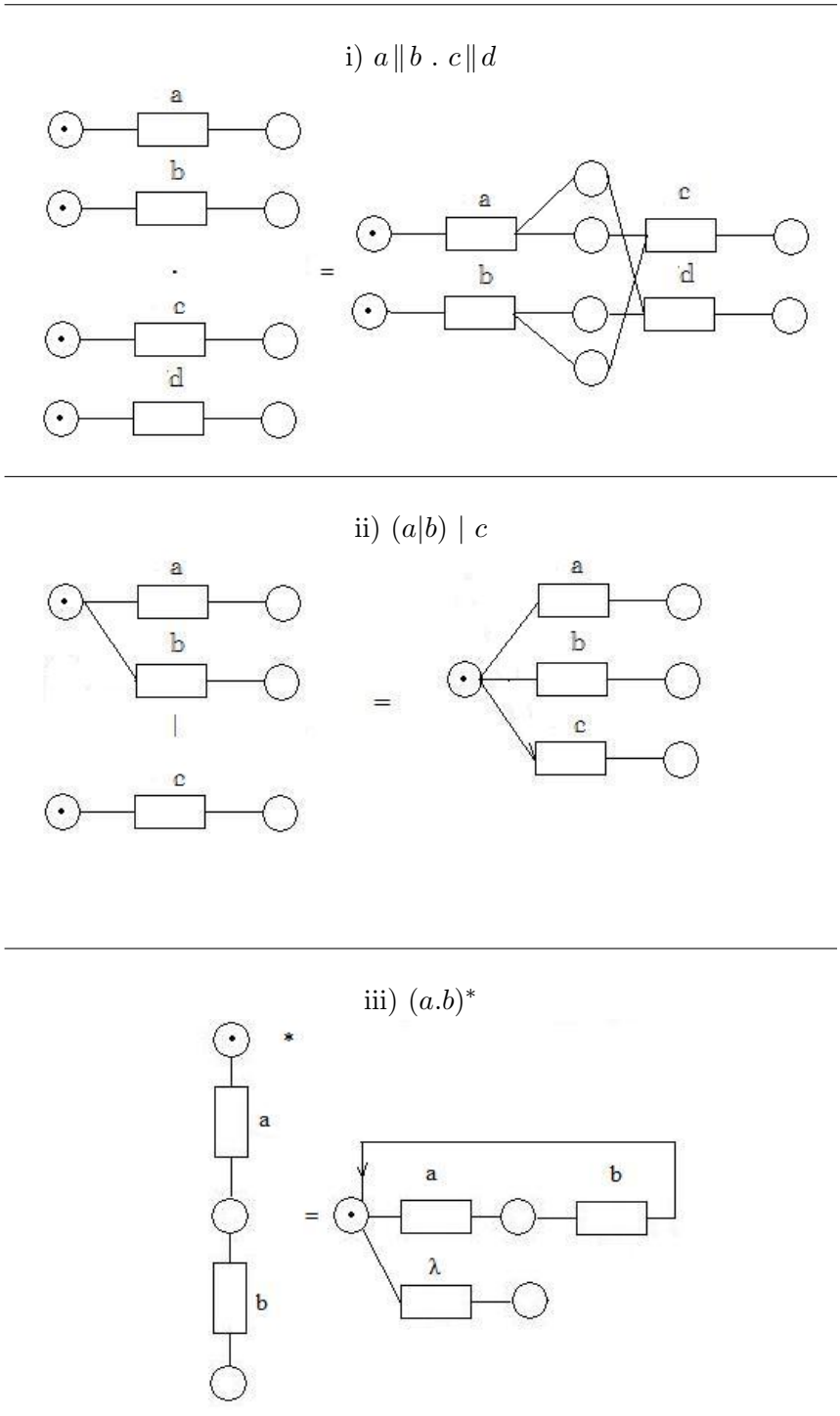
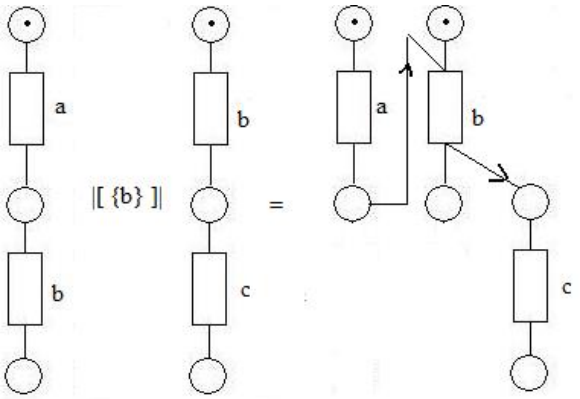


Figure 3.1: Examples of P/Ns (A)

iv) $a.b|[\{b\}]|b.c$



v) $ge \Rightarrow (b|b)$

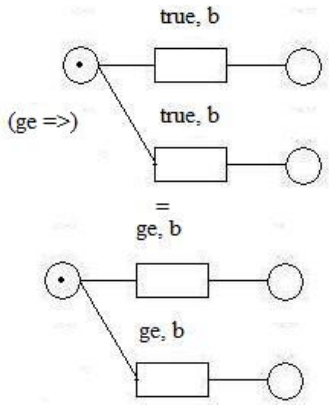
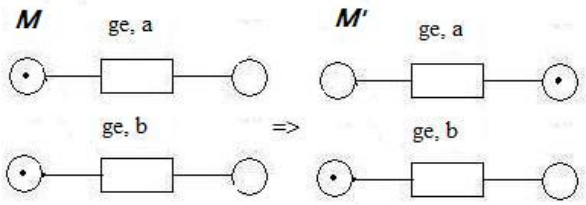


Figure 3.2: Examples of P/Ns (B)

vi) $ge \Rightarrow (a \parallel b)$



vii) $(a \parallel b)^* | c$

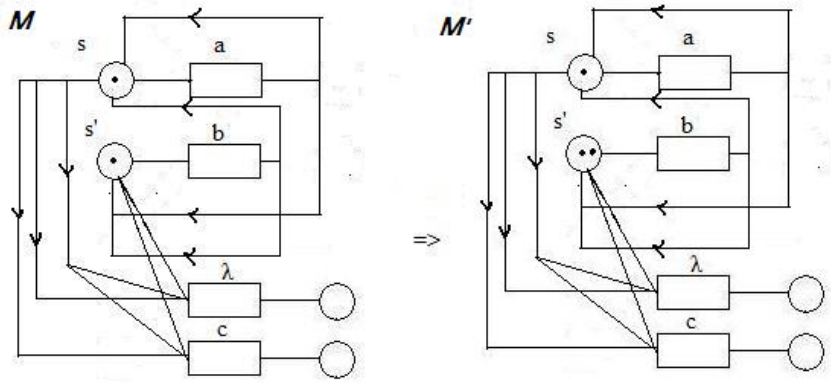


Figure 3.3: Erroneous P/N Constructions

viii) a^*

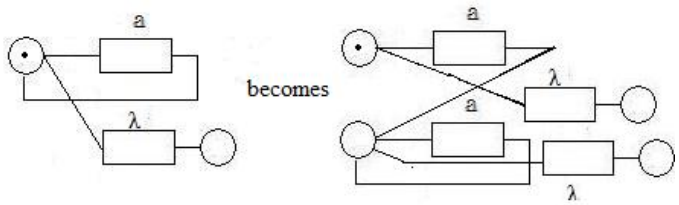


Figure 3.4: Unfoldings

Chapter 4

Translation of EB^3 to LOTOS-NT (LNT)

Based on the efficient memory Sem_{M} given in [VD13], we propose a rigorous translation algorithm from EB^3 to LNT [CCG$^+$11]. As far as we know, this is the first attempt to provide a general translation from EB^3 to a classical value-passing process algebra. It is worth noticing that CSP and LNT were already considered in [FFC$^+$10] for describing ISs, and identified as candidate target languages for translating EB^3. Another formal technique called CSP‖B [ST02] was used to describe ad hoc EB^3 specifications in [ETL$^+$04]. Since our primary objective was to provide temporal property verification features for EB^3, we focused our attention on LNT, which is one of the input languages accepted by the CADP verification toolbox [GLM$^+$11], and hence is equipped with on-the-fly model checking for action-based, branching-time logics involving data.

Another important ingredient of the translation was the multiway value-passing rendezvous of LNT, which enabled to obtain a one-to-one correspondence between the transitions of the two LTSs underlying the EB^3 and LNT descriptions, and hence to preserve strong bisimulation. The presence of array types and of usual programming language constructs (e.g., loops and conditionals) in LNT was also helpful for specifying the memory, the Kleene star-closure operators, and the EB^3 guarded expressions containing attribute function calls. At last, the constructed data types and pattern-matching mechanisms of LNT enabled a natural description of EB^3 data types and attribute functions.

We implemented our translation in the EB^{3}2LNT tool, thus making possible the analysis of EB^3 specifications using all the state-of-the-art features of the CADP toolbox, in particular the verification of data-based temporal properties expressed in MCL [MT08] using the on-the-fly model checker EVALUATOR 4.0.

4.1 CADP Tool

CADP (*Construction and Analysis of Distributed Processes*) is a toolbox used to verify asynchronous concurrent systems. Initially, CADP featured a compiler and explicit state space generator for the LOTOS language called Caesar and an bisimulation equivalence checker called Aldebaran.

Nowadays, CADP allows users to specify complex asynchronous systems, offers interactive simulation, rapid prototyping, verification (via model checking, equivalence checking), testing,

and performance evaluation. Verification is carried out by way of reachability analysis, on-the-fly verification, compositional verification, distributed verification and static analysis. Last but not least, CADP provides scripting languages for describing elaborated verification scenarios.

4.2 The Language LNT

LNT is an input language accepted by CADP. LNT aims at providing 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 a LNT specification.

We present the fragment of LNT that serves as the target of our translation. 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, written G, G_1, ..., G_n, and may be guarded using Boolean conditions on the received values (**where** clause). LNT allows multiway rendezvous with bidirectional (send/receive) value exchange on the same gate occurrence, each offer O being either a send value offer (`!`) or a receive value offer (`?`), independently of the other offers. Expressions E are built from variables, type constructors, function applications and constants. Labels L identify loops, which can be escaped using "**break** L" from inside the loop body. Processes are parameterized by gates and data variables. LNT syntax and semantics are formally defined in SOS style in [CCG+11].

4.2.1 Syntax and Dynamic Semantics of LNT

We present the syntax and the dynamic semantics (see [CCG+11]) of LNT programs using the formal *Structural Operational Semantics* (SOS) rules of LNT.

- Let $\mathcal{V}$ be a set of variable identifiers, let $\mathcal{C}$ be a set of type constructor identifiers and let $\mathcal{F}$ be a set of function identifiers. The syntax of LNT expressions is defined as follows:

$$v \quad ::= \quad x \quad \mid \quad C(v_1, \ldots, v_l) \quad \mid \quad F(v_1, \ldots, v_l),$$

  where $x \in \mathcal{V}$, $C \in \mathcal{V}$ is a user-defined or predefined constructor and $F \in \mathcal{V}$ is a user-defined or predefined function. The difference between constructor and function identifiers lies in that only the latter are evaluated. A typical constructor identifier is *cons* that takes an typed element and a list of typed elemets to create a list. As in the case of EB^3, we define the memory (LNT memory) as a partial function from variables to values. Let $\mathcal{M}_{\mathsf{LNT}}$ be the domain of LNT memories. The denotation of LNT expressions v under LNT memory M is defined as follows:

$$\begin{aligned} [\![x]\!]_{\mathsf{LNT}}(\mathsf{M}) &= \mathsf{M}(x) \\ [\![F(v_1, \ldots, v_l)]\!]_{\mathsf{LNT}}(\mathsf{M}) &= \mathsf{F}\big([\![v_1]\!]_{\mathsf{LNT}}\ (\mathsf{M}),\ \ldots,\ [\![v_l]\!]_{\mathsf{LNT}}\ (\mathsf{M})\big) \\ [\![C(v_1, \ldots, v_l)]\!]_{\mathsf{LNT}}(\mathsf{M}) &= C\big([\![v_1]\!]_{\mathsf{LNT}}\ (\mathsf{M}),\ \ldots,\ [\![v_l]\!]_{\mathsf{LNT}}\ (\mathsf{M})\big), \end{aligned}$$

  where F denotes the denotation of function identifier F. In particular, F is interpreted as a mathematical function applied to $[\![v_1]\!]_{\mathsf{LNT}}\ (\mathsf{M})$, ... and $[\![v_1]\!]_{\mathsf{LNT}}\ (\mathsf{M})$.

The dynamic semantics of an LNT expression v is defined alternatively as relation $\rightarrow_e$:

$$\{v\}\ \mathsf{M} \rightarrow_e [\![v]\!]_{\mathsf{LNT}}(\mathsf{M}), \tag{4.1}$$

whose meaning is that v evaluates under LNT memory M to $[\![v]\!]_{\mathsf{LNT}}(\mathsf{M})$. Remark that this notation is used widely in the following definitions.

The grammar of LNT statements is defined as follows:

$$I \ ::= \ \textbf{null} \ \mid \ I_1; I_2 \ \mid \ \textbf{break}\ L \ \mid \ \textbf{loop}\ L\ \textbf{in}\ I_0\ \textbf{end loop} \ \mid \ x := v,$$

where $x \in \mathcal{V}$ or x is an access to some user-defined array. The statements are used in the LNT behaviours defined later on. The dynamic semantics of an LNT statement I is defined as a relation of the form:

$$\{I\}\ \mathsf{M} \xrightarrow{\alpha}_s \mathsf{M}', \tag{4.2}$$

where α is a label, and M, M' are LNT memories related to statement I. The meaning is that statement I is executed under LNT memory M and LNT memory M is updated to memory M' after the execution. Label α can take one of the following forms:

- **exit**, which denotes normal execution of statement I. System execution carries on with the following instruction.
- **brk(L)**, which denotes forced termination of statement I via a "**break** L" statement appearing in statement I. System execution carries on with the instruction succeeding the loop that is marked with label L. The set of labels used to mark loops is denoted by $\mathcal{L}$ and the set $\{\textbf{brk}(L) \mid L \in \mathcal{L}\}$ is denoted by $\mathcal{B}_L$.
- **ret**(v), where v is an LNT expression, which denotes termination on **return** statements. The execution must carry on with the next instruction in the caller. The set $\{\textbf{ret}(v) \mid v \text{ is an LNT expression}\} \cup \{\textbf{ret}\}$ is denoted by $\mathcal{R}_V$.

Hence, upon substitution of statement I in relation (4.2), the dynamic semantics of LNT statements unfolds to the following rules:

- The **null** statement executes normally without changing the initial memory.

$$\frac{}{\{\textbf{null}\}\ \mathsf{M} \xrightarrow{\textbf{exit}}_s \mathsf{M}}$$

- Statement "$I_1; I_2$" starts by executing statement I_1. As soon as I_1 is executed, the system carries on with the execution of statement I_2.

$$\frac{\{I_1\}\ \mathsf{M} \xrightarrow{\textbf{exit}}_s \mathsf{M}' \quad \{I_2\}\ \mathsf{M}' \xrightarrow{\alpha}_s \mathsf{M}'' \quad \alpha \in \{\textbf{exit}\} \cup \mathcal{B}_L \cup \mathcal{R}_V}{\{I_1; I_2\}\ \mathsf{M} \xrightarrow{\alpha}_s \mathsf{M}''}$$

  If, by any chance, statement I_1 is terminated abruptly, i.e. via instruction **break**, then statement "$I_1; I_2$" terminates abruptly too.

$$\frac{\{I_1\}\ \mathsf{M} \xrightarrow{\textbf{brk}(L)}_s \mathsf{M}'}{\{I_1;\ I_2\}\ \mathsf{M} \xrightarrow{\textbf{brk}(L)}_s \mathsf{M}'}$$

– The assignment "$x := v$" terminates normally after updating the memory by assigning the denotation of value expression v to the left-hand variable.

$$\frac{\{v\}\ \mathsf{M} \rightarrow_e \mathsf{v}}{\{x := v\}\ \mathsf{M} \xrightarrow{\mathbf{exit}}_s \mathsf{M} \oplus [\mathsf{x} \leftarrow \mathsf{v}]},$$

where $\oplus$ is the standard update operator defined in Chapter 2.

– Statement "**break** L" terminates and passes label L to the context. The memory does not change.

$$\frac{}{\{\mathbf{break}\ L\}\ \mathsf{M} \xrightarrow{\mathbf{brk}(L)}_s \mathsf{M}}$$

– The breakable loop "**loop** L **in** I_0 **end loop**" starts by executing statement I_0. If I_0 terminates normally, then the breakable loop is executed once again.

$$\frac{\{I_0\}\ \mathsf{M} \xrightarrow{\mathbf{exit}}_s \mathsf{M}' \quad \{\mathbf{loop}\ L\ \mathbf{in}\ I_0\ \mathbf{end\ loop}\}\ \mathsf{M}' \xrightarrow{\alpha}_s \mathsf{M}'' \quad \alpha \in \{\mathbf{exit}\} \cup \mathcal{B}_L \cup \mathcal{R}_V}{\{\mathbf{loop}\ L\ \mathbf{in}\ I_0\ \mathbf{end\ loop}\}\ \mathsf{M} \xrightarrow{\alpha}_s \mathsf{M}''}$$

If I_0 terminates abruptly via statement "**break**", then the breakable loop terminates normally.

$$\frac{\{I_0\}\ \mathsf{M} \xrightarrow{\mathbf{brk}(L)}_s \mathsf{M}'}{\{\mathbf{loop}\ L\ \mathbf{in}\ I_0\ \mathbf{end\ loop}\}\ \mathsf{M} \xrightarrow{\mathbf{exit}}_s \mathsf{M}'}$$

• The syntax of LNT patterns is defined as follows:

$$P \quad ::= \quad x \quad | \quad \mathbf{any} \quad | \quad C(P_1, \ldots, P_l) \quad | \quad F(v_1, \ldots, v_l) \quad | \quad P_0\ \mathbf{where}\ v,$$

where $x \in \mathcal{V}$. The dynamic semantics of patterns is defined either as relation:

$$\{P \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathsf{M}'$$

denoting that pattern P matches value v under memory M updating M to M' or relation:

$$\{P \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathbf{fail}$$

denoting that pattern P fails to match value v under memory M.

By way of structural induction on pattern P, we obtain the following reduction rules:

– The LNT wildcard **any** matches any value v under LNT memory M without affecting it:

$$\frac{}{\{\mathbf{any} \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathsf{M}}$$

– Variable x matches any value expression v under LNT memory M and M is updated to $\mathsf{M} \oplus [\mathsf{x} \leftarrow \mathsf{v}]$:

$$\frac{}{\{x \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathsf{M} \oplus [\mathsf{x} \leftarrow \mathsf{v}]}.$$

– Pattern $C(P_1, \ldots, P_l)$ matches value $C(\mathsf{v_1}, \ldots, \mathsf{v_l})$ under LNT memory $\mathsf{M_1}$ and $\mathsf{M_1}$ is updated to $\mathsf{M_{l+1}}$ if pattern P_k matches value $\mathsf{v_k}$ under LNT memory $\mathsf{M_k}$ and $\mathsf{M_k}$ is updated to $\mathsf{M_{k+1}}$ for all $k \in \{1, \ldots, l\}$.

$$\frac{\forall k \in \{1, \ldots, l\} : \{P_k \sharp \mathsf{v_k}\}\ \mathsf{M_k} \rightarrow_p \mathsf{M_{k+1}}}{\{C(P_1, \ldots, P_l) \sharp C(\mathsf{v_1}, \ldots, \mathsf{v_l})\}\ \mathsf{M_1} \rightarrow_p \mathsf{M_{l+1}}}$$

Note that patterns $P_1, \ldots, P_l$ are evaluated from left to right.

Pattern $C(P_1, \ldots, P_l)$ fails to match value $C(\mathsf{v_1}, \ldots, \mathsf{v_l})$ under LNT memory $\mathsf{M_1}$ if for some $j \in \{1, \ldots, l-1\}$ pattern P_k matches value $\mathsf{v_k}$ under LNT memory $\mathsf{M_k}$ and $\mathsf{M_k}$ is updated to $\mathsf{M_{k+1}}$ for all $k \in \{1, \ldots, j\}$ and pattern P_{j+1} fails to match value v_{j+1} under LNT memory $\mathsf{M_{j+1}}$.

$$\frac{j < l \quad \forall k \in \{1, \ldots, j\} : \{P_k \sharp \mathsf{v_k}\}\ \mathsf{M_k} \rightarrow_p \mathsf{M_{k+1}} \quad \{P_{j+1} \sharp \mathsf{v_{j+1}}\}\ \mathsf{M_{j+1}} \rightarrow_p \textbf{false}}{\{C(P_1, \ldots, P_l) \sharp C(\mathsf{v_1}, \ldots, \mathsf{v_l})\}\ \mathsf{M_1} \rightarrow_p \textbf{false}}$$

Moreover, $C(P_1, \ldots, P_l)$ for $C \in \mathcal{C}$ fails to match $C'(\mathsf{v_1}, \ldots, \mathsf{v_m})$ for $C \in \mathcal{C}$ and any values $\mathsf{v_1}$, …, $\mathsf{v_1}$ such that $C \neq C'$ or the arity of C is not equal to the arity of C', i.e. $l \neq m$:

$$\frac{C \neq C' \vee l \neq m}{\{C(P_1, \ldots, P_l) \sharp C(\mathsf{v_1}, \ldots, \mathsf{v_m})\}\ \mathsf{M} \rightarrow_p \textbf{fail}}$$

– Pattern $F(P_1, \ldots, P_l)$ matches value $\mathsf{F}(\mathsf{v_1}, \ldots, \mathsf{v_l})$ under LNT memory $\mathsf{M_1}$ and $\mathsf{M_1}$ is updated to $\mathsf{M_{l+1}}$ if pattern P_k matches value $\mathsf{v_k}$ under LNT memory $\mathsf{M_k}$ and $\mathsf{M_k}$ is updated to $\mathsf{M_{k+1}}$ for all $k \in \{1, \ldots, l\}$. Notation F stands for the standard interpretation of F as a mathematical function.

$$\frac{\forall k \in \{1, \ldots, l\} : \{P_k \sharp \mathsf{v_k}\}\ \mathsf{M_k} \rightarrow_p \mathsf{M_{k+1}}}{\{F(P_1, \ldots, F_l) \sharp \mathsf{F}(\mathsf{v_1}, \ldots, \mathsf{v_l})\}\ \mathsf{M_1} \rightarrow_p \mathsf{M_{l+1}}}$$

– Pattern "P_0 **where** V" matches value v under LNT memory M and M is updated to $\mathsf{M'}$ if pattern P_0 matches v under M, M is updated to $\mathsf{M'}$ and V evaluates to true under $\mathsf{M'}$:

$$\frac{\{P_0 \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathsf{M'} \quad \{V\}\ \mathsf{M'} \rightarrow_e \textbf{true}}{\{P_0\ \textbf{where}\ V \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathsf{M'}}$$

$$\frac{\{P_0 \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathsf{M'} \quad \{V\}\ \mathsf{M'} \rightarrow_e \textbf{false}}{\{P_0\ \textbf{where}\ V \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \textbf{false}} \qquad \frac{\{P_0 \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \textbf{false}}{\{P_0\ \textbf{where}\ V \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \textbf{false}}$$

- LNT caters for the bidirectional exchange of values between LNT process expressions via the mechanism of offers. The abstract syntax of offers is the following:

$$O ::= \quad !v \quad \mid \quad ?P$$

The dynamic semantics of an offer O is defined as a relation of the form:

$$\{O\}\ \mathsf{M} \rightarrow_o \mathsf{M'},$$

where M is the initial LNT memory related to offer O, and $\mathsf{M'}$ is the updated LNT memory.

– A send offer "$!\,v$" matches a value v under LNT memory M if condition v evaluates to v under M:

$$\frac{\{v\}\ \mathsf{M} \rightarrow_e \mathsf{v}}{\{!\,v \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_o \mathsf{M}}$$

– A receive offer "$?P$" matches a value v under LNT memory M if and only if pattern P matches v under M:

$$\frac{\{P \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_p \mathsf{M}'}{\{?P \sharp \mathsf{v}\}\ \mathsf{M} \rightarrow_o \mathsf{M}'}$$

- The abstract syntax of LNT behaviours is defined as follows:

$$\begin{array}{rcl} B & ::= & \textbf{stop} \mid \textbf{null} \mid G\,(O_1,\ldots,O_n)\ \textbf{where}\ v \mid B_1;B_2 \\ & \mid & \textbf{if}\ v\ \textbf{then}\ B_1\ \textbf{else}\ B_2\ \textbf{end if} \mid \textbf{select}\ B_1\ \texttt{[]}\ \ldots\ \texttt{[]}\ B_n\ \textbf{end select} \\ & \mid & \textbf{loop}\ L\ \textbf{in}\ B\ \textbf{end loop} \mid \textbf{break}\ L \mid \textbf{var}\ x{:}T\ \textbf{in}\ B\ \textbf{end var} \mid I \\ & \mid & \textbf{par}\ G_1,\ldots,G_n\ \textbf{in}\ B_1\ \texttt{||}\ \ldots\ \texttt{||}\ B_n\ \textbf{end par} \mid P\texttt{[}G_1,\ldots,G_n\texttt{]}\ (E_1,\ldots,E_n) \end{array}$$

The dynamic semantics of an LNT behaviour B is defined as a relation of the form:

$$\{B\}\ \mathsf{M} \xrightarrow{\alpha}_b B'\ \{\mathsf{M}'\} \tag{4.3}$$

where α is a label that can take one of the following forms:

– **exit**, whose meaning is similar to the meaning of label **exit** for LNT statements. In relation (4.3), label **exit** denotes normal execution of behaviour B, and behaviour B' is equal to **stop** denoting the deadlock state.

– **brk**(L), whose meaning is similar to the meaning of label **break** for LNT statements, denotes forced termination of statement I via a "**break** L" statement appearing in statement I. Transitions labelled with **brk**(L) occur within the range of breakable loops. As in the case of LNT statements, the set of labels used to mark loops is denoted by $\mathcal{L}$ and the set $\{\textbf{brk}(L) \mid L \in \mathcal{L}\}$ is denoted by $\mathcal{B}_{\mathcal{L}}$.

– **i** or $G(c_1,\ldots,c_n)$ that is called *communication label*, where G is a gate and $c_1,\ldots,c_n \in \mathcal{C}$ are constants. We denote by $\mathcal{C}_v$ the set of *communication labels*. In the following, $gate(\alpha)$ returns the gate of label α:

$$gate\ (\textbf{i}) = \textbf{i} \quad \text{and} \quad gate\ (G(c_1,\ldots,c_n)) = G.$$

M is the LNT memory related to behaviour B, and M' is the LNT memory related to behaviour B'. The meaning is that behaviour B is reduced to behaviour B' under LNT memory M and memory M is updated to memory M' after label α has been executed.

By way of structural induction on behaviour B, we obtain the following reduction rules:

– No reduction rule is related to behaviour **stop**, which denotes inaction.

– Behaviour **null** that is equivalent to λ action terminates normally and the memory remains unchanged:

$$\frac{}{\{\textbf{null}\}\ \mathsf{M} \xrightarrow{\textbf{exit}}_b \{\textbf{stop}\}\ \mathsf{M}}$$

– As in the case of statement "$I_1; I_2$", behaviour "$B_1; B_2$" starts by executing behaviour B_1. As soon as B_1 is executed, the system carries on with the execution of behaviour B_2.

$$\frac{\{B_1\}\ \mathsf{M} \xrightarrow{\mathbf{exit}}_b \{\mathbf{stop}\}\ \mathsf{M}' \quad \{B_2\}\ \mathsf{M}' \xrightarrow{\alpha}_b \{B_2'\}\ \mathsf{M}'' \quad \alpha \in \{\mathbf{exit}\} \cup \mathcal{B}_{\mathcal{L}} \cup \mathcal{C}_v}{\{B_1; B_2\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_2'\}\ \mathsf{M}''}$$

If, by any chance, statement B_1 is terminated abruptly, i.e. via label **break**, then statement "$B_1; B_2$" terminates via label **break** too.

$$\frac{\{B_1\}\ \mathsf{M} \xrightarrow{\mathbf{brk}(L)}_b \{\mathbf{stop}\}\ \mathsf{M}'}{\{B_1; B_2\}\ \mathsf{M} \xrightarrow{\mathbf{brk}(L)}_b \{\mathbf{stop}\}\ \mathsf{M}'}$$

If behaviour B_1 offers a communication label, then behaviour B_1 must continue execution until it terminates:

$$\frac{\{B_1\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_1'\}\ \mathsf{M}' \quad \alpha \in \mathcal{C}_v}{\{B_1; B_2\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_1'; B_2\}\ \mathsf{M}'}$$

– Behaviour "**break** L" terminates and passes label L to the context. As in the case of statement "**break** L", the initial memory remains unchanged after the execution:

$$\frac{}{\{\mathbf{break}\ L\}\ \mathsf{M} \xrightarrow{\mathbf{brk}(L)}_s \{\mathbf{stop}\}\ \mathsf{M}}$$

– Let values $c_1, \ldots, c_n$ and let $\mathsf{c_k}$ denote the standard denotation of c_k with respect to function $\mathcal{F}$ for $k \in \{1, \ldots, n\}$. Behaviour "$G\,(O_1, \ldots, O_n)$ **where** V" executes communication label $G\,(c_1, \ldots, c_n)$ and terminates updating initial LNT memory $\mathsf{M_1}$ to $\mathsf{M_{n+1}}$ if and only if offer O_k matches value $\mathsf{c_k}$ for all $k \in \{1, \ldots, n\}$ updating initial LNT memory $\mathsf{M_k}$ to $\mathsf{M_{k+1}}$ and condition V evaluates to true under $\mathsf{M_{n+1}}$:

$$\frac{\{O_1 \sharp \mathsf{c_1}\}\ \mathsf{M_1} \rightarrow_o \mathsf{M_2} \quad \ldots \quad \{O_n \sharp \mathsf{c_n}\}\ \mathsf{M_n} \rightarrow_o \mathsf{M_{n+1}} \quad \{V\}\ \mathsf{M_{n+1}} \rightarrow_e \mathbf{true}}{\{G\,(O_1, \ldots, O_n)\ \mathbf{where}\ V\}\ \mathsf{M} \xrightarrow{G\,(c_1, \ldots, c_n)}_b \{\mathbf{null}\}\ \mathsf{M_{n+1}}}$$

Note that the offers are evaluated from the left to the right updating the memory after every evaluation.

– Behaviour "**select** B_1 [] ... [] B_n **end select**" behaves as one of the LNT behaviours B_1, ..., and B_n:

$$\frac{i \in \{1, \ldots, n\} \quad \{B_i\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_i'\}\ \mathsf{M}'}{\{\mathbf{select}\ B_1\ []\ \ldots\ []\ B_n\ \mathbf{end\ select}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_i'\}\ \mathsf{M}'}$$

– Behaviour I behaves exactly as statement I:

$$\frac{\alpha \in \{\mathbf{exit}\} \cup \mathcal{B}_L \cup \mathcal{C}_v \quad \{I\}\ \mathsf{M} \xrightarrow{\alpha}_s \{I'\}\ \mathsf{M}'}{\{I\}\ \mathsf{M} \xrightarrow{\alpha}_b \{I'\}\ \mathsf{M}'}$$

– The formal semantics of behaviour "**if** V **then** B_1 **else** B_2 **end if**" is the following:

$$\frac{\{V\} \rightarrow_e \mathsf{true} \quad \{B_1\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_1'\}\ \mathsf{M}'}{\{\textbf{if } V \textbf{ then } B_1 \textbf{ else } B_2 \textbf{ end if}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_1'\}\ \mathsf{M}'}$$

$$\frac{\{V\} \rightarrow_e \mathsf{false} \quad \{B_2\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_2'\}\ \mathsf{M}'}{\{\textbf{if } V \textbf{ then } B_1 \textbf{ else } B_2 \textbf{ end if}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_2'\}\ \mathsf{M}'}$$

– The formal semantics of behaviour "**var** $x{:}T$ **in** B **end var**" is defined as follows:

$$\frac{c \in D_T \quad \mathsf{M}' = \mathsf{M} \oplus [\mathsf{x} \leftarrow \mathsf{c}] \quad \{B\}\ \mathsf{M}' \xrightarrow{\alpha}_b \{B'\}\ \mathsf{M}''}{\{\textbf{var } x{:}T \textbf{ in } B \textbf{ end var}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B'\}\ \mathsf{M}''}$$

Notice that variable x is assigned the standard denotation c (constant) of element c that resides in domain D and its type is T. Hence, the initial LNT memory M is updated to M'. On condition that LNT behaviour B reduces to B' updating LNT memory M' to M'', LNT behaviour "**var** $x{:}T$ **in** B **end var**" reduces to B' updating LNT memory M to M''.

– The dynamic semantics of behaviour LNT "**loop** L **in** B_0 **end loop**" is more complicated than the standard semantics of the corresponding LNT **loop** statement. The reason lies in the eventual occurrence of an unknown number of communications in the body of the **loop** construct before termination. In particular, it is defined by way of the intermediate construct:

$$\textbf{trap } L \textbf{ in } B_1 \textbf{ else } B_2 \textbf{ end trap}.$$

in the following manner:

$$\frac{\{\textbf{trap } L \textbf{ in } B_0 \textbf{ else loop } L \textbf{ in } B_0 \textbf{ end loop end trap}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_0'\}\ \mathsf{M}'}{\{\textbf{loop } L \textbf{ in } B_0 \textbf{ end loop}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_0'\}\ \mathsf{M}'}$$

We proceed with the definition of "**trap** L **in** B_1 **else** B_2 **end trap**". If B_1 offers a communication label α, α is offered by the **trap** construct:

$$\frac{\alpha \in \mathcal{C}_v \quad \{B_1\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_1'\}\ \mathsf{M}'}{\{\textbf{trap } L \textbf{ in } B_1 \textbf{ else } B_2 \textbf{ end trap}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{\textbf{trap } L \textbf{ in } B_1' \textbf{ else } B_2 \textbf{ end trap}\}\ \mathsf{M}'}$$

If behaviour B_1 terminates on a "**break** L" statement, then the **trap** structure terminates normally and the memory remains unchanged:

$$\frac{\{B_1\}\ \mathsf{M} \xrightarrow{\textbf{brk}(L)}_b \{\textbf{stop}\}\ \mathsf{M}}{\{\textbf{trap } L \textbf{ in } B_1 \textbf{ else } B_2 \textbf{ end trap}\}\ \mathsf{M} \xrightarrow{\textbf{exit}}_b \{\textbf{stop}\}\ \mathsf{M}}$$

If behaviour B_1 terminates on a "**break** L'" statement such that $L' \neq L$, then the **trap** construct terminates on "**break** L'" and the memory remains unchanged:

$$\frac{L' \in \mathcal{L} \quad L' \neq L \quad \{B_1\}\ \mathsf{M} \xrightarrow{\textbf{brk}(L')}_b \{\textbf{stop}\}\ \mathsf{M}}{\{\textbf{trap } L \textbf{ in } B_1 \textbf{ else } B_2 \textbf{ end trap}\}\ \mathsf{M} \xrightarrow{\textbf{brk}(L')}_b \{\textbf{stop}\}\ \mathsf{M}}$$

If behaviour B_1 terminates normally (without "**break** L" statement), then the execution of the **trap** construct follows the execution of behaviour B_2 as below:

$$\frac{\alpha \notin \mathcal{B}_L \quad \{B_1\}\ \mathsf{M} \xrightarrow{\mathbf{exit}}_b \{B_1'\}\ \mathsf{M}' \quad \{B_2\}\ \mathsf{M}' \xrightarrow{\alpha}_b \{B_2'\}\ \mathsf{M}''}{\{\mathbf{trap}\ L\ \mathbf{in}\ B_1\ \mathbf{else}\ B_2\ \mathbf{end\ trap}\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_2'\}\ \mathsf{M}}$$

– We consider a simplified (and more suitable to the EB^{3}2LNT translator) version of the LNT parallel composition construct:

$$B \doteq \mathbf{par}\ G_1, \ldots, G_m\ \mathbf{in}\ B_1\ ||\ \ldots\ ||\ B_n\ \mathbf{end\ par}$$

with synchronization on gates $G_1, \ldots, G_m$ and its corresponding formal semantics. We denote by "$\mathsf{M_1} \ominus \mathsf{M_2}$" the *difference* between LNT memories $\mathsf{M_1}$ and $\mathsf{M_2}$ that is defined as follows:

$$(\mathsf{M_1} \ominus \mathsf{M_2})\ (X) = \begin{cases} \mathsf{M_1}(X), & \text{if } \mathsf{M_1}(X) \text{ is defined and } \mathsf{M_2}(X) \text{ is undefined or } \mathsf{M_2}(X) \neq \mathsf{M_1}(X) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Hence, if behaviours $B_1, \ldots, B_n$ offer communication label α such that $gate(\alpha)$ coincides with one of the gates $G_1, \ldots, G_m$, then B offers communication label α, then the following rule is applied:

$$\frac{\alpha \in \mathcal{C}_v \quad gate(\alpha) \in \{G_1, \ldots, G_m\} \quad \forall i \in \{1, \ldots, n\} : \{B_i\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_i'\}\ \mathsf{M_i}}{\{B\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B'\}\ \mathsf{M}\ \oplus\ (\mathsf{M_1} \ominus \mathsf{M})\ \oplus\ \ldots\ \oplus\ (\mathsf{M_n} \ominus \mathsf{M})},$$

where "$B' \doteq \mathbf{par}\ G_1, \ldots, G_m\ \mathbf{in}\ B_1'\ ||\ \ldots\ ||\ B_n'\ \mathbf{end\ par}$". If there exists exactly one (denoted as $!\exists$ below) of the LNT behaviours $B_1, \ldots, B_n$ offering a communication label α indexed by $i \in \{1, \ldots, m\}$ such that $gate(\alpha)$ does not belong to the set of gates $\{G_1, \ldots, G_m\}$, then we may write:

$$\frac{\alpha \in \mathcal{C}_v \quad gate(\alpha) \notin \{G_1, \ldots, G_m\} \quad !\exists i \in \{1, \ldots, m\} : \{B_i\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B_i'\}\ \mathsf{M_i}}{\{B\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B'\}\ \mathsf{M}\ \oplus (\mathsf{M_i} \ominus \mathsf{M})},$$

where "$B' \doteq \mathbf{par}\ G_1, \ldots, G_m\ \mathbf{in}\ B_1'\ ||\ \ldots\ ||\ B_n'\ \mathbf{end\ par}$", and "$B_j' \doteq B_j$" for all $j \neq i$. Last but not least, if parallel behaviours $B_1, \ldots,$ and B_n terminate normally, then the parallel composition terminates normally:

$$\frac{\forall i \in \{1, \ldots, m\} : \{B_i\}\ \mathsf{M} \xrightarrow{\mathbf{exit}}_b \{B_i'\}\ \mathsf{M_i}}{\{B\}\ \mathsf{M} \xrightarrow{\mathbf{exit}}_b \{B'\}\ \mathsf{M}\ \oplus\ (\mathsf{M_1} \ominus \mathsf{M})\ \oplus\ \ldots\ \oplus\ (\mathsf{M_n} \ominus \mathsf{M})},$$

where "$B' \doteq \mathbf{par}\ G_1, \ldots, G_m\ \mathbf{in}\ B_1'\ ||\ \ldots\ ||\ B_n'\ \mathbf{end\ par}$". In the previous rules, the updated LNT memory that is related to the updated LNT behaviour is equal to the initial LNT memory updated accordingly to encompass the local updates in the parallel branches. Furthermore, all variables defined in the LNT memories of the corresponding parallel branches should be disjoint in order to ensure the correctness of the previous transformations. The full version of the aforementioned semantics can be found in [CCG$^+$11].

– Before defining the formal semantics of behaviour $P\,[G'_1,\ldots,G'_n]\ (v_1,\ldots,v_m)$, we first need to define the notion of gate substitution. Let LNT behaviour B. Notation $B\big[\ G'_1/G_1,\ldots,G'_n/G_n\ \big]$ denotes the syntactic substitution of gate G_1 by G'_1, the syntactic substitution of gate G_2 by G'_2 etc. within the body of LNT behaviour B. LNT allows the definition of processes.
We restrict our attention to processes, whose parameters can only be passed *by-value* in the sense that if these parameters are modified within the body of the process callee, the effect is not passed to the process caller after the callee has been executed. Such formal parameters are marked with the keyword "**in**" in the process prototype definition. Note that this subset of LNT user-defined process definitions is sufficient for EB^{3}2LNT.
An LNT process definition can take the following form:

$$\textbf{process}\ P\,[G_1,\ldots,G_n]\ (\textbf{in}\ X_1,\ldots,\textbf{in}\ X_m)\ \textbf{in}\ B\ \textbf{end process},$$

where B is an LNT behaviour and $X_1,\ldots,X_m$ are P's formal parameters. We proceed with the formal semantics of process calls $P\,[G'_1,\ldots,G'_n]\ (v_1,\ldots,v_m)$. First, gate substitution "$\sigma = [G'_1/G_1,\ldots,G'_n/G_n]$" takes place within the body of LNT behaviour B. Then, value $\mathsf{v_i}$ (the interpretation of v_i) is assigned to the corresponding formal parameter X_i of process P for $i \in \{1,\ldots,m\}$. In the following, notation $B(\sigma,\ \mathsf{M_c})$ is an auxiliary structure denoting LNT behaviour B accompanied by gate substitution σ and the LNT memory $\mathsf{M_c}$ the moment that the process call occurs. If $B(\sigma,\ \mathsf{M_c})$ offers a communication label α, so does process call $P\,[G'_1,\ldots,G'_n]\ (v_1,\ldots,v_m)$:

$$\frac{\forall i \in \{1,\ldots,m\} : \{v_i\}\ \mathsf{M} \xrightarrow{\alpha}_e \mathsf{v_i} \quad \{B(\sigma,\ \mathsf{M_c})\}\ \mathsf{M'} \xrightarrow{\alpha}_b \{B''\}\ \mathsf{M''}}{\{P\,[G'_1,\ldots,G'_n]\ (v_1,\ldots,v_m)\}\ \mathsf{M_c} \xrightarrow{\alpha}_b \{B''\}\ \mathsf{M''}}$$

where "$\alpha \in \{\textbf{exit}\} \cup \mathcal{C}_V \cup \mathcal{B}_\mathcal{L}$", "$\mathsf{M'} = [X_1 \leftarrow \mathsf{v_1},\ldots,X_m \leftarrow \mathsf{v_m}]$" and $\mathsf{M_c}$ is the caller's memory the moment that process P is called. If the body of behaviour B offers a communication label α, then the communication label is renamed in keeping with the passing parameters (denoted as $\alpha\sigma$ below) and the execution of $B\ [G'_1/G_1,\ldots,G'_n/G_n]$ terminates normally:

$$\frac{\{B\}\ \mathsf{M} \xrightarrow{\alpha}_b \{B'\}\ \mathsf{M'} \quad \alpha \in \{\textbf{exit}\} \cup \mathcal{C}_V \cup \mathcal{B}_L}{\{B(\sigma,\ \mathsf{M_c})\}\ \mathsf{M} \xrightarrow{\alpha\sigma}_b \{B'(\sigma,\ \mathsf{M_c})\}\ \mathsf{M'}}$$

If behaviour B terminates normally, then the process call terminates normally and memory $\mathsf{M_c}$ is restored:

$$\frac{\{B\}\ \mathsf{M} \xrightarrow{\textbf{exit}}_b \{\textbf{stop}\}\ \mathsf{M'}}{\{B\ (\sigma,\ \mathsf{M_c})\}\ \mathsf{M} \xrightarrow{\textbf{exit}}_b \{\textbf{stop}\}\ \mathsf{M_c}}$$

4.3 Translation from EB^3 to LNT

Principles.

Our translation of EB^3 relies on the memory semantics Sem_{M}. Thus, we explicitly model in LNT a memory, which stores the state variables corresponding to attribute functions (we call these variables *attribute variables*) and is modified each time an action is executed.

Assuming n attribute functions $f_1, \ldots, f_n$, we model the memory as a process M placed in parallel with the rest of the system (a common approach for modeling global variables in process algebras), which manages for each attribute function f_i an attribute variable (also named f_i) that encodes the function. To read the values of these attribute variables (i.e., to evaluate the attribute functions), 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 (including λ, to which we associate a LNT gate also written λ in abstract syntax for convenience), and updates its attribute variables accordingly. The list of attribute variables $\overline{f} = (f_1, \ldots, f_n)$ is added as a supplementary offer on each EB^3 action $\alpha\,(\overline{v})$, so that attribute variables can be directly accessed to evaluate the guard associated to the action, wherever needed, while guaranteeing the *guard-action atomicity*. Therefore, every action $\alpha\,(\overline{v})$ will be encoded in LNT as $\alpha\,(!\,\overline{v}, ?\,\overline{f})$[1], and synchronized with an action of the form $\alpha\,(?\,\overline{x}, !\,\overline{f})$ in the memory process M, thus taking benefit of bidirectional value exchange during the rendezvous.

Translation of attribute functions.

Ordering attribute functions as described in Section 2.2.3 allows the memory to be updated consistently, from f_1 to f_n in turn. At every instant, already-updated values correspond to calls of the form $f_h\,(\mathsf{T}, \ldots)$ (the value of f_h on the current trace), whereas calls of the form $f_h\,(\mathit{front}\,(\mathsf{T}), \ldots)$ are replaced by accesses to a copy $\overline{g}$ of the memory $\overline{f}$, which was made before starting the update. This encoding thus enables the trace parameter to be discharged from function calls, ensuring that while updating f_i, accesses to f_h with $h < i$ necessarily correspond to calls with parameter T.

The process M is defined in Figure 4.1. It runs an infinite loop, which "listens" to all possible actions α_j of the system. At this point, we recall the existence of unique *attribute function* definition $f_i\ (\mathsf{T} : \mathcal{T}, y_1 : T_1, \ldots, y_s : T_s) : T$ (see Figure 2.1 for details). Each attribute variable f_i is an array with s dimensions, where s is the common arity for attribute functions f_i minus 1, because the trace parameter is now discharged. Each dimension of array f_i, thus, corresponds to one formal parameter in $\overline{y_i}$, so that:

$$f_i\,\mathtt{[}\mathbf{ord}\,(v_1)\mathtt{]} \ldots \mathtt{[}\mathbf{ord}\,(v_s)\mathtt{]}$$

encodes the current value of:

$$f_i\,(\mathsf{T}, v_1, \ldots, v_s),$$

where $\mathbf{ord}\,(v_k)$ is a predefined LNT function that denotes the *ordinate* of the value that corresponds to variable v_k, i.e., a unique number between 1 and the cardinal number $|D_k|$ of domain D_k that stocks elements of type T_k.

Expression upd_i^j for $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, q\}$ of Figure 4.1 is used to implement the effect of action $\alpha_j(\overline{x})$ on attribute variables f_i. Its definition is given in Figure 4.2. In particular, upd_i^j is defined by way of auxiliary function *enum*. For vector $\overline{z} = (z_1, \ldots, z_o)$, $y_k :: \overline{z}$ is equivalent notation for $(y_k, z_1, \ldots, z_o)$ and [ ] stands for the empty vector. For each type T_k, we assume the existence of functions first_k that returns the first element of type T, last_k that returns the last element of type T_k, and $\mathit{next}_k\,(x)$ that returns the successor of x on condition that the type of variable x is T_k (following the total order induced by **ord**). For

[1] if $\overline{v} = (v_1, \ldots, v_s)$, then $\alpha\,(!\,\overline{v}, ?\,\overline{f})$ is an abbreviation for $\alpha\ (!\,v_1, \ldots, !\,v_s, ?\,f_1, \ldots, ?\,f_n)$

```
process M [α_1, ..., α_q, λ : any] is
  var f̄, ḡ : T̄,   ȳ : T̄_at,   x̄ : T̄_ac   in
    upd_1^0; ... ; upd_n^0;
    loop
      ḡ := f̄
      select
      α_1 (?x̄, !f̄); upd_1^1; ... ; upd_n^1
      [] ... []
      α_q (?x̄, !f̄); upd_1^q; ... ; upd_n^q
      [] λ (!f̄)
      end select
    end loop
  end var
end process
```

Figure 4.1: LNT code for the memory process implementing the attribute functions

example, let $T_k = \{m_1, m_2, m_3\}$. Then, it is $first_k = m_1$, $next_k(m_1) = m_2$, $next_k(m_2) = m_3$ and $last_k = m_3$. According to the definition of "$enum\,(y_x :: \overline{z}, B)$" of Figure 4.2, the **loop** structure is employed to assign to y_k the current element of domain D_k (the first element of D_k is assigned at first) and a recursive call to $enum\,(\overline{z}, B)$ is taken. If y_k has not been assigned the last value of D_k, which is expressed by condition $y_k \neq last_k$, the next value of D_k is assigned to y_k via $y_k := next_k\,(y_k)$ and $enum\,(\overline{z}, B)$ is called once again. Otherwise, the program breaks from the **loop** structure. This approach guarantees that, when computing the effect of actions on attribute variables via assignment B, all attribute variables are taken into consideration.

Note also that *enum* depends on function *mod* (E) which transforms an expression E by syntactically replacing function calls by array accesses as described previously. Namely according to the definition of $mod\,(w_i^j)$ of Figure 4.2, expressions of the form "$f_h\,(\mathsf{T},\ \overline{y})$" are replaced by $f_h\,[\mathbf{ord}\,(\overline{y})]$ and expressions of the form "$f_h\,(front\,(\mathsf{T}),\ \overline{y})$" are replaced by $g_h\,[\mathbf{ord}\,(\overline{y})]$ for all $h < n$. Recall at this point that the initial values of *attribute variables* $\overline{g}$ are by convection equal to $\perp$, which entails the replacement of *attribute function* calls of the form "$f_h\ (front\,(\mathsf{T}), \overline{y})$" in EB^3 type (1) value expressions w_i^0 by $\perp$ for all $h \in \{1, \ldots, n\}$. Notation $\overline{y} : \overline{T}_{at}$ is an abbreviation for "$y_1 : T_1, \ldots, y_s : T_s$" and $\overline{T}$ denotes the vector $(T, \ldots, T)$ of size n that is equal to the number of *attribute function* names in the system.

Expression $next_k\,(x)$ is implemented as LNT expression "**val** (**ord** $(x) + 1$)". Note that in LNT assignment "$x :=$ **val** (i)" identifier **val** stands for the predefined LNT function that returns the i-th element of given ordered set denoting the domain of x's type. Such functions are available in LNT for all finite types.

Similarly, assuming common parameter vector $\overline{x} \doteq (x_1, \ldots, x_p)$ for all action labels α_j for all $j \in \{1, \ldots, q\}$ or, equivalently, action name definitions of the form $\alpha_j(x_1 : T_1, \ldots, x_p : T_p)$ for all $j \in \{1, \ldots, q\}$, we define "$\overline{x} : \overline{T}_{ac}$" that is syntactic sugar for "$x_1 : T_1, \ldots, x_p : T_p$",

$$
\begin{array}{l}
upd_i^j \doteq enum\,(\overline{y}, f_i\,[\![ord\,(\overline{y})]\!] := mod\,(w_i^j)) \\
enum\,([\,], B) \doteq B \\
enum\,(y_k :: \overline{z}, B) \doteq y_k := first_k; \\
\qquad\qquad\qquad\qquad \textbf{loop } L_k \textbf{ in} \\
\qquad\qquad\qquad\qquad\quad enum\,(\overline{z}, B) \\
\qquad\qquad\qquad\qquad\quad \textbf{if } y_k \neq last_k \textbf{ then } y_k := next_k\,(y_k) \textbf{ else break } L_k \textbf{ end if} \\
\qquad\qquad\qquad\qquad \textbf{end loop} \\
mod\,(w_i^0) \doteq w_i^0\ [\ f_h\,(\mathsf{T},\ \overline{y}) := f_h\,[\![\textbf{ord}\,(\overline{y})]\!] \quad \mid \quad \forall h < n\ ] \\
\qquad\qquad\qquad\quad [\ f_h\,(front\,(\mathsf{T}),\ \overline{y}) := \bot \quad \mid \quad \forall h \in \{1, \ldots, n\}\ ] \\
mod\,(w_i^j) = w_i^j\ [\ f_h\,(\mathsf{T},\ \overline{y}) := f_h\,[\![\textbf{ord}\,(\overline{y})]\!] \quad \mid \quad \forall h < n\ ] \\
\qquad\qquad\qquad\quad [\ f_h\,(front\,(\mathsf{T}),\ \overline{y}) := g_h\,[\![\textbf{ord}\,(\overline{y})]\!] \quad \mid \quad \forall h \in \{1, \ldots, n\}\ ], \quad \text{if } j \neq 0
\end{array}
$$

Figure 4.2: LNT code for expression upd_i^j of Figure 4.1

where types $T_1, \ldots, T_p$ are not to be confused with the types corresponding to the *attribute* parameters $y_k \in \overline{y}$ for $k \in \{1, \ldots, s\}$ as seen earlier. Then, upon synchronization on action $\alpha_j\,(?\overline{x}, !\,\overline{f})$ with the LNT process corresponding to EB^3's *main* process (see translation of processes below), the values of all attribute variables f_i for $i \in \{1, \ldots, n\}$ are updated using function upd_i^j.

As an example, we demonstrate how the definition of Figure 4.1 regarding the memory process, applies to the EB^3 specification of the library management system for one book and two members. Member and book IDs are defined in the LNT program as follows:

type *MID* **is** $m_1, m_2, m_\bot$ **with** "**eq**","**ne**","**ord**","**val**" **end type**
type *BID* **is** $b_1, b_\bot$ **with** "**eq**","**ne**","**ord**","**val**" **end type**

Identifier **eq** (**ne**) denotes the equality (inequality) operator among member IDs or book IDs. The bottom value for member IDs is denoted as $m_\bot$ and the bottom value for book IDs is denoted as $b_\bot$. In the following, the type referring to the number of books possessed by each member of the library is denoted as an array *NB* and the type referring to the current borrower of each book is denoted as an array *BOR*. Hence, *NB* and *BOR* are defined in LNT as follows:

type *NB* **is array**[0..2] **of** *NAT* **end type**
type *BOR* **is array**[0..1] **of** *MID* **end type**

Now, we need to explain how the execution of communication label *LEND* modifies *attribute variable* vectors "*borrower* : *BOR*" and "*nbLoans* : *NB*". First, we remark that, according to the EB^3 specification of the library management system, the execution of EB^3 action "*Lend* (bId, mId)" modifies *attribute variables* $borrower[bId]$ and $nbLoans[mId]$. Hence, following the definition of Figure 4.1, we need an auxilliary variable bId' to go through all possible values of book IDs, i.e. the elements that inhabit *BID*'s domain, in order to simulate the modification of $borrower[bId]$ in case that book bId is lent to member mId. In each iteration of the **loop** construct, it is checked if bId is equal to the current value of bId', in which case $borrower[bId]$ is set to the current borrower, i.e. mId. Then, it is checked if bId'

is equal to the last element of *BID*'s domain, i.e. b_1, in which case the program control breaks from the **loop** construct. Otherwise, bId' is assigned the next element in the ordered set of *BID*'s domain, i.e. "$bId' :=$ **val** (**ord** $(bId') + 1$)", and the execution of the **loop** construct is repeated.

Similarly, we need an auxiliary variable mId' to go through all possible values of member IDs, i.e. the elements that inhabit *MID*'s domain, in order to simulate the modification of $nbLoans[bId]$ in case that book bId is lent to member mId. In each iteration of the **loop** construct, it is checked if mId' coincides with the current value of mId', in which case $nbLoans[mId]$ is increased by one. Then, it is checked if mId' is equal to the last element of *MID*'s domain, i.e. m_2, in which case the program control breaks from the **loop** construct. Otherwise, mId' is assigned the next element in the ordered set of *MID*'s domain, i.e. "$mId' :=$ **val** (**ord** $(mId') + 1$)", and the execution of the **loop** construct is repeated.

The code for the memory M is then given as follows:

```
process M [Acquire, Discard, Register, Unregister, Lend, Return : any] is
  var mId : MID, bId : BID, bId' : BID,
      mId' : MID, borrower : BOR, nbLoans : NB   in
      borrower := BOR(m⊥);   nbLoans := NB(0);
  loop
    select
        Acquire (?bId)
    []  Discard (?bId, ?borrower)
    []  Register (?mId)
    []  Unregister (?mId)
    []  Lend (?bId, ?mId, !nbLoans, !borrower);
        bId' := b1;
        loop L1 in
          if (bId' eq bId) then
            borrower [ord (bId')] := mId
          end if;
          if (bId' eq b1) then
            break L1
          else
            bId' := val (ord (bId') + 1)
          end if;
        end loop
        mId' := m1;
        loop L2 in
          if (mId' eq mId) then
            nbLoans [ord (mId')] := nbLoans [ord (mId')] + 1
          end if;
          if (mId' eq m2) then
            break L2
          else
            mId' := val (ord (mId') + 1)
          end if
        end loop
```

```
        []  RET (?bId);
            mId' := m1;
            loop L1 in
              if (mId' eq borrower [ord (bId)]) then
                nbLoans [ord (mId')] := nbLoans [ord (mId')] - 1
              end if;
              if (mId' eq m2) then
                break L1
              else
                mId' := val (ord (mId') + 1)
              end if
            end loop
            bId' := b1;
            loop L1 in
              if (bId' eq bId) then
                borrower [ord (bId)] := m⊥
              end if;
              if (bId' eq b1) then
                break L1
              else
                bId' := val (ord (bId') + 1)
              end if
            end loop;
      end select
    end loop
    end var
  end process
```

Note that LNT statement "*borrower* := $BOR(m_\perp)$" is syntactic sugar for LNT statement "*borrower*[**ord** (b_1)] := $m_\perp$". Similarly, "*nbLoans* := $NB(0)$" is syntactic sugar for "*nbLoans*[**ord** (m_1)] := 0; *nbLoans*[**ord** (m_2)] := 0". Note that the initial number of loans is set to 0, whereas according to Figure 2.2 it should be set to $\perp$. The reason is that it is impossible to define $Nat_\perp$ of Figure 2.2 without recourse to complex LNT data structures. Hence, the symbol $\perp$ of Figure 2.2 is basically matched to the symbol 0 in the above LNT program, the symbol 0 of Figure 2.2 is matched to 1 in the above LNT program etc.

Optimizations. Notice that the inert action λ has been removed from the previous LNT specification, since λ is does not appear in the program script and, as a result, it is supposed not to affect the control flow.

Similarly, *attribute variable* vector $\overline{f}$ = (*nbLoans*, *borrower*) is removed from the parameter vector of communication labels that synchronize with expressions of the form "$\alpha\,(\overline{v}, ?\overline{f})$ **where** $mod\,(C)$" present in other LNT processes (see Figure 4.3), for which the

corresponding guard $mod\,(C)$ makes no use of *attribute variable* vector $\overline{f}$. In particular, this optimization applies to communication labels *Acquire*, *Register*, *Unregister* and *Return*. The static analysis of LNT specifications cater for an efficient discovery of all guards, as well as their related communication labels.

The only coordinates of *attribute variable* vector $\overline{f} = (nbLoans, borrower)$ passed as parameters to the communication labels "$\alpha\,(\overline{v}, ?\overline{f})$ **where** $mod\,(C)$" are exactly those appearing in $mod\,(C)$. Notice, for example, that vector *nbLoans* is not a subset of the parameter vector passed to communication label *Discard*, since the static analysis of the LNT specification reveals that no corresponding guard to *Discard* makes use of *nbLoans*.

The code referring to *attribute variables* that remain unchanged during a communication is omitted. This is the case for communication labels *Acquire*, *Discard*, *Register* and *Unregister* in the previous LNT code.

It turns out that the code simulating the effect of EB^3 action "*Lend* (bId, mId)" on *attribute variables* $borrower[bId]$ and $nbLoans[mId]$ can be optimized based on the simple observation that *attribute variables* $borrower[bId']$ for $bId' \neq bId$ and $nbLoans[mId']$ for $mId' \neq mId$ remain unaffected. Hence, the corresponding part of process M regarding EB^3 action "*Lend* (bId, mId)" can be modified as follows:

```
[]  Lend (?bId, ?mId, !nbLoans, !borrower);
      borrower [ord (bId)] := mId;
      nbLoans [ord (mId)] := nbLoans [ord (mId)] + 1
```

A similar approach applies to communication label *Ret*. The optimized LNT specification of the library management system is given in the appendix. Note that this optimization technique is applicable whenever the **in** offers of the communication label in question, i.e. the parameters marked with ?, suffice to determine which *attribute variables* are modified. Based on our experience with EB^3 specifications, this optimization technique is often applicable on the corresponding LNT specifications. It is also incorporated on our tool EB^{3}2LNT.

Translation of processes.

We define a translation function t from an EB^3 process expression to an LNT process. Most EB^3 constructs are process algebra constructs with a direct correspondence in LNT. The main difficulty arises in the translation of guarded process expressions of the form "$C \Rightarrow E_0$" in a way that guarantees the *guard-action atomicity*. This led us to consider a second parameter for the translation function t, namely the condition C, whose evaluation is delayed until the first action occurring in the process expression E_0. The definition of $t\,(E, C)$ is given in Figure 4.3. An EB^3 specification E_0 will then be translated into:

$$\textbf{par}\ \alpha_1, \ldots, \alpha_q, \lambda\ \textbf{in}\ t\,(E_0, \text{true})\ \texttt{||}\ M\ \texttt{[}\alpha_1, \ldots, \alpha_q, \lambda\texttt{]}\ \textbf{end par}$$

and every process definition of the form "$P\,(\overline{x}) = E$" will be translated into the process:

$$\textbf{process}\ P\ \texttt{[}\alpha_1, \ldots, \alpha_q, \lambda : \textbf{any}\texttt{]}\ (\overline{x} : \text{type}\,(\overline{x}))\ \textbf{is}\ t\,(E, \text{true})\ \textbf{end process},$$

where $\{\alpha_1, \ldots, \alpha_q\} = Lab$. The Rules of Figure 4.3 can be commented as follows:

- Rule (1) translates the λ action. Note that λ cannot be translated to the empty LNT statement **null**, because execution of λ may depend on a guard C, whose evaluation

$$
\begin{array}{rcll}
t\,(\lambda, C) & = & \lambda\,(?\overline{f})\ \textbf{where}\ mod\,(C) & (1)\\
t\,(\alpha\,(\overline{v}), C) & = & \alpha\,(\overline{v}, ?\overline{f})\ \textbf{where}\ mod\,(C) & (2)\\
t\,(E_1.E_2, C) & = & t\,(E_1, C);\ t\,(E_2, \mathrm{true}) & (3)\\
t\,(C' \Rightarrow E_0, C) & = & t\,(E_0, C\ \textbf{andthen}\ C') & (4)\\
t\,(E_1 \mid E_2, C) & = & \textbf{select}\ t\,(E_1, C)\ \texttt{[]}\ t\,(E_2, C)\ \textbf{end select} & (5)\\
t\,(\mid x : V : E_0, C) & = & \textbf{var}\ x := \textbf{any}\ V;\ t\,(E_0, C)\ \textbf{end var} & (6)\\
t\,(E_0{}^*, \mathrm{true}) & = & \textbf{loop}\ L_{E_0}\ \textbf{in} & \\
 & & \quad \textbf{select} & \\
 & & \quad\quad \lambda\,(?\overline{f});\ \textbf{break}\ L_{E_0}\ \texttt{[]}\ t\,(E_0, \mathrm{true}) & \\
 & & \quad \textbf{end select} & \\
 & & \textbf{end loop} & (7)\\
t\,(E_1\ \texttt{|[}\Delta\texttt{]|}\ E_2, \mathrm{true}) & = & \textbf{par}\ \Delta\ \textbf{in}\ t\,(E_1, \mathrm{true})\ \texttt{||}\ t\,(E_2, \mathrm{true})\ \textbf{end par} & (8)\\
t\,(\texttt{|[}\Delta\texttt{]|}\, x : V : E_0, \mathrm{true}) & = & \textbf{par}\ \Delta\ \textbf{in}\ E_0[x := v_1]\ \texttt{||}\ \ldots\ \texttt{||}\ E_0[x := v_n]\ \textbf{end par} & \\
 & & \quad \text{where } V = \{v_1, \ldots, v_n\} & (9)\\
t\,(P\,(\overline{v}), \mathrm{true}) & = & P\ \texttt{[}\alpha_1, \ldots, \alpha_q, \lambda\texttt{]}\ (\overline{v}) & (10)
\end{array}
$$

In all other cases:

$$
t\,(E_0, C) = \begin{cases}
\textbf{if}\ mod\,(C)\ \textbf{then}\ t\,(E_0, \mathrm{true})\ \textbf{else stop end if} & \\
\quad \text{if } C \text{ does not use attribute functions} & \\
\textbf{par}\ \alpha_1, \ldots, \alpha_q, \lambda\ \textbf{in} & \\
\quad t\,(E_0, \mathrm{true}) & \\
\texttt{||}\ pr_C\ \texttt{[}\alpha_1, \ldots, \alpha_q, \lambda\texttt{]}\ (vars\,(C)) & \\
\textbf{end par} \qquad \text{otherwise} &
\end{cases} \quad (11)
$$

Figure 4.3: Translation from EB^3 process to LNT process

requires the memory to be read, so as to get *attribute variable* values. This is done by the LNT communication action $\lambda\,(?\overline{f})$. The guard C is evaluated after replacing calls to attribute functions (all of which have the form $f_i\,(\mathsf{T}, \overline{v_i})$) by the appropriate *attribute variables*, using function *mod* defined in Figure 4.1. Rule (2) is similar but handles visible actions.

- Rule (3) translates EB^3 sequential composition into LNT sequential composition, passing the evaluation of C to the first process expression.

- Rule (4) makes a conjunction between the guard of the current process expression with the guard already accumulated from the context.

- Rules (5) and (6) translate the choice and quantified choice operators of EB^3 into their direct LNT counterpart.

- Rule (7) translates the Kleene closure into a combination of LNT loop and select, following the identity $E_0{}^* = \lambda \mid E_0.E_0{}^*$.
- Rule (8) translates EB^3 parallel composition into LNT parallel composition.
- Rule (9) translates EB^3 quantified parallel composition into LNT parallel composition by expanding the type V of the quantification variable, since LNT does not have a quantified parallel composition operator.
- Rule (10) translates an EB^3 process call into the corresponding LNT process call, which requires gates to be passed as parameters.
- Rules (7) to (10) only apply when the guard C is trivially true. In the other cases, we must apply Rule (11), which generates code implementing the guard. If C does not use attribute functions, i.e., does not depend on the trace, then it can be evaluated immediately without communicating with the memory process (first case). Otherwise, the guard evaluation must be delayed until the first action of the process expression E_0. When E_0 is either a Kleene closure, a parallel composition, or a process call, identifying its first action syntactically is not obvious. One solution would consist in expanding E_0 into a choice in which every branch has a fixed initial action[2], to which the guard would be added. We preferred an alternative solution that avoids the potential combinatorial explosion of code due to static expansion. A process pr_C (defined in Figure 4.4) is placed in parallel to $t\,(E_0, \text{true})$ and both processes synchronize on all actions. Process pr_C imposes on $t\,(E_0, \text{true})$ the constraint that the first executed action must satisfy the condition C (**then** branch). For subsequent actions, the condition is relaxed (**else** branch).

[2]Such a form, commonly called *head normal form* [BPS01], is used principally in the context of the process algebra ACP [BK85] to analyse the behaviour of recursive processes.

```
process pr_C [α_1, ..., α_q, λ : any] (vars (C) : type (vars (C))) is
var start : bool,   x̄ : T̄_ac,   f̄ : T̄ in
  start := true;
  loop L in select
    if start then
      start := false;
      select
        α_1 (?x̄, ?f̄) where mod (C)
        [] ... []
        α_q (?x̄, ?f̄) where mod (C)
        []
        λ (?f̄) where mod (C)
      end select
    else
      select
        α_1 (?x̄, ?f̄)
        [] ... []
        α_q (?x̄, ?f̄)
        []
        λ (?f̄)
      end select
    end if
  [] break L end select end loop
end var
end process
```

Figure 4.4: Process pr_C

Example Revisited. The **main** process of the LNT specification describing the library management system is defined as follows:

```
process Main [Acquire, Discard, Register, Unregister, Lend, Return : any] is
  par Acquire, Discard, Register, Unregister, Lend, Return in
    par
      loop L in
        select break L [] book [Acquire, Discard] (b1)
        end select
      end loop
    ||
      par
        loop L in
          select break L [] member [Register, Unregister, Lend, Return] (m1)
          end select
        end loop
      ||
        loop L in
          select break L [] member [Register, Unregister, Lend, Return] (m2)
          end select
        end loop
      end par
    end par
  ||
    M [Acquire, Discard, Register, Unregister, Lend, Return]
  end par
end process
```

Note that the equivant LNT process of EB^3 process *main* is placed in parallel to memory process M. Moreover, the simulation of EB^3 quantified parallel synchronization operator (that has no equivalent operator in LNT) is applied to EB^3 expressions "|||$\,bId : BID : book\,(bId)^*$" and "|||$\,mId : MID : member\,(mId)^*$" and the EB^3 Kleene Closure operator is simulated as described in Figure 4.3.

The body of LNT processes *book* and *member* is defined as below:

```
process book [Acquire, Discard : any] (bid : BOOKID) is
  var borrower : BOR in
    Acquire(bid); Discard (bid, ?borrower) where (borrower [ord (bid)] eq m⊥)
  end var
end process
```

process *member* [*Register*, *Unregister*, *Lend*, *Return* : **any**] (*mid* : *MEMBERID*) **is**
  Register (*mid*);
  loop *L* **in**
    select break *L* [] *loan* [*Lend*, *Return*] (*mid*, *b*1)
    end select
  end loop;
  Unregister (*mid*)
end process

The definition of LNT process *member* depends on LNT process *loan*, whose definition is given below:

process *loan* [*Lend*, *Return* : **any**] (*mid* : *MEMBERID*, *bid* : *BOOKID*) **is**
  var *borrower* : *BOR*, *nbLoans* : *NB* **in** (* NbLoans is set to 1 *) **in**
    Lend (*bId*, *mId*, ?*nbLoans*, ?*borrower*)
      where ((*borrower* [**ord** (*bId*)] **eq** $m_\perp$) **and** (*nbLoans* [**ord** (*mId*)] **eq** 1));
    Return (*bId*)
  end var
end process

The following example illustrates and justifies the use of process pr_C as a means to solve the *guard-action atomicity* problem. Consider the EB^3 system:

$$C \Rightarrow \mathit{Lend}\,(b_1,\ m_1)\ |||\ \mathit{Return}\,(b_2),$$

where C denotes the Boolean condition:

$$\mathit{borrower}\,(\mathsf{T},\ b_1) = \perp\ \wedge\ \mathit{nbLoans}\,(\mathsf{T},\ m_1) < \mathit{NbLoans}$$

and $\mathit{Lab} = \{\mathit{Lend}, \mathit{Return}\}$. The LNT code corresponding to this system is the following:

par *Lend*, *Return*, λ **in**
  par *Lend*, *Return*, λ **in**
    par *Lend* (b_1, m_1, ?$\overline{f}$) || *Return* (b_2, ?$\overline{f}$) **end par**
  || pr_C [*Lend*, *Return*, λ] (b_1, m_1)
  end par
|| *M* [*Lend*, *Return*, λ]
end par

The first action executed by this system may be either *Lend* or *Return*. We consider the case where *Lend* is executed first. According to the LNT semantics, it results from the multiway synchronization of the following three actions:

- "*Lend* (b_1, m_1, ?$\overline{f}$)" in the above expression,
- "*Lend* (?b, ?m, ?$\overline{f}$) **where** *borrower*[**ord**(b_1)] $= \perp\ \wedge$ *nbLoans*[**ord**(m_1)] $<$ *NbLoans*" in process pr_C (at this moment, *start* is true, see Fig. 4.4), and

- "*Lend* (?b, ?m, !$\overline{f}$)" in process M (see Fig. 4.1).

Thus, in pr_C at synchronization time, $\overline{f}$ is an up-to-date copy of the memory stored in M, $b = b_1$, and $m = m_1$. The only condition for the synchronization to occur is the guard *mod* (C), whose value is computed using the up-to-date copy $\overline{f}$ of the memory. In case *mod* (C) evaluates to true, no other action (susceptible to modifying $\overline{f}$) can occur between the evaluation of *mod* (C) and the occurrence of *Lend* as both happen synchronously, thus achieving the *guard-action atomicity*. Once *Lend* has occurred, *Return* can occur without any condition, as the value of *start* has now become false.

We developed an automatic translator tool from EB^3 specifications to LNT, named EB^{3}2LNT, implemented using the Ocaml Lex/Yacc compiler construction technology. It consists of about 900 lines of OCaml code. We applied EB^{3}2LNT on a benchmark of EB^3 specifications, which includes variations of the library management system (examined in its simplest version in Chapter 2) and a file transfer system.

We noticed that, for each EB^3 specification, the code size of the equivalent LNT specification is twice as big. Part of this expansion is caused by the fact that LNT is more structured than EB^3: LNT requires more keywords and gates have to be declared and passed as parameters to each process call. By looking at the Rules of Figure 4.3, we can see that the other causes of expansion are Rule (5), which duplicates the condition C, and Rule (9), which duplicates the body E_0 of the quantified parallel composition operator "|[Δ]| $x : V : E_0$" as many times as there are elements in the set V. Both expansions are linear in the size of the source EB^3 code. However, in the case of a nested parallel composition "|[Δ_1]| $x_1 : V_1 : \ldots$ |[Δ_n]| $x_n : V_n : E_0$", the expansion factor is as high as the product of the number of elements in the respective sets $V_1, \ldots, V_n$, which may be large. If E_0 is a big process expression, the expansion can be limited by encapsulating E_0 in a parameterized process "$P_{E_0}\,(x_1, \ldots, x_n)$" and replacing duplicated occurrences of E_0 by appropriate instances of P_{E_0}.

4.4 The Simplified File Transfer System

EB^3 is mainly used for the specification of ISs. However, it is possible to use EB^3 in order to specify file transfer protocols. Let's consider a simplified file transfer system protocol describing the interplay between n clients requesting files that are stocked in m servers. A process interface running in parallel to clients and servers serves an intermediary between them. The exact specification describing the protocol in natural language is given below:

P1. A client C sends a file transfer request "Rqt !F !C" to the server interface regarding file F. Ultimately, if file F is stocked in server S, there is always a procedure that forwards the previous request to server S "Fwd !F !C".

P2. The message should only be sent to one client at a time. In particular, if request message "Fwd !F !C !S" from client C regarding file F reaches server S via the server interface, then file F eventually is sent from server S and reaches client C. Moreover, no pieces of file F can be sent from any other server $S_1 \neq S$ to any other client $C1 \neq C$.

P3. The server interface can set the owner of file F to server S_2 via communication label "*Chown* !F !S_2" provided that F is not being transferred to any client.

P4. The maximum number of file requests is limited to 2.

P5. If file F is being transferred to client C, no other client can ask for F, unless the acknowledgement packet "Ack !F !S !C" has been sent to client C (denoting correct termination of the file transmission).

P6. While a file is being transferred to client C, C can ask for a second file. As a means not to exceed the upper limit on the total numbers of requests, we must make sure that no other requests are made in the meantime.

The EB^3 specification of the simplified file transfer system is given in Figures 4.5 and 4.6:

$SID = \{s_1, \ldots, s_m\}, FID = \{f_1, \ldots, f_p\}, CID = \{c_1, \ldots, c_q\}$
$Rqt\ (f : FID,\ c : CID);$
$Fwd\ (f : FID,\ c : CID,\ s : SID);$
$Chown\ (f : FID,\ s : SID);$
$Trans\ (f : FID,\ s : SID,\ c : CID);$
$Ack\ (f : FID,\ s : SID,\ c : CID);$

$client\ (c : CID) =$
  $|\ (f, s) : (FID, SID) : (numb\_trans\ (\mathsf{T},\ f) = 0) \wedge (files\_down\ (\mathsf{T},\ c) < 2) \Rightarrow Rqt\ (f,\ c)\ |$
    $Trans\ (f,\ s,\ c)\ |\ Ack\ (f,\ s,\ c)$
$clientPR\ (c : CID) = client\ (c)^*$

$interface\ () =$
  $|\ (f, c, s) : (FID, SID, CID) : Rqt\ (f,\ c).\ Fwd\ (f,\ c,\ s)\ |$
  $|\ (f, s) : (FID, CID) : Chown\ (f,\ s)$
$interfacePR\ () = interface\ ()^*$

$server\ (s : SID) =$
  $|\ (f, c) : (FID, CID) :$
    $(owner\ (\mathsf{T},\ f) = s) \Rightarrow Fwd\ (f,\ c)\ |$
    $(owner\ (\mathsf{T},\ f) = s) \wedge (numb\_trans\ (\mathsf{T},\ f) \leq sizef\ (\mathsf{T},\ f) \wedge (dest\ (\mathsf{T},\ f) = s) \Rightarrow$
      $Trans\ (f, s, c)$
    $(owner\ (\mathsf{T},\ f) = s) \wedge (numb\_trans\ (\mathsf{T},\ f) \geq sizef\ (\mathsf{T},\ f) \wedge (dest\ (\mathsf{T},\ f) = s) \Rightarrow$
      $Ack\ (f, s, c)$
$serverPR\ () = server\ ()^*$

$main =$
  $(|||\ c : CID : clientPR\ (c))\ |[Rqt]|\ interfacePR\ ()$
    $|[Fwd, Trans, Ack]|$
  $(|||\ s : SID : serverPR\ (s))$

Figure 4.5: EB^3 specification of the simplified file transfer system (A)

Using EB^{3}2LNT, we translated the EB^3 specification of the library management system to LNT.

num_trans $(\mathsf{T} : \mathcal{T},\ f : FID) : Nat_\perp\ =$ **match** *last* (T) **with** $\perp_{\mathsf{T}}$: $\perp$ | *Rqt* (f, c) : 0 | *Trans* (f, s, c) : *num_trans* (*front* $(\mathsf{T}),\ c) + 1$ | *Ack* (f, s, c) : $\perp$ | _ : *num_trans* (*front* $(\mathsf{T}), f)$ **end match**;	*dest* $(\mathsf{T} : \mathcal{T},\ f : FID) : CID_\perp\ =$ **match** *last* (T) **with** $\perp_{\mathsf{T}}$: $\perp$ | *Fwd* (f, c, s) : c | *Ack* (f, s, c) : $\perp$ | _ : *dest* (*front* $(\mathsf{T}), f)$ **end match**;

files_down $(\mathsf{T} : \mathcal{T},\ c : CID) : Nat_\perp\ =$ **match** *last* (T) **with** $\perp_{\mathsf{T}}$: 0 | *Rqt* (f, c) : *files_down* (*front* $(\mathsf{T}),\ c) + 1$ | _ : *files_down* (*front* $(\mathsf{T}), c)$ **end match**;	*owner* $(\mathsf{T} : \mathcal{T},\ f : FID) : SID_\perp\ =$ **match** *last* (T) **with** $\perp_{\mathsf{T}}$: $(*\ code\ *)$ | *Chown* (f, s) : s | _ : *owner* (*front* $(\mathsf{T}), f)$ **end match**

Figure 4.6: Attribute function definitions describing the simplified file transfer system (B)

4.5 LNT Code for the Simplified File Transfer System

We give the optimized LNT code for the simplified Library File Transfer System for 2 files, 2 clients, and 2 servers:

```
module fts is

type SERVER is s1, s2 with "eq","ne","ord" end type
type CLIENT is c1, c2, c⊥ with "eq","ne","ord" end type
type FILE is s1, s2 with "eq","ne","ord" end type
type OWN is array[0..1] of SERVER end type
type SIZEFILE is array[0..1] of NAT end type

process client [Rqt, Trans, Ack : any] (c : CLIENT) is
  var owner : OWN, num_trans : NUMBTRANS, sizeof : SIZEFILE,
      files_down : FILESDOWN, dest : DESTIN, f : FILE, s : SERVER in
    f := any FILE;
    select
        Rqt (!f, !c, ?numb_trans, ?files_down)
          where ((numb_trans [ord (f)] = 0) and (files_down [ord (c)] < 2))
      [] Trans (!f, ?s, !c, ?owner, ?numb_trans, ?sizeof, ?dest)
      [] Ack (!f, ?s, !c, ?owner, ?numb_trans, ?sizeof, ?dest)
    end select
  end var
end process
```

```
process clientPR [Rqt, Trans, Ack : any] (c : CLIENT) is
  loop L in
    select break L [] client [Rqt, Trans, Ack] (c)
    end select
  end loop;
end process

process interface [Rqt, Fwd, Chown : any] () is
  var owner : OWN, num_trans : NUMBTRANS, files_down : FILESDOWN,
      dest : DESTIN, c : CLIENT, f : FILE, s : SERVER in
    select
          Rqt (?f, ?c, ?numb_trans, ?files_down);
          Fwd (!f, !c, ?s, ?owner)
      []  var f : FILE, s : SERVER in
            f := any FILE; s := any SERVER;
            Chown (!f, !s, ?numb_trans)
              where (numb_trans [ord (f)] = 0)
          end var
    end select
  end var
end process

process interfacePR [Rqt, Fwd, Chown : any] () is
  loop L in
    select break L [] interface [Rqt, Fwd, Chown] ()
    end select
  end loop;
end process
```

```
process server [Fwd, Trans, Ack : any] (s : SERVER) is
  var owner : OWN, num_trans : NUMBTRANS, sizeof : SIZEFILE,
      files_down : FILESDOWN, dest : DESTIN, c : CLIENT, f : FILE in
    select
        Fwd (?f, ?c, !s, !owner)
          where (owner [ord (f)] = s)
      [] Trans (?f, !s, ?c, ?owner, ?numb_trans, ?sizeof, ?dest)
          where ((owner [ord (f)] = s)
                 and (numb_trans [ord (f)] ≤ sizef [ord (f)])
                 and (dest [ord (f)] = c))
      [] Ack (?f, !s, ?c, ?owner, ?numb_trans, ?sizeof, ?dest)
          where ((owner [ord (f)] = s)
                 and (numb_trans [ord (f)] > sizef [ord (f)])
                 and (dest [ord (f)] = c))
    end select
  end var
end process

process serverPR [Rqt, Trans, Ack : any] (s : SERVER) is
  loop L in
    select break L [] server [Fwd, Trans, Ack] (s)
    end select
  end loop;
end process
```

```
process M [Rqt, Fwd, Chown, Trans, Ack : any] is
  var f : FILE, s : SERVER, c : CLIENT,
      numb_trans : NUMBTRANS, sizef : SIZEFILE, owner : OWN,
      files_down : FILESDOWN, dest : DESTIN in
      numb_trans := NUMBTRANS(0); files_down := FILESDOWN(0);
      dest := DESTIN(c⊥); owner := OWN(s1);
      sizef := SIZEFILE(0); owner [ord (f1)] := s1; owner [ord (f2)] := s2;
      sizef [ord (f1)] := 2; sizef [ord (f2)] := 3;
  loop
    select
        Rqt (?f, ?c, !numb_trans, !files_down);
        numb_trans [ord (f)] := 1;
        files_down [ord (c)] := files_down [ord (c)] + 1
    []  Fwd (?f, ?c, ?s, !owner);
        dest [ord (f)] := c
    []  Chown (?f, ?s, !numb_trans);
        owner [ord (f)] := s
    []  Trans (?f, ?s, ?c, !owner, !numb_trans, !sizeof, !dest);
        numb_trans [ord (f)] := numb_trans [ord (f)] + 1
    []  Ack (?f, ?s, ?c, !owner, !numb_trans, !sizeof, !dest);
        numb_trans [ord (f)] := 0;
        dest [ord (f)] := c⊥
    end select
  end loop
  end var
end process
```

```
process Main [Rqt, Fwd, Chown, Trans, Ack : any] () is
  par Rqt, Fwd, Chown, Trans, Ack in
    par Fwd, Trans, Ack in
      par Rqt in
        par
          clientPR [Rqt, Trans, Ack](c1)
        ||
          clientPR [Rqt, Trans, Ack](c2)
        end par
      ||
        interfacePR [Rqt, Fwd, Chown]()
      end par
    ||
      par
        serverPR [Fwd, Trans, Ack](s1)
      ||
        serverPR [Fwd, Trans, Ack](s2)
      end par
    end par
  ||
    M [Rqt, Fwd, Chown, Trans, Ack]
  end par
end process
```

4.6 Proof of equivalence of EB^3 and LNT Specifications

In this section, we attempt a rigorous correctness proof for the equivalence of EB^3 and LNT specifications. The proof is carried out by induction on EB^3 process expressions.

4.6.1 Preliminary Definitions

Let $D_k = \{d_k^1, \ldots, d_k^{|D_k|}\}$ for $k \in \{1, \ldots, s\}$ denote the finite domain that characterizes the k-th *attribute parameter* $y_k \in \overline{y}$ of *attribute function* $f_i\,(\mathsf{T}, \overline{y})$ for $i \in \{1, \ldots, n\}$ whose type is T_k (see 2.1 for details). Note that $|D_k|$ denotes the cardinality of D_k. The LNT memory model we will be using in this section for LNT specifications (LNT memories in shorthand) is an extension of the memory model introduced in the context of EB^3 specifications in Section 2.1.

More specifically, LNT memories may contain:

- any assigned values to *attribute variables* denoted by $\overline{\mathsf{f}}\,[\overline{\mathsf{y}}]$ that correspond to the *attribute variables* $\overline{f}\;[\overline{y}]$ appearing in LNT specifications,
- any assigned values to the *attribute variables* denoted by $\overline{\mathsf{g}}\,[\overline{\mathsf{y}}]$ that correspond to the copy of *attribute variables* $\overline{g}\;[\overline{y}]$ appearing in the body of process M (Figure 4.1),

- any assigned values to variables denoted by y referring to the *attribute parameters* $y \in \overline{y}$,
- any assigned values to vector $\overline{\mathsf{x}}$ corresponding to the parameter vector $\overline{x}$ of actions $\alpha_j(\overline{x})$ for $j \in \{1, \ldots, q\}$. Remark that only one vector can be stocked in memory at a time.

Also, note that a variable appearing in the LNT program as x is represented as x in the corresponding LNT memory. This convention is used widely in the following definitions and proofs.

Definition 4.6.1. ***Equivalence between EB^3 and LNT Memories***
Let M_1 *be an* EB^3 *memory and* M_2 *be an LNT memory. We stipulate that* M_1 *is equivalent to* M_2 *denoted as* $\mathsf{M}_1 \approx \mathsf{M}_2$*, if and only if every attribute variable is assigned the same value both by* M_1 *and* M_2*, or defined formally:*

- *for any attribute vector* $\overline{c}$ *of constants and for any* $i \in \{1, \ldots, n\}$ *such that* $f_i(\mathsf{T}, \overline{y})$ *is a well-formed type (1) value expression, exactly one of the following statements is valid:*
  - $\mathsf{M_1}(\mathsf{f_i})(\overline{\mathsf{c}})$ *and* $\mathsf{M_2}(\mathsf{f_i})(\overline{\mathsf{c}})$ *are not defined, or*
  - $\mathsf{M_1}(\mathsf{f_i})(\overline{\mathsf{c}}) = \mathsf{M_2}(\mathsf{f_i})(\overline{\mathsf{c}})$

Definition 4.6.2. ***Subvectors***
Let $\overline{y} = (y_1, \ldots, y_l)$ *denote an attribute vector. Then, vector* $\overline{y}_{k \to m} = (y_k, \ldots, y_m)$ *for* $k, m \in \{1, \ldots, s\}$ *such that* $k \leq m$ *will be called subvector of* $\overline{y}$*. The obvious extension of subvectors to memories is defined as follows:*

$$\mathsf{M}(\overline{\mathsf{y}}_{k \to m}) = (\mathsf{M}(\mathsf{y}_k), \ldots, \mathsf{M}(\mathsf{y}_m)).$$

4.6.2 Reasoning about the memory

Upon unfolding LNT statement upd_i^j that appears in the body of process M (see Figure 4.1 for details), upd_i^j is transformed as depicted in Figure 2.3.10. Let I_k denote the code block that relates to I_{k+1} in the following manner:

$$I_k \doteq y_k := first_k;\ \textbf{loop}\ L_k\ \textbf{in}\ I_{k+1};\ J_k\ \textbf{end loop}, \tag{4.4}$$

where $0 < k \leq s$ and LNT statement J_k is defined as follows:

$$J_k \doteq \textbf{if}\ y_k \neq last_k\ \textbf{then}\ y_k := next(y_k)\ \textbf{else break}\ L_k\ \textbf{end if}$$

for $0 < k \leq s$. Setting variable k to s, (4.4) is transformed as follows:

$$I_s \doteq y_s := first_s;\ \textbf{loop}\ L_s\ \textbf{in}\ f_i\,[\,\textbf{ord}(\overline{y})\,] := mod(w_i^j);\ J_s\ \textbf{end loop}.$$

Notice that code block I_1 coincides with statement upd_i^j. Let also $\mathsf{R_k}$ be the LNT memory referring to block I_k the moment that I_k is about to be executed, and let $\mathsf{R_k^\sharp}$ be the LNT memory the moment that block I_k has just been executed. Hence, $\mathsf{R_k}$ and $\mathsf{R_k^\sharp}$ are related with the following formula:

$$\{I_k\}\ \mathsf{R_k} \xrightarrow{\textbf{exit}}_s \mathsf{R_k^\sharp}. \tag{4.5}$$

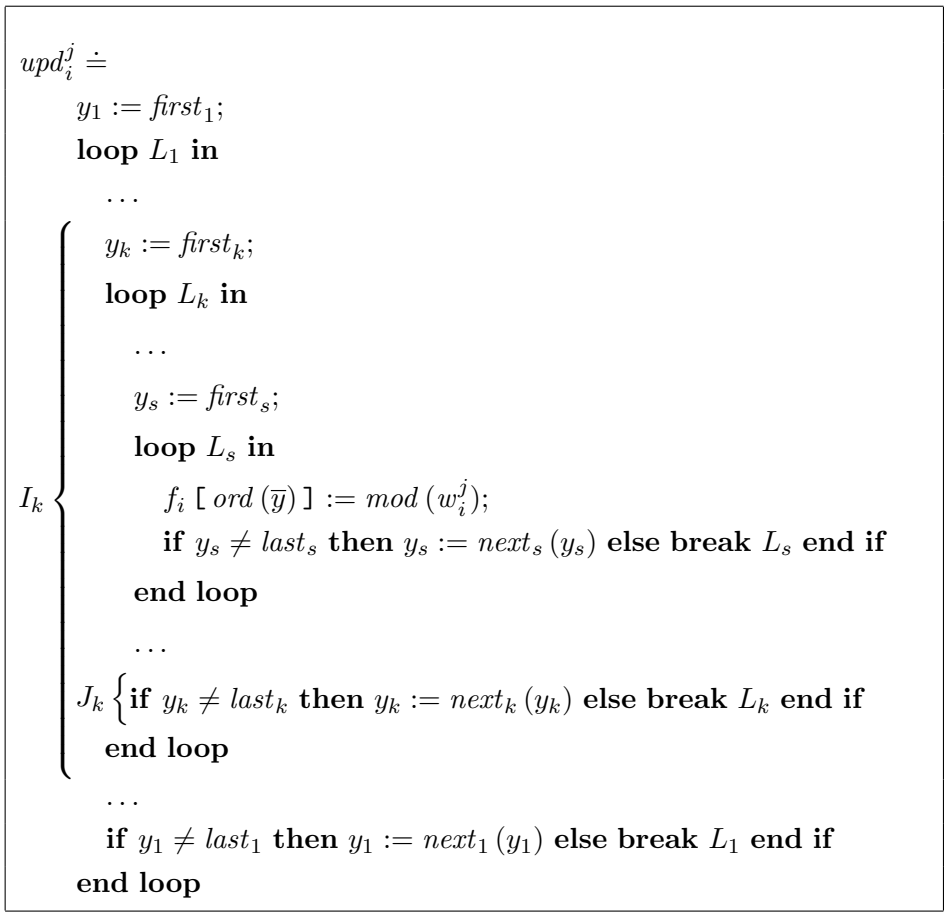


Figure 4.7: Unfolding function upd_i^j

It follows, by simple observation, of Figure 4.7 that LNT statement I_s coincides with the innermost code block of upd_i^j. We proceed with the LNT reduction rules induced by transition "$\{I_s\}\ \mathsf{R_s} \xrightarrow{\textbf{exit}}_s \mathsf{R_s^\sharp}$" and, then, we demonstrate how LNT memory $\mathsf{R_s^\sharp}$ can be calculated. By way of compositional reasoning, executing transition (4.5) triggers the following rule:

$$\frac{\{y_s := first_s\}\ \mathsf{R_s} \xrightarrow{\textbf{exit}}_s \mathsf{R_s}' \quad (4.7)}{\{y_s := first_s;\ \textbf{loop}\ L_s\ \textbf{in}\ f_i\ [\,\textbf{ord}\,(\overline{y})\,]\ := mod\,(w_i^j);\ J_s\ \textbf{end loop}\}\ \mathsf{R_s} \xrightarrow{\textbf{exit}}_s \mathsf{R_s^\sharp}} \quad (4.6)$$

Rule (4.6) assigns to *attribute parameter* y_s the first element of domain D_s, i.e. d_s^1, updates LNT memory $\mathsf{R_s}$ to intermediate LNT memory "$\mathsf{R_s}' = \mathsf{R_s} \oplus [\,\mathsf{y_s} \leftarrow \mathsf{d_s^1}\,]$", and, finally, triggers Rule (4.7):

$$\frac{(4.8)\ \ldots\ (4.12)}{\{\textbf{loop}\ L_s\ \textbf{in}\ f_i\ [\,\textbf{ord}\,(\overline{y})\,]\ := mod\,(w_i^j);\ J_s\ \textbf{end loop}\}\ \mathsf{R_s}' \xrightarrow{\textbf{exit}}_s \mathsf{R_s}^\sharp} \quad (4.7)$$

As for Rule (4.7), the values of *attribute parameters* y_k for $k < s$ carried along by memory $\mathsf{R_s}$ remain unchanged throughout the execution. On the other hand, every time an iteration of

the breakable **loop** construct takes place, a specific *attribute variable* $\mathsf{f_i}\,[\,\mathsf{y_1},\ldots,\mathsf{y_s}\,]$ is modified via the following assignment:

$$f_i\;[\,\mathbf{ord}\,(y_1)\,]\ldots[\,\mathbf{ord}\,(y_s)\,]\;:=\;mod\,(w_i^j)$$

and, subsequently, *attribute parameter* $\mathsf{y_s}$ is assigned a new element from domain D_s. As a result, $|D_s|$ *attribute variables* $\mathsf{f_i}\,[\mathsf{y_1},\ldots,\mathsf{y_s}]$ are modified in total, where $|D_s|$ denotes the cardinality of domain D_s as usual, and LNT memory $\mathsf{R_s}'$ is updated to LNT memory $\mathsf{R_s^\sharp}$. With the exception of Rule (4.12) that denotes "breaking" from the **loop** construct, each reduction rule showing up in the premises of reduction rule (4.7) refers to a specific element of D_s or, equivalently, to a specific *attribute variable* $\mathsf{f_i}\,[\,\mathsf{y_1},\ldots,\mathsf{y_s}\,]$ that is updated. The implied reduction rules denoted by "..." in the premises of Rule (4.7) are similar in effect to Rule (4.8) and the total number of triggered rules depends linearly on $|D_s|$. In particular, Rule (4.8), which is specified as follows:

$$\frac{(4.9)\quad(4.10)}{\{f_i\;[\,ord\,(\overline{y})\,]\;:=\;mod\,(w_i^j);\;J_s\}\;\mathsf{R_s}'\xrightarrow{\mathbf{exit}}_s\mathsf{R_s}''},\tag{4.8}$$

triggers Rule (4.9) and Rule (4.10) that pass the effect of LNT assignments "$f_i\;[\,ord\,(\overline{y})\,]\;:=\;mod\,(w_i^j)$" and "$y_s\;:=\;next_s$" to $\mathsf{R_s}'$, thus, updating LNT memory $\mathsf{R_s}'$ to the intermediate LNT memory $\mathsf{R_s}''$. More concretely, Rule (4.9) is specified as follows:

$$\frac{\{mod\,(w_i^j)\}\;\mathsf{R_s}'\rightarrow_e\mathsf{w_i^j}}{\{f_i\;[\,\mathbf{ord}\,(\overline{y})\,]\;:=\;mod\,(w_i^j)\}\;\mathsf{R_s}'\xrightarrow{\mathbf{exit}}_s\mathsf{R_s}'\oplus\left[\,\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j}\,\right]},\tag{4.9}$$

$$\text{where}\quad f_i\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j}\;\doteq\;f_i\left[\,\mathsf{R_s}'(\mathsf{y_1}),\;\ldots,\;\mathsf{R_s}'(\mathsf{y_s})\,\right]\leftarrow[\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\;(\mathsf{R_s}'),$$

since for $k\in\{1,\ldots,s\}$ *attribute parameter* $\mathsf{y_k}\in\overline{\mathsf{y}}$ is replaced by the corresponding value assigned to y by LNT memory $\mathsf{R_s}'=\mathsf{R_s}\oplus[\,\mathsf{y_s}\leftarrow\mathsf{d_s^1}\,]$.

Moreover, Rule (4.10) as given below:

$$\frac{(4.11)\quad\{y_s:=next_s\,(y_s)\}\;\mathsf{R_s}'\oplus\left[\,\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j}\,\right]\xrightarrow{\mathbf{exit}}_s\mathsf{R_s}''}{\{J_s\}\;\mathsf{R_s}'\oplus\left[\,\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j}\,\right]\xrightarrow{\mathbf{exit}}_s\mathsf{R_s}''},\tag{4.10}$$

reflects the standard semantics of the "**if** *cond* **then** I_1 **else** I_2 **else if**" construct, where *cond* is a Boolean condition that evaluates to true and I_1, I_2 stand for valid LNT statements. In particular, it verifies that the current value of $\mathsf{y_s}$ is not the last in the ordered set D_s as reflected in the following reduction rule:

$$\{y_s\neq last_s\}\;\mathsf{R_s}'\oplus\left[\,\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j}\,\right]\rightarrow_e\mathsf{true}\tag{4.11}$$

and assigns the next value of ordered set D_s to $\mathsf{y_s}$, i.e. d_s^2. LNT memory $\mathsf{N_s}''$ is, then, calculated as follows:

$$\begin{aligned}\mathsf{R_s}''\;&=\;\mathsf{R_s}'\oplus\left[\,\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j}\,\right]\oplus\left[\,\mathsf{y}_s\leftarrow\mathsf{d_s^2}\,\right]\\&=\;\mathsf{R_s}\;\oplus\left[\,\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j},\;\mathsf{y_s}\leftarrow\mathsf{d_s^2}\,\right]\\\text{where}\quad\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,]\leftarrow\mathsf{w_i^j}\;&\doteq\;\mathsf{f_i}\left[\,\mathsf{R_s}(\mathsf{y_1}),\;\ldots,\;\mathsf{R_s}(\mathsf{y_{s-1}}),\;\mathsf{d_s^1}\,\right]\leftarrow[\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\;\left(\mathsf{R_s}\oplus[\,\mathsf{y_s}\leftarrow\mathsf{d_s^1}\,]\right)\end{aligned}$$

since *attribute parameter* $\mathsf{y_k} \in \overline{\mathsf{y}}$ for $k \in \{1, \ldots, s-1\}$ is replaced by the corresponding value assigned to $\mathsf{y_k}$ by LNT memory $\mathsf{R_s}$, i.e. $\mathsf{R_s}'(\mathsf{y_1}) = \mathsf{R_s}(\mathsf{y_1}),\ \ldots,\ \mathsf{R_s}'(\mathsf{y_{s-1}}) = \mathsf{R_s}(\mathsf{y_{s-1}})$, and $\mathsf{y_s}$ is replaced by $\mathsf{d_s^1}$, i.e. $\mathsf{R_s}'(\mathsf{y_s}) = \mathsf{d_s^1}$. Notice that the only *attribute variable* $\mathsf{f_i}\ [\,\overline{\mathsf{y}}\,]$ that is updated is the one, for which the value of *attribute parameter* $\mathsf{y_s}$ is equal to $\mathsf{d_s^1}$.

Moreover, Rule (4.12):

$$\frac{\{y_s \neq last_s\}\ \mathsf{R_s}^\sharp \rightarrow_e \mathsf{false} \quad \{\mathbf{break}\ L_s\}\ \mathsf{R_s}^\sharp \xrightarrow{\mathbf{brk}(L_s)}_s \mathsf{R_s}^\sharp}{\{J_s\}\ \mathsf{R_s}^\sharp \xrightarrow{\mathbf{brk}(L_s)}_s \mathsf{R_s}^\sharp} \tag{4.12}$$

reflects the standard semantics of the "**if** *cond* **then** I_1 **else** I_2 **else if**" construct when *cond* evaluates to false. In particular, Rule (4.12) expresses forced termination (through the **break** construct) after $|D_s|$ consecutive executions of statement "$f_i\ [\,ord\,(\overline{y})\,] := mod\,(w_i^j);\ J_s$" in Rule (4.7) no sooner than y_s is found to be equal to the last element of D_s, i.e. $d_s^{|D_s|}$. Notice that the **break** construct has no effect on the memory, from which follows directly that:

$$\begin{array}{rcl} \mathsf{R_s}^\sharp & = & \mathsf{R_s} \oplus \left[\ \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} \quad \middle|\quad d_s \in D_s\ \right] \oplus \left[\ \mathsf{y_s} \leftarrow \mathsf{d_s^{|D_s|}}\ \right], \\ \text{where} \quad \overline{\mathsf{f}}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} & \doteq & (\mathsf{R_s(y_1)},\ \ldots,\ \mathsf{R_s(y_{s-1})},\ \mathsf{d_s}) \leftarrow [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\ (\mathsf{R_s} \oplus [\ \mathsf{y_s} \leftarrow \mathsf{d_s}\,]) \end{array}$$

since for $k \in \{1, \ldots, s-1\}$ *attribute parameter* $\mathsf{y_k} \in \overline{\mathsf{y}}$ is replaced by the corresponding value assigned to $\mathsf{y_k}$ by LNT memory $\mathsf{R_s}$ and $\mathsf{d_s}$ is replaced by any possible element in domain D_s. Setting $k = s-1$, (4.4) is transformed as follows:

$$I_{s-1} \doteq y_{s-1} := first_{s-1};\ \mathbf{loop}\ L_{s-1}\ \mathbf{in}\ I_s;\ J_{s-1}\ \mathbf{end\ loop} \tag{4.13}$$

Following similar reasoning, it is fairly easy to determine the reduction rules triggered by transition:

$$\{I_{s-1}\}\ \mathsf{R_{s-1}} \xrightarrow{\mathbf{exit}}_s \mathsf{R_{s-1}}^\sharp, \tag{4.14}$$

where $\mathsf{R_{s-1}}$ is the memory referring to the moment prior to I_{s-1}'s execution. Upon execution of statement "$y_{s-1} := first_{s-1}$", LNT memory R_s is set to "$\mathsf{R_{s-1}} \oplus [\,\mathsf{y_{s-1}} \leftarrow \mathsf{d_{s-1}^1}\,]$". The reduction rules induced by transition (4.14) are similar to the rules induced by transition "$\{I_s\}\ \mathsf{R_s} \xrightarrow{\mathbf{exit}}_s \mathsf{R_s^\sharp}$" that were seen previously. As a consequence, we come up easily with the following result:

$$\begin{array}{rcl} \mathsf{R_{s-1}}^\sharp & = & \mathsf{R_{s-1}} \oplus \left[\ \mathsf{f_i}\ [\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} \quad \middle|\quad d_{s-1} \in D_{s-1},\ d_s \in D_s\ \right] \\ & & \qquad \oplus \left[\ \mathsf{y_{s-1}} \leftarrow \mathsf{d_{s-1}^{|D_{s-1}|}},\ \mathsf{y_s} \leftarrow \mathsf{d_s^{|D_s|}}\ \right] \\ \text{where} \quad \overline{\mathsf{f}}\ [\ \overline{\mathsf{d}}\ ] \leftarrow \mathsf{w_i^j} & \doteq & \overline{\mathsf{f}}\ \left[\ \mathsf{R_s(y_1)},\ \ldots,\ \mathsf{R_s(y_{s-2})},\ \mathsf{d_{s-1}},\ \mathsf{d_s}\ \right] \leftarrow \\ & & [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\ \left(\mathsf{R_s} \oplus [\ \mathsf{y_{s-1}} \leftarrow \mathsf{d_{s-1}},\quad \mathsf{y_s} \leftarrow \mathsf{d_s}\,]\right) \end{array}$$

and $\mathsf{d_s}$ is replaced by all possible elements in D_s and $\mathsf{d_{s-1}}$ is replaced by all possible elements in D_{s-1}. Repeating the previous calculations successively for $k = s-2, \ldots, 1$, the relation between LNT memories $\mathsf{R_1}^\sharp$ and $\mathsf{R_1}$ regarding transition:

$$\{I_1\}\ \mathsf{R_1} \xrightarrow{\mathbf{exit}}_s \mathsf{R_1}^\sharp$$

is calculated as follows:

$$\begin{array}{rcl} \mathsf{R_1}^\sharp & = & \mathsf{R_1} \oplus \left[\, \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} \;\;\middle|\;\; d_1 \in D_1,\ \ldots,\ d_s \in D_s \,\right] \\ & & \oplus \left[\, \mathsf{y_1} \leftarrow \mathsf{d_1^{|D_1|}},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{d_s^{|D_s|}} \,\right] \\ \text{where} \quad \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} & \doteq & \mathsf{f_i}\,[\,\mathsf{d_1}, \ldots, \mathsf{d_s}\,] \leftarrow [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\ \left(\mathsf{R_1} \oplus [\,\mathsf{y_1} \leftarrow \mathsf{d_1},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{d_s}\,]\right) \end{array} \tag{4.15}$$

and $\mathsf{d_k}$ is replaced by all possible elements in D_k for $k \in \{1, \ldots, s\}$.

We, then, refer to the body of process M in Figure 4.1. We denote by $\mathsf{N_i^j}$ the memory the moment succeeding the execution of upd_i^j. Notice that $\mathsf{R_1}$ coincides with $\mathsf{N_{i-1}^j}$ in the upper calculations. We denote the initial memory by $\mathsf{N_0^j}$. Applying compositional reasoning, the execution of LNT statement "$upd_1^j;\ \ldots;\ upd_n^j$" results in the following rule:

$$\frac{\{upd_1^0\}\ \mathsf{N_0^j} \xrightarrow{\mathbf{exit}}_s \mathsf{N_1^j},\ \ldots,\ \{upd_n^j\}\ \mathsf{N_{n-1}^j} \xrightarrow{\mathbf{exit}}_s \mathsf{N_n^j}}{\{upd_1^j;\ \ldots;\ upd_n^j\}\ \mathsf{N_0^j} \xrightarrow{\mathbf{exit}}_s \mathsf{N_n^j}} \tag{4.16}$$

and, by (4.15), the relation between memories $\mathsf{N_i^j}$ and $\mathsf{N_i^{j-1}}$ is calculated as follows:

$$\begin{array}{rcl} \mathsf{N_i^j} & = & \mathsf{N_{i-1}^j} \oplus \left[\, \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} \;\;\middle|\;\; d_1 \in D_1,\ \ldots,\ d_s \in D_s \,\right] \\ \text{where} \quad \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} & \doteq & \mathsf{f_i}\,[\,\mathsf{d_1}, \ldots, \mathsf{d_s}\,] \leftarrow [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\ \left(\mathsf{N_{i-1}^j} \oplus [\,\mathsf{y_1} \leftarrow \mathsf{d_1},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{d_s}\,]\right) \end{array} \tag{4.17}$$

for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, q\}$. Notice that the modified values of *attribute parameters* $\mathsf{y_k} \in \overline{\mathsf{y}}$ (compare with (4.15)) have been discharged from $\mathsf{N_i^j}$ for brevity, as it is of no particular use for the ensuing calculations. From now on, the assigned values to $\mathsf{y_k}$ will be indicated explicitly wherever needed. Hence, by way of (4.17), LNT memory $\mathsf{N_n^j}$ can be calculated as shown below:

$$\begin{array}{rcl} \mathsf{N_n^j} & = & \mathsf{N_0^j} \oplus \left[\, \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} \;\;\middle|\;\; d_1 \in D_1,\ \ldots,\ d_s \in D_s \,\right] \\ \text{where} \quad \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} & \doteq & \mathsf{f_i}\,[\,\mathsf{d_1}, \ldots, \mathsf{d_s}\,] \leftarrow [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\ \left(\mathsf{N_{i-1}^j} \oplus [\,\mathsf{y_1} \leftarrow \mathsf{d_1},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{d_s}\,]\right) \end{array} \tag{4.18}$$

for $j \in \{1, \ldots, q\}$. In order to calculate the LNT memory at system start, i.e. $\mathsf{N_n^0}$, we need to set $j = 0$ in (4.18). Notice that LNT memory $\mathsf{N_0^0}$ relating to process M prior to the execution of behaviour upd_1^0 is $[\ ]$. Hence, LNT memory $\mathsf{N_n^0}$ can be calculated as follows:

$$\begin{array}{rcl} \mathsf{N_n^0} & = & \left[\, \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^j} \;\;\middle|\;\; d_1 \in D_1,\ \ldots,\ d_s \in D_s \,\right] \\ \text{where} \quad \mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow \mathsf{w_i^0} & \doteq & \mathsf{f_i}\,[\,\mathsf{d_1}, \ldots, \mathsf{d_s}\,] \leftarrow [\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}\ \left(\mathsf{N_{i-1}^0} \oplus [\,\mathsf{y_1} \leftarrow \mathsf{d_1},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{d_s}\,]\right) \end{array} \tag{4.19}$$

As a means to simplify the presentation, we use the following definitions regarding process M of Figure 4.1:

$$\begin{array}{rcl} D & \doteq & \mathbf{select}\ \alpha_1\,(?\overline{x},\ !\,\overline{f});\ upd_1^1;\ \ldots;\ upd_n^1\ [\,]\ \ldots\ [\,] \\ & & \qquad \alpha_q\,(?\overline{x},\ !\,\overline{f});\ upd_1^q;\ \ldots;\ upd_n^q\ [\,]\ \lambda\,(!\,\overline{f}) \\ & & \mathbf{end\ select} \end{array} \tag{4.20}$$

$$lp \doteq \mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop} \tag{4.21}$$

Now, let $C \Rightarrow E$ be a *well-formed* EB^3 process expression. We denote that:

$$\begin{aligned} B\,(E,C) \;\doteq\; & \textbf{par } \alpha_1,\ldots,\alpha_q,\lambda \textbf{ in } t\,(E,C) \textbf{ ||} \\ & \qquad \textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop} \\ & \textbf{end par} \end{aligned} \tag{4.22}$$

Notice that the execution of LNT statement "$upd_1^0;\ \ldots;\ upd_n^0$" in the body of process M is orthogonal to the execution of behaviour $t\,(E, true)$ in the sense that the execution of "$upd_1^0;\ \ldots;\ upd_n^0$" precedes all synchronisations on actions among LNT behaviours $t\,(E, true)$ and M. This observation can be justified by the only possible LNT rule reduction scenario (see the semantics of LNT parallel composition):

$$\frac{(4.24)}{\{\textbf{par } \alpha_1,\ldots,\alpha_q,\lambda \textbf{ in } t\,(E,\text{true}) \textbf{ || } M\ \texttt{[}\alpha_1,\ldots,\alpha_q,\lambda\texttt{]} \textbf{ end par}\}\ [\,]\ \xrightarrow{\textbf{exit}}_b\ \{B\,(E,true)\}\ \mathsf{N_n^0}} \tag{4.23}$$

that is applicable to LNT behaviour:

$$\textbf{par } \alpha_1,\ldots,\alpha_q,\lambda \textbf{ in } t\,(E,\text{true}) \textbf{ || } M\ \texttt{[}\alpha_1,\ldots,\alpha_q,\lambda\texttt{]} \textbf{ end par}$$

corresponding to EB^3 expression E. Rule (4.23) denotes partial execution of LNT behaviour "$M\ \texttt{[}\alpha_1,\ldots,\alpha_q,\lambda\texttt{]}$", which constitutes the right construct of $t\,(E, true)$ and triggers the following reduction rule:

$$\frac{(4.16)}{\{M\ \texttt{[}\alpha_1,\ldots,\alpha_q,\lambda\texttt{]}\}\ [\,]\ \xrightarrow{\textbf{exit}}_b\ \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N_n^0}}, \tag{4.24}$$

that passes the effect of LNT statement "$upd_1^0;\ \ldots;\ upd_n^0$" on the initial LNT memory $\mathsf{N_0^0} = [\,]$ by assigning to the *attribute variables* their initial values. By observation of Rule (4.16), it follows (for $j = 0$) that the updated memory of Rule (4.24) is equal to $\mathsf{N_n^0}$.

Theorem 4.6.1. ***Equivalence of*** $\mathsf{N_n^0}$ ***and*** $\mathsf{M_0}$
Let EB^3 be well-formed EB^3 *specification given as process definition "main = E".* EB^3 *memory* $\mathsf{M_0}$ *characterizing EB^3 at system start is equivalent to LNT memory* $\mathsf{N_n^0}$ *characterizing process M in the corresponding LNT specification the exact moment succeeding the execution of LNT behaviour "$upd_1^0;\ldots;upd_n^0$", i.e.* $\mathsf{N_n^0} \approx \mathsf{M_0}$.

Proof. Let vector of constants $\overline{d} = (d_1,\ldots,d_s)$ such that $d_1 \in D_1,\ldots,d_2 \in D_s$. Let the *trace* variable T be equal to empty *trace*, i.e. "$\mathsf{T} = [\,]$". We need to prove that:

$$\mathsf{N_n^0(f_i)(d_1,\ldots,d_s)} = \mathsf{M_0(f_i)(d_1,\ldots,d_s)} \tag{4.25}$$

for all $i \in \{1,\ldots,n\}$. Combining (4.19) and Figure 4.1, it follows that:

$$\begin{aligned} \mathsf{N_n^0(f_i)[d_1,\ldots,d_s]} &= [\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}^0} \oplus [\mathsf{y_1} \leftarrow \mathsf{d_1},\ \ldots,\ \mathsf{y_s} \leftarrow \mathsf{d_s}]) \\ \text{where}\quad mod\,(w_i^0) &= w_i^0\ [f_h\,(\mathsf{T},\overline{y}) := f_h\,\texttt{[}ord\,(\overline{y})\texttt{]}, f_h\,(front\,(\mathsf{T}),\overline{y}) := \bot] \end{aligned}$$

Furthermore, by way of Definition 2.3.9, it is:

$$\mathsf{M_0(f_i)(d_1,\ldots,d_s)} = [\![w_i^0]\!]_1^{\mathsf{M}}([\ \mathsf{y_1} \leftarrow \mathsf{d_1},\ldots,\mathsf{y_s} \leftarrow \mathsf{d_s}], \mathsf{M_0}).$$

Hence, in order to prove that $\mathsf{N}^0_{\mathsf{n}} \approx \mathsf{M}_0$, it suffices to prove that:

$$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0), \tag{4.26}$$

where "$\tau = [\mathsf{y}_1 \leftarrow \mathsf{d}_1, \ldots, \mathsf{y}_s \leftarrow \mathsf{d}_1]$". We proceed with structural induction on EB^3 type (1) value expressions w_i^0. Then, applying the dynamic semantics of type (1) LNT value expressions in Section 4.2.1, we obtain the following results:

- w_i^0 reduces to constant $c \in \mathcal{C}$. Then, it is "$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![mod\,(c)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = \mathsf{c}$". By application of Definition 2.3.8, we find that "$[\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0) = [\![c]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0) = \mathsf{c}$". Hence, it is derived that "$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0)$" that completes the proof.

- w_i^0 reduces to variable $x \in \mathcal{V}$. Then, it is "$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![mod\,(x)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = \tau(x)$", since variable x can only be a formal parameter of action label α_j for some $j \in \{1, \ldots, q\}$. By application of Definition 2.3.8, we find that "$[\![x]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0) = \tau(x)$". Hence, it is derived that "$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0)$" that completes the proof.

- w_i^0 reduces to $g(v_1, \ldots, v_l)$. We recall that function identifier g is interpreted by function $\mathcal{F}$ as is the case for all function identifiers. This allows us to write the following equation:

$$\begin{aligned} [\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) &= [\![mod\ (g(v_1, \ldots, v_l))]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) \\ &= \mathsf{g}\big([\![mod(v_1)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau), \ldots, [\![mod(v_l)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau)\big). \end{aligned}$$

  From the induction hypothesis, it is derived that:

$$[\![mod(v_1)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![v_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0),\ \ldots,\ [\![mod(v_l)\ ]\!]_{\mathsf{LNT}}\ (\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![v_l]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0).$$

  Then, by application of Definition 2.3.8, we find that:

$$[\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0) = \mathsf{g}([\![v_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0), \ldots, [\![v_l]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0))$$

  and, combining the previous results, it can be easily derived that:

$$[\![g(v_1, \ldots, v_l)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0)$$

  or, simply, "$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0)$" that completes the proof.

- w_i^0 reduces to *attribute function* call $f_h\,(front\,(\mathsf{T}), \overline{u})$ for $h \in \{1, \ldots, n\}$. Then, we may write that:

$$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![mod\,(f_h\,(front\,(\mathsf{T}), \overline{u}))]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![\bot]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = \bot.$$

  By application of Definition 2.3.8, it is deduced that:

$$[\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0) = [\![f_h\,(front\,(\mathsf{T}),\ \overline{u})]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}_0) = [\![\bot]\!]_1^{\mathsf{M}}\ (\tau,\ \mathsf{M}_0) = \bot,$$

  which completes the proof.

- w_i^0 reduces to *attribute function* call $f_h\,(\mathsf{T},\ \overline{u})$ for $h < i$, where $\overline{u} \doteq (u_1, \ldots, u_s)$ is a vector of valid EB^3 expressions. Applying the dynamic semantics of Section 4.2.1, it is deduced that:

$$\begin{aligned}
[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}\ (\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) &= [\![mod\ (f_h\,(\mathsf{T}, \overline{u}))]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) \\
&= [\![f_h\,[\![\overline{u}]\!]]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) \quad \text{(Definition of } mod\text{)} \\
&= [\![mod\ (w_h^j)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{h-1}} \oplus \tau \oplus \tau') \quad (4.19) \\
&= [\![w_h^j]\!]_1^{\mathsf{M}}(\tau \oplus \tau', \mathsf{M_0}) \\
&= [\![\mathsf{f_h}(\mathsf{T}, u_1, \ldots, u_s)]\!]_1^{\mathsf{M}}(\tau,\ \mathsf{M_0}) \quad \text{(Definition 2.3.8)},
\end{aligned}$$

where "$\tau' \doteq [\mathsf{y_1} \leftarrow [\![u_1]\!]_{\mathsf{LNT}}\ (\mathsf{N}^0_{\mathsf{h-1}} \oplus \tau), \ldots, \mathsf{y_s} \leftarrow [\![u_s]\!]_{\mathsf{LNT}}\ (\mathsf{N}^0_{\mathsf{h-1}} \oplus \tau)]$". Remark that the following proposition:

$$[\![mod\ (w_h^j)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{h-1}} \oplus \tau \oplus \tau') = [\![w_h^j]\!]_1^{\mathsf{M}}(\tau \oplus \tau', \mathsf{M_0}) \tag{4.27}$$

is obtained by substituting index i with h for $h < i$ and substituting environment τ with $\tau \oplus \tau'$ in (4.26). The rest of the proof follows the lines of the corresponding proof for Theorem 2.4.1. Repeating the previous calculations and applying inductive reasoning, the correctness of (4.27) boils down to the correctness of the following proposition:

$$[\![mod\ (w_1^j)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{0}} \oplus \tau \oplus \tau') = [\![w_1^j]\!]_1^{\mathsf{M}}(\tau \oplus \tau', \mathsf{M_0}) \tag{4.28}$$

However, by way of the attribute function ordering, w_1^j cannot reduce further to any *attribute function* call $f_k\ (\mathsf{T}, \ldots)$ for $k > 0$, which means that the proof steps taken so far to prove (4.27) suffice to complete the proof for (4.28). Hence, it follows that:

$$[\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![w_i^0]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}).$$

- w_i^j reduces to "**if** $g(v_1, \ldots, v_l)$ **then** v_{l+1} **else** v_{l+2} **end if**".

  1. Let "$[\![mod(g(v_1, \ldots, v_l))]\!]_{\mathsf{LNT}}\ (\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = \mathsf{true}$". The inductive principle applies to the "syntactically smaller" value expression $g(v_1, \ldots, v_l)$, which allows us to write:

  $$[\![mod(g(v_1, \ldots, v_l))]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = [\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}),$$

  By applying the dynamic semantics of Section 4.2.1, it is found that:

  $$\begin{gathered}
  [\![mod\ (g(v_1, \ldots, v_l))]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau) = \\
  \mathsf{g}([\![mod(v_1)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau), \ldots, [\![mod(v_l)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau)).
  \end{gathered}$$

  From the induction hypothesis (and with compositional reasoning), it follows that:

  $$\begin{gathered}
  \mathsf{g}([\![mod(v_1)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau), \ldots, [\![mod(v_l)]\!]_{\mathsf{LNT}}(\mathsf{N}^0_{\mathsf{i-1}} \oplus \tau)) = \\
  \mathsf{g}([\![v_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}), \ldots, [\![v_l]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}))
  \end{gathered}$$

  Then, by applying Definition 2.3.8, we find that:

  $$[\![g(v_1, \ldots, v_l)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}) = \mathsf{g}([\![mod(v_1)]\!]_1^{\mathsf{M}}(\tau, \mathsf{M_0}), \ldots, [\![mod(v_l)]\!]_{\mathsf{M}}(\tau, \mathsf{M_0}))$$

and, combining the previous results, it is easily derived that "$[\![g(v_1,\ldots,v_l)]\!]_{\mathsf{M}}(\tau,\mathsf{M_0}) = \mathsf{true}$". Then, $[\![w_i^0]\!]_{\mathsf{LNT}}(\mathsf{N^0_{i-1}} \oplus \tau)$ can be transformed as follows:

$$\begin{aligned}
[\![mod(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N^0_{i-1}} \oplus \tau) &= [\![mod(\textbf{if } g(v_1,\ldots,v_l) \textbf{ then } v_{l+1} \textbf{ else } v_{l+2})]\!]_{\mathsf{LNT}}(\mathsf{N^0_{i-1}} \oplus \tau) \\
&= [\![mod(v_{l+1})]\!]_{\mathsf{LNT}}(\mathsf{N^0_{i-1}} \oplus \tau) \quad \text{(Definition 2.3.5)} \\
&= [\![v_{l+1}]\!]_1^{\mathsf{M}}(\tau,\mathsf{M_0}) \quad \text{(Induction Hypothesis)} \\
&= [\![\textbf{if } g(v_1,\ldots,v_l) \textbf{ then } v_{l+1} \textbf{ else } v_{l+2}]\!]_1^{\mathsf{M}}(\tau,\mathsf{M_0}) \\
&\quad \text{(Definition 2.3.8)} \\
&= [\![w_i^0]\!]_1^{\mathsf{M}}(\tau,\mathsf{M_0})
\end{aligned}$$

Notice also how the inductive hypothesis applies previously on v_{l+1}:

$$[\![mod(v_{l+1})]\!]_{\mathsf{LNT}}(\mathsf{N^0_{i-1}} \oplus \tau) = [\![v_{l+1}]\!]_1^{\mathsf{M}}(\tau,\mathsf{M_0})$$

and, combining the previous results, it is easily found that:

$$[\![mod\ (\textbf{if } g(v_1,\ldots,v_l) \textbf{ then } v_{l+1} \textbf{ else } v_{l+2} \textbf{ end if})]\!]_{\mathsf{LNT}}(\mathsf{N^0_{i-1}} \oplus \tau) = [\![\textbf{if } g(v_1,\ldots,v_l) \textbf{ then } v_{l+1} \textbf{ else } v_{l+2} \textbf{ end if}]\!]_1^{\mathsf{M}}(\tau,\mathsf{M_0})$$

or, equivalently, "$[\![mod\ (w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N^0_{i-1}} \oplus \tau) = [\![w_i^0]\!]_1^{\mathsf{M}}(\tau,\mathsf{M_0})$".

2. The symmetric case is similar.

□

The framework introduced so far allows us to proceed with the bisimulation equivalence theorem regarding EB^3 specifications and their corresponding LNT specifications.

4.6.3 LTS Construction for EB^3 and LNT Specifications

- Let EB^3 be a *well-formed* EB^3 specification including process prototype definition "$main = E_0$". Let also:

$$TS_{\text{EB}^3} = (S_1, \rightsquigarrow_1 \doteq \{\xrightarrow{\rho_1}_1\}_{\rho_1 \in Act_1}, s_1^0)$$

be the LTS that corresponds to EB^3, where $Act_1 \doteq \{\alpha_j\,(\overline{c}) \mid j \in 1..q\} \cup \{\lambda\}$ is a set of actions, $\rho_1 \in Act_1$ and $\overline{c}$ is a vector of constants. By virtue of Sem_M, statespace S_1 is represented by tuples of the form:

$$(E, \mathsf{M}),$$

where E is a derivation of E_0 and M is the current EB^3 memory namely the assigned values to attribute variables. Transitions of the form:

$$(E, \mathsf{M}) \rightsquigarrow_1 (E', \mathsf{M'}),$$

follow the structured EB^3 SOS Rules of Figure 2.9, and $\mathsf{M'}$ is specified as in Definition 2.3.10. The initial state s_1^0 is denoted by:

$$(E_0, \mathsf{M_0}).$$

More details on the construction of TS_{EB^3} can be found in Section 2.4.2.

- Let the LTS that corresponds to the LNT specification produced by EB^{3}2LNT with input the initial EB^3 specification EB^3:

$$TS_{LNT} = (S_2, \rightsquigarrow_2 \doteq \xrightarrow{1} \circ \xrightarrow{2} \circ \xrightarrow{3}, s_2^0),$$

where statespace S_2 is represented by tuples of the form:

$$(B\,(E), \mathsf{N}),$$

and $B\,(E)$ is the LNT expression corresponding to the EB^3 process expression E, and $\mathsf{N}^{\mathsf{j}}_{\mathsf{n}}$ is the current LNT memory. As opposed to EB^3, where executions of actions and memory updates are carried out synchronously, in LNT they can only be carried out sequentially. This remark is reflected in relation $\rightsquigarrow_2$ that constitutes a composition of three relations $\xrightarrow{1}$, $\xrightarrow{2}$ and $\xrightarrow{3}$.

Let transition $(B\,(E), \mathsf{N}) \rightsquigarrow_2 (B\,(E'), \mathsf{N}')$. Relation $\xrightarrow{1}$ corresponds to executions of LNT statement "$\overline{g} := \overline{f}$":

$$\{B\,(E)\}\ \mathsf{N} \xrightarrow{\textbf{exit}}_b \{B_1\}\ \mathsf{N_1},$$

where "$\mathsf{N_1} \doteq \mathsf{N} \oplus \left[\ \overline{\mathsf{g}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}\ \right]$" and B' stands for an intermediate LNT behaviour and $\overline{\mathsf{f}}_{\mathsf{N}}$ denotes the value assigned to *attribute vector* $\overline{\mathsf{f}}$ by LNT memory N, i.e. $\mathsf{N}(\overline{\mathsf{f}}) = \overline{\mathsf{f}}_{\mathsf{N}}$. Given the set of communication labels "$Act_2 \doteq \{\alpha_j\ (\overline{c},\ \overline{f}_N) \mid j = 1..q\} \cup \{\lambda(\overline{f}_N)\}$", relation $\xrightarrow{2}$ corresponds to executions of the form $(\xrightarrow{\alpha}_b)^{0..1} \circ \xrightarrow{\rho_2}_b \circ (\xrightarrow{\alpha}_b)^{0..1}$, where $(\xrightarrow{\alpha}_b)^{0..1}$ denotes zero or one execution of transition $\xrightarrow{\alpha}_b \in \mathcal{B}_L \cup \{\xrightarrow{\textbf{exit}}_b\}$ that is restricted to the left counterpart $t(E, C)$ of $B(E, C)$ (more details on the exact nature of transitions $\xrightarrow{\alpha}_b$ can be found in the proof section of Theorem 4.6.4) and communication label $\rho_2 \in Act_2$ such that:

$$\{B_1\}\ \mathsf{N_2} \xrightarrow{\rho_2}_b \{B_2\}\ \mathsf{N_3}, \tag{4.29}$$

where $\mathsf{N_2}$ is an LNT memory for which $\mathsf{N_2}(\overline{\mathsf{g}}) = \overline{\mathsf{f}}_{\mathsf{N}}$, $\mathsf{N_3}$ is an LNT memory for which $\mathsf{N_3}(\overline{\mathsf{x}}) = \overline{\mathsf{c}}$, and $\mathsf{N_3}(\overline{\mathsf{g}}) = \overline{\mathsf{f}}_{\mathsf{N}}$ and $\overline{x}$ denotes the formal parameter vector of label α_j that is assigned vector $\overline{c}$ (obviously, the previously assigned value to vector $\overline{x}$ is overwritten), B'' stands for an intermediate LNT behaviour, and relation $\xrightarrow{3}$ corresponds to evaluations of LNT statement "$upd_1^j; \ldots; upd_n^j$" for $j \in 1..q$:

$$\{B_2\}\ \mathsf{N_3} \xrightarrow{\textbf{exit}}_b \{B\,(E')\}\ \mathsf{N}',$$

where LNT memory N' is calculated by replacing N with $\mathsf{N_3}$ as shown in (4.18). The initial state s_2^0 of TS_{EB^3} is denoted by:

$$(B\,(E_0), \mathsf{N}^{\mathsf{0}}_{\mathsf{n}}).$$

The LNT reduction rules stipulate which communication labels can be executed. We recall that LNT and EB^3 share common process algebra operators. Hence, the construction of TS_{LNT} can be based on the compositional techniques over LTS constructions that have already been developed in Chapter 2. The interested reader may refer to the corresponding sections.

Theorem 4.6.2. ***Equivalence of EB^3 and LNT memories*** *Let EB^3 be a well-formed* EB^3 *specification including process prototype definition "main = E_0", Act_1 be a set of actions $\{\alpha_j\,(\overline{c}) \mid j \in 1..q,\ \overline{c}$ is vector of constants$\} \cup \{\lambda\}$ and $TS_{\text{EB}^3} = (S_1, \rightsquigarrow_1 \doteq \{\xrightarrow{\rho_1}_1\}_{\rho_1 \in Act_1}, s_1^0)$ be the LTS that describes the evolution of EB^3.*

Let $TS_{LNT} = (S_2, \rightsquigarrow_2 \doteq \xrightarrow{1} \circ \xrightarrow{2} \circ \xrightarrow{3}, s_2^0)$ be the LTS that describes the evolution of LNT behaviour $B\,(E_0)$ with communication labels:

$\rho_2 \in \{\alpha_j\ (\overline{c}, \overline{f}) \mid j = 1..q,\ \overline{c}$ *is vector of constants,* $\overline{f}$ *is an attribute variable vector*$\} \cup \{\lambda\}$.

Let also $(E, \mathsf{M}) \rightsquigarrow_1 (E', \mathsf{M}')$ be transition in TS_{EB^3} and $(B\,(E), \mathsf{N}) \rightsquigarrow_2 (B\,(E'), \mathsf{N}')$ be a transition in TS_{LNT} such that:

- M *and* N *are equivalent, i.e.* $\mathsf{M} \approx \mathsf{N}$,
- *1.* $\rho_1 = \alpha_j(\overline{c})$, *and* $\rho_2 = \alpha_j\big(\overline{c},\ \overline{f}_N\big)$ *for* $j \in \{1, \ldots, q\}$ *and* $\overline{c}$ *a vector of constants or*
  2. $\rho_1 = \lambda$, *and* $\rho_2 = \lambda\,(\overline{f}_N)$,

Then, M' *and* N' *are equivalent, i.e.* $\mathsf{M}' \approx \mathsf{N}'$.

Proof. Let vector of constants $\overline{d} = (d_1, \ldots, d_s)$ such that $d_1 \in D_1, \ldots, d_2 \in D_s$. Let the *trace* variable T be equal to empty *trace*, i.e. "$\mathsf{T} = [\ ]$". We need to prove that:

$$\mathsf{N}'(\mathsf{f_i})(\mathsf{d_1}, \ldots, \mathsf{d_s}) = \mathsf{M}'(\mathsf{f_i})(\mathsf{d_1}, \ldots, \mathsf{d_s}) \tag{4.30}$$

for $i \in \{1, \ldots, n\}$. Let "$\tau \doteq [\mathsf{y_1} \leftarrow \mathsf{d_1}, \ldots, \mathsf{y_s} \leftarrow \mathsf{d_s}]$". Combining 4.18 and Figure 4.1, it is derived that:

$$\begin{array}{rcl}
\mathsf{N}'(\mathsf{f_i})(\mathsf{d_1}, \ldots, \mathsf{d_s}) & = & [\![mod\,(w_i^0)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) \\
\text{where} \quad mod\,(w_i^j) & = & w_i^j[f_h\,(front\,(\mathsf{T}),\ \overline{y}) := g_h\,[\mathbf{ord}\,(\overline{y})] \quad \big| \quad \forall h \in \{1, \ldots, n\}] \\
 & & \quad [f_h\,(\mathsf{T},\ \overline{y}) := f_h\,[\mathbf{ord}\,(\overline{y})] \quad \big| \quad \forall h < i] \\
\mathsf{N_0}' & = & \mathsf{N} \oplus [\overline{\mathsf{x}} \leftarrow \overline{\mathsf{c}}, \overline{\mathsf{g}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}]
\end{array}$$

for $i \in \{1, \ldots, n\}$, where $\mathsf{N_i}$ denotes the LNT memory succeding the execution of LNT statement upd_i^j and, as a result, $\mathsf{N_n} = \mathsf{N}$. Similar conventions have been adopted for LNT memories $\mathsf{N_i}'$ for $i \in \{1, \ldots, n\}$. Furthermore, applying Definition 2.3.8, it is found that:

$$\mathsf{M}'(\mathsf{f_i})(\mathsf{d_1}, \ldots, \mathsf{d_s}) = [\![w_i^j]\!]_1^{\mathsf{M}}(\tau \cup \tau', \mathsf{M}), \tag{4.31}$$

where "$\tau' = [\ \mathsf{x_1} \leftarrow \mathsf{c_1}, \ldots, \mathsf{x_p} \leftarrow \mathsf{c_p}]$". Hence, in order to prove that $\mathsf{N}' \approx \mathsf{M}'$, it suffices to prove that:

$$[\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) = [\![w_i^j]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}'). \tag{4.32}$$

We proceed with structural induction on EB^3 type (1) value expressions w_i^j. The proof resembles closely to the corresponding proof of Theorem 4.30. In the following, we consider two cases only and leave the rest of the proof for the interested reader.

- w_i^j reduces to $f_h\,(front\,(\mathsf{T}), \overline{u})$ for $h \in \{1, \ldots, n\}$, where "$\overline{u} = (u_1, \ldots, u_s)$" is a vector of valid LNT expressions (see *attribute function* ordering for details). Applying the dynamic semantics of Section 4.2.1, it is established that:

$$\begin{aligned}
& [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) \\
= \;& [\![mod(f_h\,(front(\mathsf{T}), \overline{u}))]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) \\
= \;& (\mathsf{N_{i-1}}' \oplus \tau)(\mathsf{g_h})([\![mod(u_1)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau), \ldots, [\![mod(u_s)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau)) \\
= \;& (\mathsf{N_{h-1}} \oplus \tau)(\mathsf{f_h})([\![mod(u_1)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau), \ldots, [\![mod(u_s)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau)) \\
= \;& \mathsf{M}(\mathsf{f_h})([\![mod(u_1)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau), \ldots, [\![mod(u_s)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau)) \\
= \;& \mathsf{M}(\mathsf{f_h}')([\![u_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}'), \ldots, [\![u_s]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}')) \\
= \;& \mathsf{M}'(\mathsf{f_h})([\![u_1]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}'), \ldots, [\![u_s]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}')) \\
= \;& [\![f_h\,(front\,(\mathsf{T}), \overline{u})]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}') \\
= \;& [\![w_i^j]\!]_1^{\mathsf{M}}(\tau, \mathsf{M}')
\end{aligned}$$

  where the fact that "$(\mathsf{N_{i-1}}' \oplus \tau)(\mathsf{g_h}) = (\mathsf{N_{h-1}} \oplus \tau)(\mathsf{f_h})$" is justified as follows:

  1. "$(\mathsf{N_{i-1}}')(\mathsf{g_h}) = (\mathsf{N_{h-1}}')(\mathsf{g_h})$", since for $h < i$ *attribute variable* vector g is not modified by LNT statements upd_h^j, ..., and upd_i^j.
  2. the assigned value to $\mathsf{g_h}$ by LNT memory $\mathsf{N_{h-1}}$ is equal to the assigned value to $\mathsf{f_h}$ by LNT memory $\mathsf{N_{h-1}}'$, i.e. "$(\mathsf{N_{h-1}}')(\mathsf{g_h}) = (\mathsf{N_{h-1}})(\mathsf{f_h})$",
  3. from bullets 1, 2 and, by way of compositional reasoning, it is easily derived that "$(\mathsf{N_{i-1}}')(\mathsf{g_h}) = (\mathsf{N_{h-1}})(\mathsf{f_h})$", from which follows that "$(\mathsf{N_{i-1}} \oplus \tau)(\mathsf{g_h}) = (\mathsf{N_{h-1}} \oplus \tau)(\mathsf{f_h})$".

  We recall that $\mathsf{N} \approx \mathsf{M}$, from which we derive that "$(\mathsf{N_{h-1}} \oplus \tau)(\mathsf{f_h}) = \mathsf{M}(\mathsf{f_h}')$". Then, from the induction hypothesis, it follows directly that:

$$[\![mod(u_1)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) = [\![u_1]\!]_{\mathsf{M}}(\tau, \mathsf{M}'),\ \ldots,\ [\![mod(u_s)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) = [\![u_s]\!]_{\mathsf{M}}(\tau, \mathsf{M}').$$

  Furthermore, following the construction of TS_{EB^3} with respect to Sem_{M}, the assigned value to $\mathsf{g_h}$ by EB^3 memory M' is equal to the assigned value to $\overline{\mathsf{f}}$ by M, i.e. "$\mathsf{M}(\mathsf{f_h}') = \mathsf{M}'(\mathsf{f_h})$". Then, by applying the semantics of 2.3.8, we complete the proof.

- w_i^j reduces to $f_h\,(\mathsf{T}, \overline{u})$ for $h < i$, where "$\overline{u} = (u_1, \ldots, u_s)$" is a vector of valid LNT expressions (see *attribute function* ordering for details). Applying the dynamic semantics of Section 4.2.1, it is found that:

$$\begin{aligned}
& [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) \\
= \;& [\![mod(f_h\,(\mathsf{T}, \overline{u}))]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) \\
= \;& [\![mod(f_h\,(\mathsf{T}, \overline{u}))]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau) \\
= \;& [\![mod(w_h^j\ [\overline{y} := \overline{u}])]\!]_{\mathsf{LNT}}(\mathsf{N_{i-1}}' \oplus \tau)
\end{aligned}$$

  Notice that $h < i$. We need to repeat the previous calculations and apply inductive reasoning. The rest of the proof follows the lines of the proof related to the corresponding case of Theorem 4.6.1 and it is, therefore, omitted.

The rest of the cases are similar to the corresponding cases of Theorem 4.6.1. □

Theorem 4.6.3. *Let EB^3 be a well-formed* EB^3 *specification, let* M *be an* EB^3 *memory, and let* N *be an LNT memory such that* $\mathsf{N} \approx \mathsf{M}$. *For* EB^3 *guard ge appearing in EB^3, it follows that:*

$$[\![mod(ge)]\!]_{LNT}(\mathsf{N}) = [\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}).$$

Proof. We proceed with structural induction on guard *ge*:

- *ge* reduces to constant $c \in \mathcal{C}$. Then, applying Definition 2.3.12 and 2.3.7, the result is established as follows:

$$[\![mod(ge)]\!]_{LNT}(\mathsf{N}) = [\![mod(c)]\!]_{LNT}(\mathsf{N}) = \mathcal{F}(c) = [\![c]\!]_3^{\mathsf{M}}(\mathsf{M}) = [\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}).$$

- *ge* reduces to functional term of the form $g(ge_1, \ldots, ge_n)$.

$$\begin{aligned}
[\![mod(ge)]\!]_{LNT}(\mathsf{N}) &= [\![mod\ (g(ge_1, \ldots, ge_l))]\!]_{LNT}(\mathsf{N}) \\
&= \mathcal{F}(g)([\![mod\ (ge_1)]\!]_{LNT}(\mathsf{N}), \ldots,\ [\![mod(ge_l)]\!]_{LNT}(\mathsf{N})) \\
&= \mathcal{F}(g)([\![ge_1]\!]_3^{\mathsf{M}}(\mathsf{M}), \ldots, [\![ge_l]\!]_3^{\mathsf{M}}(\mathsf{M})) \quad \text{(Induction Hypothesis)} \\
&= [\![g(ge_1, \ldots, ge_l)]\!]_3^{\mathsf{M}}(\mathsf{M}) \quad \text{(Definition 2.3.7)} \\
&= [\![ge]\!]_3^{\mathsf{M}}(\mathsf{M})
\end{aligned}$$

  since, applying the induction principle, it is found that "$[\![mod(ge_k)]\!]_{LNT}(\mathsf{N}) = [\![ge_k]\!]_3^{\mathsf{M}}(\mathsf{M})$" for all $k \in \{1, \ldots, l\}$.

- *ge* reduces to an *attribute function* call of the form "$f_i\,(\mathsf{T},\, v_1, \ldots, v_s)$". Then, $[\![ge]\!]_3^{\mathsf{M}}\,(\mathsf{M})$ can be transformed as follows:

$$\begin{aligned}
[\![mod(ge)]\!]_{LNT}(\mathsf{N}) &= [\![mod\ \big(f_i\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_{LNT}(\mathsf{N}) \\
&= \mathsf{N}(\mathsf{f_i})([\![mod(v_1)]\!]_{LNT}(\mathsf{N}), \ldots, [\![mod(v_s)]\!]_{LNT}(\mathsf{N})) \\
&= \mathsf{M}(\mathsf{f_i})([\![v_1]\!]_3^{\mathsf{M}}\,(\mathsf{M}) \ldots,\ [\![v_s]\!]_3^{\mathsf{M}}(\mathsf{M})) \\
&= [\![f_i\,(\mathsf{T}, v_1, \ldots, v_s)]\!]_3^{\mathsf{M}}(\mathsf{M}) \quad \text{(Definition 2.3.7)} \\
&= [\![ge]\!]_3^{\mathsf{M}}(\mathsf{M})
\end{aligned}$$

  since, by the induction principle, it is "$[\![mod(v_k)]\!]_{LNT}(\mathsf{N}) = [\![v_k]\!]_3^{\mathsf{M}}(\mathsf{M})$" for $k \in \{1, \ldots, s\}$ and, by way of $\mathsf{M} \approx \mathsf{N}$, it is "$\mathsf{M}(\mathsf{f_i}) = \mathsf{N}(\mathsf{f_i})$".

□

4.6.4 Bisimulation Equivalence of EB^3 and LNT Specifications

Theorem 4.6.4. ***Bisimulation Equivalence of EB^3 and LNT***

Let EB^3 be EB^3 *specification including process prototype definition "main = E_0". Let also TS_{EB^3} be the LTS that corresponds to EB^3, and let TS_{LNT} be the LTS that corresponds to LNT behaviour $B(E)$ (see Section 4.6.3 for details on the construction of TS_{EB^3} and TS_{LNT}). Then, TS_{EB^3} is bisimilar to TS_{LNT}, i.e. $TS_{EB^3} \sim TS_{LNT}$.*

Proof. We consider the following relation:

$$R = \{ \ \langle (ge \Rightarrow E, \mathsf{M}), (B(E, ge), \mathsf{N}) \rangle \ \mid \ E \text{ is an EB}^3 \text{ process expression } \wedge \ B(E, ge) \text{ is the LNT equivalent of } ge \Rightarrow E \ \wedge \ \mathsf{M} \text{ is an EB}^3 \text{ memory } \wedge \ \mathsf{N} \text{ is an LNT memory } \wedge \ \mathsf{M} \approx \mathsf{N} \ \}$$

Note that the definition of relation R is not restricted to tuples of the form $(E, \mathsf{M}) \in S_{\mathsf{M}}$ and $(B(E, ge), \mathsf{N}) \in S_{\mathsf{T}}$ for any EB^3 expression E. We recall that "$s^0_{\mathsf{T}/\mathsf{M}} = (E_0, \mathsf{M_0})$" and "$s^0_{LNT} = (B(E_0, true), \mathsf{N^0_n})$". By force of Theorem 4.6.1, EB^3 memory $\mathsf{M_0}$ is compatible with LNT memory $\mathsf{N^0_n}$, from which follows that $\langle (E, \mathsf{M_0}), (B(E_0, true), \mathsf{N^0_n}) \rangle \in R$.

Let $ge \Rightarrow E$ be an EB^3 process expression and let M be an EB^3 memory associated to $ge \Rightarrow E$ such that $(ge \Rightarrow E, \mathsf{M}) \in TS_{\text{EB}^3}$. Let also $B(E, ge)$ be the equivalent LNT expression of $ge \Rightarrow E$ and let N be the associated LNT memory to $B(E, ge)$ such that $(B(E, ge), \mathsf{N}) \in TS_{LNT}$ and $\mathsf{M} \approx \mathsf{N}$, from which follows that $\langle (ge \Rightarrow E, \mathsf{M}), (B(E, ge), \mathsf{N}) \rangle \in R$ and $(ge \Rightarrow E, \mathsf{M}) \rightsquigarrow_1 (E', \mathsf{M'}) \in \delta_{\text{EB}^3}$, where $\rho_1 \in \{\alpha_j\,(\overline{c}) \mid j \in 1..q, \ \overline{c} \text{ is a vector of constants}\} \cup \{\lambda\}$ and $\mathsf{M'}$ is an EB^3 memory.

Then, for every transition of the form $(B\,(E, ge), \mathsf{N}) \rightsquigarrow_2 (B(E', true), \mathsf{N'}) \in \delta_{LNT}$, for which $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N) \mid j \in 1..q, \ \overline{c} \text{ is a vector of constants}\} \cup \{\lambda\,(\overline{f}_N)\}$, where $\overline{f}_N$ is the value assigned to *attribute variable* vector $\overline{f}$ by LNT memory N, i.e. $\overline{f}_N \doteq \mathsf{N}(\overline{f})$, we need to prove that $\langle (E', \mathsf{M'}), (B\,(E', true), \mathsf{N'}) \rangle \in R$. The converse proposition should be established as well.

Let $ge \Rightarrow E$ denote the EB^3 process expression that remains to be executed, i.e. $main = ge \Rightarrow E$. We proceed with structural induction on E:

- Let $E = \alpha_j\,(\overline{v})$ for $j \in \{1, \ldots, q\}$, where $\overline{v}$ is a vector of EB^3 type (1) value expressions. Note that the only possible EB^3 transition scenario is the following:

$$\frac{[\![ge]\!]^{\mathsf{M}}_3 = \mathbf{true} \quad (\alpha_j\,(\overline{v}), \mathsf{M}) \xrightarrow{\alpha_j\,(\overline{c})}_1 \big(\surd, upd\big(\alpha_j\,(\overline{c}), \mathsf{M}\big)\big)}{(ge \Rightarrow \alpha_j\,(\overline{v}), \mathsf{M}) \xrightarrow{\alpha_j\,(\overline{c})}_1 \big(\surd, upd\big(\alpha_j\,(\overline{c}), \mathsf{M}\big)\big)},$$

  where the body of *upd* is specified as in Chapter 2 and $\overline{c}$ is a vector of constants. Notice that "$[\![ge]\!]^{\mathsf{M}}_3(\mathsf{M}) = \mathsf{true}$" follows from Sem_{M}. Then, by replacing EB^3 process expression E with action $\alpha_j\,(\overline{v})$ in Definition 4.22, we obtain the following LNT behaviour:

$$\begin{array}{rcl} B\,\big(\alpha_j\,(\overline{v}), ge\big) & = & \mathbf{par}\ \alpha_1, \ldots, \alpha_q, \lambda\ \mathbf{in} \\ & & \quad \alpha_j\,(\overline{v}, ?\overline{f})\ \mathbf{where}\ mod(ge)\ \ ||\ \ \mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop} \\ & & \mathbf{end\ par} \end{array}$$

  Similarly, by replacing EB^3 process expression E with the inert action $\surd$ in Definition 4.22 and by replacing guard C with *true*, we obtain the following LNT behaviour:

$$\begin{array}{rcl} B\,(\surd, true) & = & \mathbf{par}\ \alpha_1, \ldots, \alpha_q, \lambda\ \mathbf{in} \\ & & \quad \mathbf{null\ where\ true}\ \ ||\ \ \mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop} \\ & & \mathbf{end\ par} \end{array}$$

We prove that the execution of $B\,(\alpha_j\,(\overline{v}), ge)$ triggers the following LNT reduction rule:

$$\frac{(4.34)\quad (4.36)}{\{B\,(\alpha_j\,(\overline{v}), ge)\}\ \mathsf{N} \rightsquigarrow_2 \{B\,(\surd, \mathit{true})\}\ \mathsf{N}'}, \tag{4.33}$$

where N' is an LNT memory for which $\mathit{upd}\big(\alpha_j\,(\overline{c}),\ \mathsf{M}\big) \approx \mathsf{N}'$.

Rule (4.33) reflects the unique reduction scenario for the parallel composition construct $B\,(\alpha_j\,(\overline{v}), \mathit{true})$ to $B\,(\surd, \mathit{true})$. As a first step, the *attribute vector* $\overline{\mathsf{f}}$ is stored to $\overline{\mathsf{g}}$ namely via the execution of LNT statement "$\overline{g} := \overline{f}$". Then, a series of offers related to communication label $\alpha_j\,(\overline{v}, \overline{f}_N)$ occurs, which is followed by the execution of LNT statement "$\mathit{upd}_1^j;\ \ldots; \mathit{upd}_n^j$" that modifies the *attribute variables*.

In particular, Rule (4.33) triggers Rule (4.34) that corresponds to the left counterpart of LNT behaviour $B\,(\alpha_j\,(\overline{v}), \mathit{true})$ and Rule (4.36) that corresponds to the right counterpart of LNT behaviour $B\,(\alpha_j\,(\overline{v}), \mathit{true})$.

On the other hand, the following reduction rule:

$$\frac{\{\mathit{mod}\,(ge)\}\ \mathsf{N} \rightarrow_e \mathsf{true}\quad (4.35)}{\{\alpha_j\,(\overline{v}, ?\overline{f})\ \mathbf{where}\ \mathit{mod}\,(ge)\}\ \mathsf{N} \xrightarrow{\alpha_j\,(\overline{v}, \overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N}} \tag{4.34}$$

updates the LNT memory N, corresponding to LNT behaviour "$\alpha_j\,(\overline{v}, ?\overline{f})$ **where true**", to LNT memory "$\mathsf{N} \oplus [\,\overline{\mathsf{f}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}\,] = \mathsf{N}$" (since both counterparts of the parallel composition have initially the same LNT memory N, assignment $\overline{\mathsf{f}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}$ is already present in LNT memory N) that contains the bound value $\overline{f}_N$ to *attribute variable* vector $\overline{\mathsf{f}}$ as a result of LNT behaviour "$\alpha_j\,(\overline{v}, ?\overline{f})$ **where true**" synchronizing with LNT behaviour "$\alpha_j\,(?\overline{x_j}, !\,\overline{f})$" of the right counterpart on communication label $\alpha_j\,(\overline{v}, \overline{f}_N)$ such that "$\rightarrow_2 = \xrightarrow{\alpha_j\,(\overline{v}, \overline{f}_N)}_b$". For this to happen, $\mathit{mod}\,(ge)$ is evaluated to true, i.e. "$\{\mathit{mod}\,(ge)\} \rightarrow_e \mathsf{true}$", which is a direct consequence of $[\![ge]\!]_3^{\mathsf{M}} = \mathsf{true}$ and Theorem 4.6.3. Rule (4.35) is specified as follows:

$$\frac{\dfrac{\dfrac{\{\overline{v}\}\ \mathsf{N} \rightarrow_e \overline{\mathsf{v}}}{\{\overline{v}\ \#\ \overline{\mathsf{v}}\}\ \mathsf{N} \rightarrow_p \mathsf{N}}}{\{!\,\overline{v}\ \#\ \overline{\mathsf{v}}\}\ \mathsf{N} \rightarrow_o \mathsf{N}} \quad \dfrac{\{\overline{f}\}\ \mathsf{N} \rightarrow_e \mathsf{N}}{\{?\overline{f}\ \#\ \overline{\mathsf{f}}_{\mathsf{N}}\}\ \mathsf{N} \rightarrow_o \mathsf{N}}}{\{\alpha_j\,(!\,\overline{v}, ?\overline{f})\}\ \mathsf{N} \xrightarrow{\alpha_j\,(\overline{v}, \overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N}} \tag{4.35}$$

The standard semantics of offers guarantees that any variable bindings in the system should take place starting from the leftmost offer to the rightmost offer of communication label $\alpha_j\,(\overline{v}, \overline{f}_N)$. In particular, send offer $!\,\overline{v}$ matches value $\overline{\mathsf{v}}$, which is the value to which $\overline{v}$ is evaluated without affecting the current memory. Then, receive offer $?\overline{f}$ matches value $\overline{f}_N$ that is communicated by the right counterpart of $B\,(\alpha_j\,(\overline{v}))$ during the aforementioned synchronization.

It can be easily verified that the previous operation takes place after the execution of "$\overline{g} := \overline{f}$" in Rule (4.36). Hence, it is "$\xrightarrow{1} = \xrightarrow{\mathbf{exit}}_b$". More specifically, Rule (4.36) takes the following form:

$$\frac{(4.37)}{\{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_b \{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N}'} \tag{4.36}$$

modifies the right counterpart of LNT behaviour $B\,(\alpha_j\,(\overline{v}))$, which is a typical **loop** construct. Note that the LNT memory referring to this counterpart is N too. We recall that the dynamic semantics of breakable **loop** is defined with the aid of intermediate construct **trap**. In keeping with the standard LNT semantics, LNT statement "$\overline{g} := \overline{f};\ D$" terminates normally (without a **break**) causing the **trap** construct to reduce to lp. The syntax of lp is given in (4.21). In the following, we denote "$upd \doteq upd_1^j;\ \ldots; upd_n^j$" for brevity. Hence, the induced Rule (4.37) is specified as follows:

$$\dfrac{\dfrac{\{\overline{f}\}\mathsf{N} \rightarrow_e \overline{\mathsf{f}}_{\mathsf{N}}}{\{\overline{g} := \overline{f}\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_b \{\mathbf{stop}\}\ \mathsf{N}_1} \qquad \dfrac{\dfrac{(4.38)\quad \{upd\}\ \mathsf{N}_2 \xrightarrow{\mathbf{exit}}_b \{\mathbf{null}\}\mathsf{N}'}{\{\alpha_j\,(?\overline{x_j}, !\,\overline{f});\ upd\}\mathsf{N}_1 \xrightarrow{\mathbf{exit}}_b \{\mathbf{null}\}\ \mathsf{N}'}}{\{D\}\ \mathsf{N}_1 \xrightarrow{\mathbf{exit}}_b \{\mathbf{null}\}\ \mathsf{N}'}}{\{\overline{g} := \overline{f};\ D\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_b \{\mathbf{null}\}\ \mathsf{N}'} \tag{4.37}$$

where "$\mathsf{N}_1 = \mathsf{N} \oplus \left[\,\overline{\mathsf{g}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}\,\right]$". Applying compositional reasoning, Rule (4.37) triggers two rule scenarios. The left side scenario treats LNT statement "$\overline{g} := \overline{f}$" and the consequent update of LNT memory N to N_1. The syntax of D is given in Equation 4.20. The right side scenario coincides with the standard reduction rule for the **select** construct. In particular, LNT behaviour "$\alpha_j\,(?\overline{x}, !\,\overline{f});\ upd$" is picked by communication label $\alpha_j\,(\overline{v}, \overline{f}_N)$ among all possible LNT behaviours constituting LNT behaviour D. Then, applying compositional reasoning, Rule (4.37) triggers Rule (4.38):

$$\dfrac{\dfrac{\{\overline{x}\ \#\ \overline{\mathsf{v}}\}\ \mathsf{N}_1 \rightarrow_p \mathsf{N}_2}{\{?\overline{x}\ \#\ \overline{\mathsf{v}}\}\ \mathsf{N}_1 \rightarrow_o \mathsf{N}_2} \qquad \dfrac{\{\overline{f}\}\ \mathsf{N}_2 \rightarrow_e \overline{f}_N}{\{!\,\overline{f}\ \#\ \overline{f}_N\}\ \mathsf{N}_2 \rightarrow_o \mathsf{N}_2}}{\{\alpha_j\,(?\overline{x}, !\,\overline{f})\}\ \mathsf{N}_1 \xrightarrow{\alpha_j\,(\overline{v},\overline{f}_N)}_b \mathsf{N}_2} \tag{4.38}$$

that treats the execution of communication label $\alpha_j\,(\overline{v}, \overline{f}_N)$ such that "$\xrightarrow{2}\doteq\xrightarrow{\alpha_j\,(\overline{v},\overline{f}_N)}_b$" and the execution of LNT statement "$upd_1^j;\ \ldots; upd_n^j$" such that "$\xrightarrow{3}\doteq\xrightarrow{\mathbf{exit}}_b$", which justifies the use of $\rightsquigarrow_2$ notation in Rule (4.33). First, receive offer $?\overline{x}$ matches value $\overline{\mathsf{v}}$ that is communicated by the left counterpart of $B\,(\alpha_j\,(\overline{v}))$ during the aforementioned synchronization, thus, updating LNT memory N_2 to:

$$\mathsf{N}_2 = \mathsf{N}_1 \oplus \left[\,\overline{\mathsf{x}} \leftarrow \overline{\mathsf{v}}\,\right] = \mathsf{N} \oplus \left[\,\overline{\mathsf{x}} \leftarrow \overline{\mathsf{v}},\ \overline{\mathsf{g}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}\,\right]. \tag{4.39}$$

Then, send offer $!\,\overline{f}$ matches value $\overline{f}_N$, which is the value assigned to the *attribute variables* by LNT memory N (as seen previously) without affecting the current memory. In the absence of guarded expression, the transfer of $\overline{f}_N$ from the right counterpart of $B(\alpha_j(\overline{v}))$ to the left counterpart of $B(\alpha_j(\overline{v}))$ via communication offer does not affect the program flow directly. Then, applying the theory of Section 4.6.3, LNT memory N' can be calculated as follows:

$$\begin{aligned}\mathsf{N}' \;=\; & \left[\,\overline{\mathsf{x}} \leftarrow \overline{\mathsf{c}},\ \overline{\mathsf{g}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}\,\right] \;\cup \\ & \left[\,\mathsf{f_i}\,[\,\overline{\mathsf{d}}\,] \leftarrow [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}}\big(\mathsf{N}'_{\mathsf{i}-1} \oplus \tau\big) \;\;\middle|\;\; d_1 \in D_1,\ \ldots,\ d_s \in D_s,\ 1 \leq i \leq n\,\right]\end{aligned}$$

where $\mathsf{N_i}'$ corresponds to the LNT memory the moment succeeding the execution of upd_i^j (see Section 4.6.3 for details) and "$\tau = [\ \mathsf{x_1} \leftarrow \mathsf{c_1},\ \ldots,\ \mathsf{x_p} \leftarrow \mathsf{c_p}\ ]$". By appeal to Theorem 4.6.2, we deduce that $\mathsf{M}' \approx \mathsf{N}'$, which establishes the truth of Rule (4.33). The converse direction of the proof follows directly, which completes the proof for case $E = \alpha_j(\overline{v})$.

- Let $E = \lambda$. Then, the only possible EB^3 transition is the following:

$$\frac{[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathbf{true} \quad (\lambda, \mathsf{M}) \xrightarrow{\lambda}_1 (\surd, \mathsf{M})}{(ge \Rightarrow \lambda, \mathsf{M}) \xrightarrow{\lambda}_1 (\surd, \mathsf{M})}.$$

As previously, it follows from Sem_{M} that "$[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true}$". By replacing EB^3 process expression E with the inert action λ in formula (4.22), we obtain the following LNT behaviour:

$$\begin{array}{rcl} B\,(\lambda, ge) & = & \mathbf{par}\ \alpha_1,\ \ldots,\ \alpha_q,\ \lambda\ \mathbf{in} \\ & & \quad \lambda\,(?\overline{f})\ \mathbf{where}\ mod\,(ge)\ \ ||\ \ \mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop} \\ & & \mathbf{end\ par} \end{array}$$

We need to prove that the execution of $B\,(\lambda, ge)$ triggers the following LNT reduction rule:

$$\frac{(4.41) \quad (4.43)}{\{B\,(\lambda, ge)\}\ \mathsf{N} \rightsquigarrow_2 \{B\,(\surd, true)\}\ \mathsf{N}'}, \tag{4.40}$$

where N' is an LNT memory for which $\mathsf{M} \approx \mathsf{N}'$. The proof follows the lines of the previous proof. In particular, Rule (4.41):

$$\frac{\{mod\,(ge)\}\ (\mathsf{N}) \rightarrow_e \mathsf{true} \quad (4.42)}{\{\lambda\,(?\overline{f})\ \mathbf{where}\ \mathsf{true}\}\ \mathsf{N} \xrightarrow{\lambda\,(\overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N}} \tag{4.41}$$

Note also that "$\{mod\,(ge)\} \rightarrow_e \mathsf{true}$" is, by force of Theorem 4.6.3, equivalent to "$[\![mod(ge)]\!]_{LNT}(\mathsf{N}) = \mathsf{true}$". Rule (4.42) is specified as follows:

$$\frac{\dfrac{\{\overline{f}\}\ \mathsf{N} \rightarrow_e \mathsf{N}}{\{?\overline{f}\ \#\ \overline{\mathsf{f}}_{\mathsf{N}}\}\ \mathsf{N} \rightarrow_o \mathsf{N}}}{\{\lambda\,(?\overline{f})\}\ \mathsf{N} \xrightarrow{\lambda\,(\overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N}} \tag{4.42}$$

where "$\mathsf{N} = \mathsf{N} \oplus [\,\overline{\mathsf{f}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}\,]$"' as stated previously. Rule (4.43) takes the following form:

$$\frac{\dfrac{(4.44)}{\{\mathbf{trap}\ \overline{g} := \overline{f};\ D\ \mathbf{else}\ lp\ \mathbf{end\ trap}\}\ \mathsf{N} \rightsquigarrow_2 \{lp\}\ \mathsf{N}'}}{\{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N} \rightsquigarrow_2 \{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N}'} \tag{4.43}$$

The induced Rule (4.44) is specified as follows:

$$\frac{\dfrac{\{\overline{f}\}\ \mathsf{N} \rightarrow_e \overline{\mathsf{f}}_{\mathsf{N}}}{\{\overline{g} := \overline{f}\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_b \{\mathbf{stop}\}\ \mathsf{N}_1} \quad \dfrac{\dfrac{(4.45) \quad \{upd\}\ \mathsf{N}_2 \xrightarrow{\mathbf{exit}}_b \{\mathbf{null}\}\ \mathsf{N}'}{\{\lambda\,(!\overline{f});\ upd\}\ \mathsf{N}_1 \xrightarrow{\mathbf{exit}}_b \{\mathbf{null}\}\ \mathsf{N}'}}{\{D\}\ \mathsf{N}_1 \xrightarrow{\mathbf{exit}}_b \{\mathbf{null}\}\ \mathsf{N}'}}{\{\overline{g} := \overline{f};\ D\}\ \mathsf{N} \rightsquigarrow_2 \{\mathbf{null}\}\ \mathsf{N}'} \tag{4.44}$$

where "$\mathsf{N}_1 = \mathsf{N} \oplus [\,\overline{\mathsf{g}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}}\,]$" and $\mathsf{N}_2 = \mathsf{N}_1$ as receives no parameters other than *attribute variables*. Then, Rule (4.44) triggers the following reduction rule:

$$\frac{\dfrac{\{\overline{f}\}\ \mathsf{N}' \rightarrow_e \overline{\mathsf{f}}_{\mathsf{N}}}{\{!\overline{f}\ \#\ \overline{f}_N\}\ \mathsf{N}' \rightarrow_o \mathsf{N}'}}{\{\lambda\,(!\overline{f})\}\ \mathsf{N}' \xrightarrow{\lambda\,(\overline{f}_N)}_b \mathsf{N}'} \tag{4.45}$$

Then, applying the theory of Section 4.6.3, LNT memory N' can be calculated as follows:

$$\begin{array}{rcl} \mathsf{N}' & = & \left[\, \overline{\mathsf{g}} \leftarrow \overline{\mathsf{f}}_{\mathsf{N}} \,\right] \quad \cup \\ & & \left[\, \mathsf{f}_{\mathsf{i}} \left[\, \overline{\mathsf{d}} \,\right] \leftarrow [\![mod\,(w_i^j)]\!]_{\mathsf{LNT}} \left(\mathsf{N}^{\mathsf{j}}_{\mathsf{i}-1}\right) \quad \middle| \quad d_1 \in D_1,\ \ldots,\ d_s \in D_s,\ 1 \leq i \leq n \,\right] \end{array}$$

where $\mathsf{N_i}'$ corresponds to the LNT memory the moment succeeding the execution of upd_i^j. By appeal to Theorem 4.6.2, we deduce that $\mathsf{M} \approx \mathsf{N}'$, which establishes the truth of Rule (4.33). The converse direction of the proof follows easily, which completes the proof for case $E = \lambda$.

- Let $E = E_1.E_2$.

  We consider tuple $\left\langle (ge \Rightarrow E_1.E_2, \mathsf{M}), (B\,(E_1.E_2, ge),\ \mathsf{N}) \right\rangle \in R$ and transition $(ge \Rightarrow E_1.E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1'.E_2, \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\, j \in 1..q \mid \overline{c} \text{ is a vector of constants } \} \cup \{\lambda(\overline{f}_N)\}$.

  Hence, we need to prove that if $\left(B\,(E_1.E_2, ge),\ \mathsf{N}\right) \rightsquigarrow_2 \left(B\,(E_1'.E_2, true),\ \mathsf{N}'\right)$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\, j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, then it follows that $\left\langle (E_1'.E_2, \mathsf{M}'),\ (B\,(E_1'.E_2, true), \mathsf{N}') \right\rangle \in R$.

  Then, the only applicable EB^3 transition is the following:

$$\frac{[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true} \quad (E_1, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1', \mathsf{M}')}{(ge \Rightarrow E_1.E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1'.E_2, \mathsf{M}')}.$$

  First, we replace EB^3 process expression E with $E_1.E_2$ in relation (4.22). Then, by virtue of identity relation "$t\,(E_1.E_2, ge) = t\,(E_1, ge).\, t\,(E_2, true)$", LNT behaviour $B\,(E_1.\ E_2, ge)$ can be calculated as follows:

$$\begin{array}{rcl} B\,(E_1.E_2, ge) & = & \textbf{par } \alpha_1, \ldots, \alpha_q, \lambda \textbf{ in} \\ & & \quad t\,(E_1, ge).\, t\,(E_2, true) \ \|\| \ \textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop} \\ & & \textbf{end par} \end{array}$$

  We suppose that $\left\langle (ge \Rightarrow E_1, \mathsf{M}), (B\,(E_1, ge), \mathsf{N}) \right\rangle \in R$ and consider transition $(ge \Rightarrow E_1,\ \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1',\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\, j \in 1..q \mid \overline{c} \text{ is a vector of constants } \} \cup \{\lambda\}$. By appeal to the induction principle, there should be transition $\left(B\,(E_1, ge), \mathsf{N}\right) \rightsquigarrow_2 \left(B\,(E_1', true), \mathsf{N}'\right)$ that does not belong to $\delta_{E_0}^{LNT}$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\, j \in 1..q\} \cup \{\lambda\}$ such that $\left\langle (E_1',\ \mathsf{M}'),\ (B\,(E_1', true),\ \mathsf{N}') \right\rangle \in R$.

  By way of the inductive principle, transition $(B\,(E_1, ge),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1', true),\ \mathsf{N}')$ induces the following set of reduction rules:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N_1} \quad (4.46)$$

$$\frac{\{t\,(E_1, ge)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1', true)\}\ \mathsf{N_2} \quad (4.46)}{(B\,(E_1, ge),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1', true),\ \mathsf{N}')} \quad (4.47)$$

  where $\mathsf{N_1}$ and $\mathsf{N_2}$ are intermediate LNT memories and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that Rule (4.46) relates to the right construct of $B\,(E_1, true)$, i.e. to

memory process M. By way of compositionality, it follows directly that:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.48)$$

$$\frac{\{t\,(E_1, ge).\ t\,(E_2, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1', true).\ t\,(E_2, true)\}\ \mathsf{N}'' \quad (4.48)}{(B\,(E_1.E_2, ge),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1'.E_2, true),\ \mathsf{N}')} \quad (4.49)$$

and, as a result, $\langle (E_1'.E_2, \mathsf{M}'),\ (B\,(E_1'.E_2, true), \mathsf{N}')\rangle \in R$.

- Let $E = \surd.E_2$. We recall that $\surd$ denotes complete execution. As a result, guard ge has to be trivially $true$.

  We consider tuple $\langle (\surd.E_2, \mathsf{M}),\ (B\,(\surd.E_2, true),\ \mathsf{N})\rangle \in R$ and transition $(\surd.E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_2',\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c}$ is a vector of constants $\} \cup \{\lambda\}$.

  Hence, we need to prove that if $\big(B\,(\surd.E_2, true),\ \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(E_2', true),\ \mathsf{N}'\big)$, where label $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\ j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, it follows that $\langle (E_2', \mathsf{M}'),\ (B\,(E_2', true), \mathsf{N}')\rangle \in R$.

  Hence, the only applicable EB^3 transition is the following:

$$\frac{[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true} \quad (E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_2', \mathsf{M}')}{(ge \Rightarrow \surd.E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_2', \mathsf{M}')}.$$

  We suppose that $\langle (E_2, \mathsf{M}), (B\,(E_2, true), \mathsf{N})\rangle \in R$ and consider transition $(E_2,\ \mathsf{M}) \xrightarrow{\rho_1}_1 (E_2',\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c}$ is a vector of constants $\} \cup \{\lambda\}$. By appeal to the induction principle, there should be transition $\big(B\,(E_2, true), \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(E_2', true), \mathsf{N}'\big)$ that does not belong to $\delta_{E_0}^{LNT}$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\ j \in 1..q\} \cup \{\lambda\}$ such that $\langle (E_2',\ \mathsf{M}'),\ (B\,(E_2', true),\ \mathsf{N}')\rangle \in R$.

  By appeal to the definition of Figure 4.3, $t\,(\surd.\ E_2, ge)$ can be transformed as follows:

$$t\,(\surd.\ E_2, true) = t\,(\surd, true).\ t\,(E_2, true) = \textbf{null}.\ t\,(E_2, true) = t\,(E_2, true).$$

  Note that **null** has been removed for brevity. Hence, transition $(B\,(E_2, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_2', true),\ \mathsf{N}')$ induces the following set of reduction rules:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.50)$$

$$\frac{\{t\,(E_2, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_2', true)\}\ \mathsf{N}_2 \quad (4.50)}{(B\,(E_2, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_2', true),\ \mathsf{N}')} \quad (4.51)$$

  where N_1 and N_2 are intermediate LNT memories and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that rule (4.50) relates to the right construct of $B\,(E_2, true)$, i.e. to memory process M. Then, it follows easily that:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.52)$$

$$\frac{\{t\,(E_2, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_2', true)\}\ \mathsf{N}_2 \quad (4.52)}{(B\,(\surd.E_2, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_2', true),\ \mathsf{N}')} \quad (4.53)$$

  From Rule (4.53) and $\mathsf{M}' \approx \mathsf{N}'$, it follows that:

$$\langle (E_2', \mathsf{M}'),\ (B\,(E_2', true), \mathsf{N}')\rangle \in R.$$

- Let $E = ge' \Rightarrow E_1$.

  We assume tuple $\langle (ge \Rightarrow ge' \Rightarrow E_1, \mathsf{M}), (B\,(ge' \Rightarrow E_1, ge), \mathsf{N}) \rangle \in R$ and transition $(ge \Rightarrow ge' \Rightarrow E_1, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1',\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c}$ is a vector of constants $\} \cup \{\lambda\}$.

  Hence, we need to prove that if $\big(B\,(ge' \Rightarrow E_1, ge),\ \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(E_1', \mathit{true}),\ \mathsf{N}'\big)$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\ j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, it follows that $\langle (E_1', \mathsf{M}'),\ (B\,(E_1', \mathit{true}), \mathsf{N}') \rangle \in R$.

  First, we make use of the following identity:

$$ge \Rightarrow (ge' \Rightarrow E_1) \doteq (ge \Rightarrow ge') \Rightarrow E_1$$

  By appeal to the definition of Figure 4.3, it follows that:

$$t(ge' \Rightarrow E_1, ge) = t(E_1, ge \textbf{ andthen } ge')$$

  Then, LNT behaviour $B\,(ge' \Rightarrow E_1, ge)$ can be calculated as follows:

$$\begin{array}{rcl} B\,(ge' \Rightarrow E_1, ge) & = & \textbf{par } \alpha_1, \ldots, \alpha_q, \lambda \textbf{ in} \\ & & \quad t\,(E_1, ge \textbf{ andthen } ge')\ \ ||\ \ \textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop} \\ & & \textbf{end par} \\ & = & B\,(E_1, ge \textbf{ andthen } ge') \end{array}$$

  The induction principle can, therefore, be applied to $B\,(E_1, ge \textbf{ andthen } ge')$ as E_1 is strictly smaller than $ge' \Rightarrow E_1$. The rest of the proof is obvious.

- Let $E = E_1|E_2$.

  We consider tuple $\langle (ge \Rightarrow E_1|E_2, \mathsf{M}),\ (B\,(E_1|E_2, ge),\ \mathsf{N}) \rangle \in R$ and transition $(ge \Rightarrow E_1|E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1',\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c}$ is a vector of constants $\} \cup \{\lambda\}$.

  Hence, we need to prove that if $\big(B\,(E_1|E_2, ge),\ \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(E_1', \mathit{true}),\ \mathsf{N}'\big)$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\ j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, it follows that:

$$\langle (E_1', \mathsf{M}'), (B\,(E_1', \mathit{true}), \mathsf{N}') \rangle \in R.$$

  Following one of the corresponding reduction rules of Sem_{M}, we may deduce that:

$$\frac{(E_1, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1',\ \mathsf{M}') \quad [\![ge]\!]_3^{\mathsf{M}}\ (\mathsf{M}) = \mathsf{true}}{(ge \Rightarrow E_1|E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1',\ \mathsf{M}')}$$

  Note also that "$\{mod\,(ge)\}\ \mathsf{N} \rightarrow_e \mathsf{true}$" is, by force of Theorem 4.6.3, equivalent to "$[\![ge]\!]_3^{\mathsf{M}}\ (\mathsf{M}) = \mathsf{true}$".

  By appeal to the definition of Figure 4.3, it follows that:

$$t(E_1|E_2, ge) = \textbf{select } t(E_1, ge) \ [\ ]\ t(E_2, ge) \textbf{ end select}$$

Then, LNT behaviour $B\,(E_1|E_2, ge)$ can be calculated as follows:

$$\begin{array}{rcl} B\,(E_1|E_2, ge) & = & \textbf{par } \alpha_1, \ldots, \alpha_q, \lambda \textbf{ in} \\ & & \quad t\,(E_1|E_2, ge) \;\; || \;\; \textbf{loop } \overline{g} := \overline{f};\; D \textbf{ end loop} \\ & & \textbf{end par} \end{array}$$

Hence, transition $(B\,(E_1, ge),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1', true),\ \mathsf{N}')$ induces the following set of reduction rules:

$$\{\textbf{loop } \overline{g} := \overline{f};\; D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\; D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.54)$$

$$\frac{\{t\,(E_1, ge)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1', true)\}\ \mathsf{N}_2 \quad (4.54)}{(B\,(E_1, ge),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1', true),\ \mathsf{N}')} \quad (4.55)$$

where N_1 and N_2 are intermediate LNT memories and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that rule (4.54) relates to the right construct of $B\,(E_2, true)$, i.e. to memory process M. Then, it follows easily that:

$$\frac{\{t\,(E_1, ge)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1', true)\}\ \mathsf{N}_2}{\{t\,(E_1|E_2, ge)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1', true)\}\ \mathsf{N}_2} \quad (4.56)$$

$$\{\textbf{loop } \overline{g} := \overline{f};\; D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\; D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.57)$$

$$\frac{(4.56) \quad (4.57)}{(B\,(E_1|E_2, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1', true),\ \mathsf{N}')} \quad (4.58)$$

From Rule (4.58) and $\mathsf{M}' \approx \mathsf{N}'$, it follows that:

$$\langle (E_1', \mathsf{M}'),\ (B\,(E_1', true), \mathsf{N}') \rangle \in R.$$

- Let $E = E_0^*$ and $ge = true$.

  Hence, we consider tuple $\langle (E_0^*, \mathsf{M}),\ (B\,(E_0^*, true),\ \mathsf{N}) \rangle \in R$ and transition $(E_0^*, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_0'.E_0^*,\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c} \text{ is a vector of constants } \} \cup \{\lambda\}$.

  Then, we need to prove that if $\left(B\,(E_0^*, true),\ \mathsf{N}\right) \rightsquigarrow_2 \left(B\,(E_0'.E_0^*, true),\ \mathsf{N}'\right)$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\ j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, it follows that $\langle (E_0'.E_0^*, \mathsf{M}'),\ (B\,(E_0'.E_0^*, true), \mathsf{N}') \rangle \in R$.

  The following EB^3 reduction rule can be applied to EB^3 expression E_0:

$$\frac{(E_0, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_0',\ \mathsf{M}')}{(E_0^*, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_0'.E_0^*,\ \mathsf{M}')}$$

  By way of the induction principle, transition $(B\,(E_0, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_0', true),\ \mathsf{N}')$ induces the following set of reduction rules:

$$\{\textbf{loop } \overline{g} := \overline{f};\; D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\; D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.59)$$

$$\frac{\{t\,(E_0, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_0', true)\}\ \mathsf{N}_2 \quad (4.59)}{(B\,(E_0, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_0', true),\ \mathsf{N}')} \quad (4.60)$$

where N_1 and N_2 are intermediate LNT memories and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that rule (4.59) relates to the right construct of $B\,(E_0, \mathit{true})$, i.e. to memory process M.

By appeal to the definition of Figure 4.3, we may write that:

$$t\,(E_0{}^*, \text{true}) \quad = \quad \textbf{loop } L_{E_0} \textbf{ in } B_1 \textbf{ end loop}$$

where "$B_1 = \textbf{select } \lambda\,(?\overline{f});\ \textbf{break } L_{E_0}\ \texttt{[]}\ t\,(E_0, \text{true}) \textbf{ end select}$".

The reduction rules induced by LNT transition $(B\,(E_0^*, \mathit{true}),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_0'.E_0^*, \mathit{true}),\ \mathsf{N}_2)$ are the following:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.61)$$

$$\frac{\{t\,(E_0^*, \mathit{true})\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_0'.E_0^*, \mathit{true})\}\ \mathsf{N}_2 \quad (4.67)}{(B\,(E_0^*, \mathit{true}),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_0'.E_0^*, \mathit{true}),\ \mathsf{N}_2)} \quad (4.62)$$

By appeal to the definition of Figure 4.3, it follows directly that:

$$t\,(E_0'.E_0^*, \text{true}) \quad = \quad t\,(E_0', \text{true}).\ t\,(E_0^*, \text{true})$$

LNT behaviour $t\,(E_0')$ does not contain any command of the form "$\textbf{break } L_{E_0}$". Hence, $t\,(E_0'.E_0^*, \text{true})$ can be transformed as follows:

$$t\,(E_0'.E_0^*, \text{true}) = \textbf{trap } L \textbf{ in } t\,(E_0', \text{true}) \textbf{ else } t\,(E_0^*, \text{true}) \textbf{ end trap}$$

$$\frac{\{t(E_0, \mathit{true})\}\ \mathsf{N} \xrightarrow{2} \{t(E_0'.E_0^*, \mathit{true})\}\ \mathsf{N}_2}{\{B_1\}\ \mathsf{N} \xrightarrow{2} \{t(E_0'.E_0^*, \mathit{true})\}\ \mathsf{N}_2} \quad (4.63)$$

Note that, according to the standard definition of the **trap** construct, LNT behaviour $t\,(E_0', \text{true})$ is executed first. In the following, we stipulate that:

$$B_0' = \textbf{trap } L \textbf{ in } t\,(E_0', \text{true}) \textbf{ else } t\,(E_0^*, \text{true}) \textbf{ end trap}.$$

Hence, we may write that:

$$\frac{(4.63)}{\{\textbf{trap } L \textbf{ in } B_1 \textbf{ else } t\,(E_0^*, \text{true}) \textbf{ end trap}\}\ \mathsf{N} \xrightarrow{2} \{B_0'\}\ \mathsf{N}_2} \quad (4.64)$$

$$\frac{(4.64)}{\{t\,(E_0^*, \text{true})\}\ \mathsf{N} \xrightarrow{2} \{B_0'\}\ \mathsf{N}_2} \quad (4.65)$$

Replacing B_0' with $t(E_0', \mathit{true})$ in (4.65), we obtain the reduction rules induced by LNT transition $\{t\,(E_0, \mathit{true})\}\ \mathsf{N} \xrightarrow{\rho_2}_b \{t\,(E_0'.E_0^*, \mathit{true})\}$:

$$\frac{(4.64)}{\{t\,(E_0^*, \text{true})\}\ \mathsf{N} \xrightarrow{2} \{t(E_0'.E_0^*, \mathit{true})\}\ \mathsf{N}_2} \quad (4.66)$$

From Rule (4.66) and $\mathsf{M}' \approx \mathsf{N}'$, it follows directly that:

$$\langle (E_0'.E_0^*, \mathsf{M}'),\ (B\,(E_0'.E_0^*, \mathit{true}), \mathsf{N}') \rangle \in R.$$

Now, we consider tuple $\langle (E_0^*, \mathsf{M}),\ (B\,(E_0^*, \mathit{true}),\ \mathsf{N}) \rangle \in R$ and transition $(E_0^*, \mathsf{M}) \xrightarrow{\lambda}_1 (\surd,\ \mathsf{M})$. Then, we need to prove that if $\big(B\,(E_0^*, \mathit{true}),\ \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(\surd, \mathit{true}),\ \mathsf{N}'\big)$, where $\rho_2 = \lambda(\overline{f}_N)$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, it follows that $\langle (\surd, \mathsf{M}'),\ (B\,(\surd, \mathit{true}), \mathsf{N}') \rangle \in R$.

The following EB^3 reduction rule can be applied to EB^3 expression E_0^*:

$$\frac{}{(E_0^*, \mathsf{M}) \xrightarrow{\lambda}_1 (\surd,\ \mathsf{M})}$$

Transition $(B\,(E_0, \mathit{true}),\ \mathsf{N}) \rightsquigarrow_2 (B\,(\surd, \mathit{true}),\ \mathsf{N})$ induces the following set of reduction rules:

$$\{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N} \xrightarrow{2} \{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N_1} \tag{4.67}$$

$$\frac{\{t\,(E_0^*, \mathit{true})\}\ \mathsf{N} \xrightarrow{\lambda(\overline{f}_N)} \circ \xrightarrow{\mathbf{brk}(L_{E_0})}_b \{t\,(\surd, \mathit{true})\}\ \mathsf{N_2} \quad (4.67)}{(B\,(E_0^*, \mathit{true}),\ \mathsf{N}) \rightsquigarrow_2 (B\,(\surd, \mathit{true}),\ \mathsf{N}')} \tag{4.68}$$

where $\mathsf{N_1}$ and $\mathsf{N_2}$ are intermediate LNT memories, "$\xrightarrow{2} \doteq \xrightarrow{\lambda(\overline{f}_N)} \circ \xrightarrow{\mathbf{brk}(L_{E_0})}_b$" and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that rule (4.67) relates to the right construct of $B\,(E_0, \mathit{true})$, i.e. to memory process M.

By appeal to the definition of Figure 4.3, we may write that:

$$t\,(E_0{}^*, \mathrm{true}) \quad = \quad \mathbf{loop}\ L_{E_0}\ \mathbf{in}\ B_1\ \mathbf{end\ loop}$$

where "$B_1 = \mathbf{select}\ \lambda\,(?\overline{f});\ \mathbf{break}\ L_{E_0}$ [] $t\,(E_0, \mathrm{true})$ $\mathbf{end\ select}$".

$$\frac{(4.69) \quad (4.70)}{\{t\,(E_0^*, \mathit{true})\}\ \mathsf{N} \xrightarrow{\lambda(\overline{f}_N)} \circ \xrightarrow{\mathbf{brk}(L_{E_0})}_b \{t\,(\surd, \mathit{true}) \doteq \mathbf{stop}\}}$$

We stipulate that "$B_1' = \mathbf{trap}\ L\ \mathbf{in\ break}\ L_{E_0}\ \mathbf{else}\ t\,(E_0^*, \mathrm{true})\ \mathbf{end\ trap}$". Then, Rule (4.69) takes the following form:

$$\frac{\{\mathbf{trap}\ L\ \mathbf{in}\ B_1\ \mathbf{else}\ t\,(E_0^*, \mathrm{true})\ \mathbf{end\ trap}\}\ \mathsf{N} \xrightarrow{\lambda(\overline{f}_N)}_b \{B_1'\}\ \mathsf{N}}{\{t\,(E_0^*, \mathit{true})\}\ \mathsf{N} \xrightarrow{\lambda(\overline{f}_N)}_b \{B_1'\}\ \mathsf{N}} \tag{4.69}$$

and Rule (4.70) denoting breaking from the loop and termination is defined as follows:

$$\frac{\{\mathbf{break}\ L_{E_0}\}\ N \xrightarrow{\mathbf{brk}(L_{E_0})}_b \{\mathbf{stop}\}\ \mathsf{N}}{\{B_1'\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_b \{B\,(\surd, \mathit{true}) \doteq \mathbf{stop}\}\ \mathsf{N}} \tag{4.70}$$

From Rule (4.70) and $\mathsf{M} \approx \mathsf{N}$, it follows that:

$$\langle (\surd, \mathsf{M}\ (B\,(\surd, \mathit{true}), \mathsf{N}) \rangle \in R.$$

- Let $E = E_1 \;|[\Delta]|\; E_2$ and $ge = true$.

  Hence, we consider tuple $\langle (E_1 \;|[\Delta]|\; E_2, \mathsf{M}),\ (B\,(E_1 \;|[\Delta]|\; E_2, true),\ \mathsf{N})\rangle \in R$ and transition $(E_1 \;|[\Delta]|\; E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1' \;|[\Delta]|\; E_2,\ \mathsf{M}')$, where transition label $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c} \text{ is a vector of constants }\} \cup \{\lambda\}$.

  Then, we need to prove that if $\big(B\,(E_1 \;|[\Delta]|\; E_2, true),\ \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(E_1' \;|[\Delta]|\; E_2, true),\ \mathsf{N}'\big)$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N), j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, it follows that $\langle (E_1' \;|[\Delta]|\; E_2, \mathsf{M}'),\ (B\,(E_1' \;|[\Delta]|\; E_2, true), \mathsf{N}')\rangle \in R$.

  By appeal to the definition of Figure 4.3, it follows directly that:

$$t\,(E_1 \;|[\Delta]|\; E_2, \text{true}) \;=\; \textbf{par } \Delta \textbf{ in } t\,(E_1, \text{true}) \;||\; t\,(E_2, \text{true}) \textbf{ end par} \quad (4.71)$$

  The following EB^3 reduction rule can be applied to EB^3 expression $E_1 \;|[\Delta]|\; E_2$:

$$\frac{(E_1, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1',\ \mathsf{M}')}{(E_1 \;|[\Delta]|\; E_2,\ \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1' \;|[\Delta]|\; E_2,\ \mathsf{M}')} \quad \neg in(\rho_1, \Delta)$$

  Hence, transition $(B\,(E_1, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1', true),\ \mathsf{N})$ induces the following set of reduction rules:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.72)$$

$$\frac{\{t\,(E_1, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1', true)\}\ \mathsf{N}_2 \quad (4.72)}{(B\,(E_1', true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(\surd, true),\ \mathsf{N}')} \quad (4.73)$$

  where N_1, and N_2 are intermediate LNT memories such that "$\mathsf{N}' \doteq \mathsf{N} \oplus (\mathsf{N}_2 \ominus \mathsf{N}) \oplus (\mathsf{N}_1 \ominus \mathsf{N})$" and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that rule (4.72) relates to the right construct of $B\,(E_0, true)$, i.e. to memory process M. Then, it follows easily that:

$$\frac{\{t\,(E_1, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1', true)\}\ \mathsf{N}_2}{\{t\,(E_1 \;|[\Delta]|\; E_2, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E_1' \;|[\Delta]|\; E_2, true)\}\ \mathsf{N}_2} \quad (4.74)$$

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.75)$$

$$\frac{(4.74) \quad (4.75)}{(B\,(E_1 \;|[\Delta]|\; E_2, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E_1' \;|[\Delta]|\; E_2, true),\ \mathsf{N}')} \quad (4.76)$$

  as only construct $t\,(E_1, true)$ is affected. From Rule (4.76) and $\mathsf{M}' \approx \mathsf{N}'$, it follows directly that:

$$\langle (E_1' \;|[\Delta]|\; E_2, \mathsf{M}'),\ (B\,(E_1' \;|[\Delta]|\; E_2, true), \mathsf{N}')\rangle \in R.$$

  Now, transition $(E_1 \;|[\Delta]|\; E_2, \mathsf{M}) \xrightarrow{\rho_1}_1 (E_1' \;|[\Delta]|\; E_2',\ \mathsf{M}')$ is similar and will, therefore, not be treated.

  Unfortunately, transition $(\surd \;|[\Delta]|\; \surd, \mathsf{M}) \xrightarrow{\lambda}_1 (\surd,\ \mathsf{M})$ inducing the following trivial EB^3 reduction rule:

$$\frac{}{(\surd \;|[\Delta]|\; \surd, \mathsf{M}) \xrightarrow{\lambda}_1 (\surd,\ \mathsf{M})}$$

cannot be simulated by the corresponding LNT reduction rule:

$$\frac{\{B\,(\surd, true) \doteq \mathbf{null}\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_b \{\mathbf{stop}\}\ \mathsf{N} \quad \{B\,(\surd, true) \doteq \mathbf{null}\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_b \{\mathbf{stop}\}\ \mathsf{N}}{(B\,(\surd\ \mathsf{I[\Delta]I}\ \surd, true),\ \mathsf{N}) \xrightarrow{\mathbf{exit}}_b (\mathbf{stop} \not\doteq B\,(\surd, true),\ \mathsf{N})} \quad (4.77)$$

The reason lies in the fact that the inert action λ used to denote execution of EB^3 process $\surd\ \mathsf{I[\Delta]I}\ \surd$ cannot be matched to LNT communication label $\xrightarrow{\mathbf{exit}}_b$, as if we assume that "$B\,(\surd, true) \doteq \mathbf{null}$", no transition of the form "$(B\,(\surd\ \mathsf{I[\Delta]I}\ \surd, true),\ \mathsf{N}) \xrightarrow{\mathbf{exit}}_b (B\,(\surd, true),\ \mathsf{N})$" can take place in rule (4.77). However, bisimilarity between TS_{EB^3} and TS_{LNT} is restored if we remove the inert transition $(\surd\ \mathsf{I[\Delta]I}\ \surd, \mathsf{M}) \xrightarrow{\lambda}_1 (\surd,\ \mathsf{M})$ from the set of EB^3 reduction rules, while we consider that $(\surd\ \mathsf{I[\Delta]I}\ \surd)$ is an equivalent form of $\surd$.

- Let $E = P\,(\overline{v})$. From Figure 4.3, it follows that $ge = \text{true}$.

  Hence, we consider tuple $\langle (P\,(\overline{v}), \mathsf{M}),\ (B\,(P\,(\overline{v}), true),\ \mathsf{N})\rangle \in R$. Let also transition $(P\,(\overline{v}), \mathsf{M}) \xrightarrow{\rho_1}_1 (E',\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c}$ is a vector of constants $\} \cup \{\lambda\}$. Then, we need to prove that if $\big(B\,(P\,(\overline{v}), true),\ \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(E', true),\ \mathsf{N}'\big)$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\ j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N as usual, it follows that $\langle (E', \mathsf{M}'),\ (B\,(E', true), \mathsf{N}')\rangle \in R$.

  By appeal to the definition of Figure 4.3, it follows directly that:

$$t\,(P\,(\overline{v}), \text{true}) \quad = \quad P\,[\alpha_1, \ldots, \alpha_q, \lambda]\,(\overline{v}) \qquad (4.78)$$

  The following EB^3 reduction rule can be applied to EB^3 expression $P\,(\overline{v})$:

$$\frac{(E[\overline{x} := \overline{v}], \mathsf{M}) \xrightarrow{\rho_1} (E', \mathsf{M})}{(P\,(\overline{v}), \mathsf{M}) \xrightarrow{\rho_1} (E', \mathsf{M}')} \quad P\,(\overline{x}) = E$$

  By way of the induction principle, transition $(B\,(E[\overline{x} := \overline{v}], true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E', true),\ \mathsf{N})$ induces the following set of reduction rules:

$$\{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N}_1 \quad (4.79)$$

$$\frac{\{t\,(E[\overline{x} := \overline{v}], true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E', true)\}\ \mathsf{N}_2 \quad (4.83)}{(B\,(E[\overline{x} := \overline{v}], true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E', true),\ \mathsf{N}')} \quad (4.80)$$

  where N_1, and N_2 are intermediate LNT memories such that "$\mathsf{N}' \doteq \mathsf{N} \oplus (\mathsf{N}_2 \ominus \mathsf{N}) \oplus (\mathsf{N}_1 \ominus \mathsf{N})$" and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that rule (4.83) relates to the right construct of $B\,(E[\overline{x} := \overline{v}], true)$, i.e. to memory process M. Then, by replacing $E[\overline{x} := \overline{v}]$ with $P\,(\overline{v})$ in (4.84), we obtain the following reduction rules:

$$\{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\mathbf{loop}\ \overline{g} := \overline{f};\ D\ \mathbf{end\ loop}\}\ \mathsf{N}_1 \quad (4.81)$$

$$\frac{\{t\,(P\,(\overline{v}), true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E', true)\}\ \mathsf{N}_2 \quad (4.81)}{(B\,(P\,(\overline{v}), true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E', true),\ \mathsf{N}')} \quad (4.82)$$

  From Rule (4.82) and $\mathsf{M}' \approx \mathsf{N}'$, it follows directly that:

$$\langle (E', \mathsf{M}'),\ (B\,(E', true), \mathsf{N}')\rangle \in R.$$

- In all other cases, for which ge does not use *attribute functions*, the proof goes as follows:

  We consider tuple $\langle (ge \Rightarrow E, \mathsf{M}),\ (B\,(E, ge),\ \mathsf{N})\rangle \in R$. Let also transition $(ge \Rightarrow E, \mathsf{M}) \xrightarrow{\rho_1}_1 (E',\ \mathsf{M}')$, where $\rho_1 \in \{\alpha_j\,(\overline{c}),\ j \in 1..q \mid \overline{c}$ is a vector of constants $\} \cup \{\lambda\}$.

  Then, we need to prove that if $\big(B\,(ge \Rightarrow E, true),\ \mathsf{N}\big) \rightsquigarrow_2 \big(B\,(E', true),\ \mathsf{N}'\big)$, where $\rho_2 \in \{\alpha_j\,(\overline{c}, \overline{f}_N),\ j \in 1..q\} \cup \{\lambda(\overline{f}_N)\}$ and $\overline{f}_N$ is the value assigned to $\overline{f}$ by N, it follows that $\langle (E', \mathsf{M}'),\ (B\,(E', true), \mathsf{N}')\rangle \in R$.

  By way of the induction principle, transition $(B\,(E, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E', true),\ \mathsf{N})$ induces the following set of reduction rules:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.83)$$

$$\frac{\{t\,(E, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E', true)\}\ \mathsf{N}_2 \quad (4.83)}{(B\,(E, true),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E', true),\ \mathsf{N}')} \quad (4.84)$$

  where N_1, and N_2 are intermediate LNT memories such that "$\mathsf{N}' \doteq \mathsf{N} \oplus (\mathsf{N}_2 \ominus \mathsf{N}) \oplus (\mathsf{N}_1 \ominus \mathsf{N})$" and $\mathsf{M}' \approx \mathsf{N}'$ by the inductive hypothesis. Note that rule (4.83) relates to the right construct of $B\,(E[\overline{x} := \overline{v}], true)$, i.e. to memory process M.

  – By appeal to the definition of Figure 4.3, it follows directly that:

$$t\ (E, ge) = \textbf{if } mod\,(ge) \textbf{ then } t\,(E, \text{true}) \textbf{ else stop end if}$$

    The following EB^3 reduction rule can be applied to EB^3 expression $ge \Rightarrow E$:

$$\frac{(E, \mathsf{M}) \xrightarrow{\rho_1} (E', \mathsf{M}') \quad [\![ge]\!]_3^{\mathsf{M}}\ (\mathsf{M}) = \mathsf{true}}{(ge \Rightarrow E, \mathsf{M}) \xrightarrow{\rho_1} (E', \mathsf{M}')}$$

    We recall that evaluation "$[\![ge]\!]_3^{\mathsf{M}}\ (\mathsf{M}) = \mathsf{true}$" is, by force of Theorem 4.6.3, equivalent to "$\{mod\,(ge)\}\ \mathsf{N} \rightarrow_e \mathsf{true}$" that will be used later.

    Then, by slightly transforming (4.84), we obtain the following reduction rules:

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N}_1 \quad (4.85)$$

$$\frac{\{t\,(E, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E', true)\}\ \mathsf{N}_2 \quad \{mod\,(ge)\}\ \mathsf{N} \rightarrow_e \mathsf{true}}{\{\textbf{if } mod\,(ge) \textbf{ then } t\,(E, \text{true}) \textbf{ else stop end if}\}\ \mathsf{N} \xrightarrow{2} \{t\,(E', true)\}\ \mathsf{N}_2} \quad (4.86)$$

$$\frac{(4.86) \quad (4.85)}{(B\,(E, ge),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E', true),\ \mathsf{N}')} \quad (4.87)$$

    From Rule (4.87) and $\mathsf{M}' \approx \mathsf{N}'$, it follows directly that:

$$\langle (E', \mathsf{M}'),\ (B\,(E', true), \mathsf{N}')\rangle \in R.$$

  – If "$[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true}$", then TS_{EB^3} blocks. Similarly, "$t(E, ge) = \textbf{stop}$" and TS_{LNT} blocks too.

  – If ge uses attribute functions, then the proof goes as follows: By appeal to the definition of Figure 4.3, it follows directly that:

$$\begin{aligned} t\ (E, ge) \ = \ & \textbf{par } \alpha_1, \ldots, \alpha_q, \lambda \textbf{ in} \\ & \quad t\,(E, \text{true}) \ \|\ pr_{ge}\ [\alpha_1, \ldots, \alpha_q, \lambda]\ (vars\,(ge)) \\ & \textbf{end par} \end{aligned} \quad (4.88)$$

Rule (4.89) denotes synchronization on communication label $\alpha_j\,(\overline{x}_N, \overline{f}_N)$ between the left and right construct of $t(E, ge)$ for some $j \in \{1, \ldots, q\}$. Moreover, Rule (4.90) is part of the induction principle that remains unchanged. Now, let the following definitions:

$$\begin{aligned} \mathsf{N_3} &\doteq \mathsf{N} \oplus [\mathsf{start} \leftarrow \mathsf{false}] \\ \mathsf{N_4} &\doteq \mathsf{N'} \oplus [\mathsf{start} \leftarrow \mathsf{false}] \end{aligned}$$

Hence, we obtain the following reduction rules:

$$\frac{\{t\,(E, true)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E', true)\}\ \mathsf{N_2} \quad (4.94)}{\{t\,(E, ge)\}\ \mathsf{N} \xrightarrow{2} \{t\,(E', true)\}\ \mathsf{N_2}} \tag{4.89}$$

$$\{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N} \xrightarrow{1} \circ \xrightarrow{3} \{\textbf{loop } \overline{g} := \overline{f};\ D \textbf{ end loop}\}\ \mathsf{N_1} \tag{4.90}$$

$$\frac{(4.89) \quad (4.90)}{(B\,(E, ge),\ \mathsf{N}) \rightsquigarrow_2 (B\,(E', true),\ \mathsf{N_3})} \tag{4.91}$$

Let also the following LNT behaviours:

$B_1 \doteq$ **select**
  $\alpha_1\,(?\overline{x}, ?\overline{f})$ **where** $mod\,(ge)$ [] ... [] $\alpha_q\,(?\overline{x}, ?\overline{f})$ **where** $mod\,(ge)$
end select

$B_2 \doteq$ **select** $\alpha_1\,(?\overline{x}, ?\overline{f})$ [] ... [] $\alpha_1\,(?\overline{x}, ?\overline{f})$ **end select**

$B_3 \doteq$ **select**
  if *start* **then** *start* := false; B_1 **else** B_2 **end if** [] **break** L
end select

$B_4 \doteq$ **loop** L **in** B_3 **end loop**

Hence, the definition of pr_C can be modified as follows:

process pr_C [$\alpha_1, \ldots, \alpha_q, \lambda$: **any**] ($vars\,(C)$: $type\,(vars\,(C))$) **is**
var *start* : **bool**, $\overline{x} : \overline{T}_{ac}$, $\overline{f} : \overline{T}$ **in**
  start := true; B_4
end var
end process

Let also the following definitions:

$$\begin{aligned} B_5 &\doteq \textbf{trap } L \textbf{ in } B_3 \textbf{ else } B_4 \textbf{ end trap} \\ B_6 &\doteq \textbf{trap } L \textbf{ in } B_1 \textbf{ else } B_4 \textbf{ end trap} \\ B_7 &\doteq \textbf{trap } L \textbf{ in null else } B_4 \textbf{ end trap} \end{aligned}$$

Rule (4.94) denotes reduction of pr_{ge} [$\alpha_1, \ldots, \alpha_q, \lambda$] ($vars\,(ge)$). In particular, variable *start* is initially set to *true*, then changes to *false*, $mod(ge)$ is evaluated to

true, and communication label $\alpha_j\,(\overline{x}_N, \overline{f}_N)$ is executed.

$$\frac{\{B_4\}\ \mathsf{N} \oplus [\mathsf{start} \leftarrow \mathsf{true}] \xrightarrow{\mathbf{exit}}_b \{B_1\}\ \mathsf{N_3}}{\{B_5\}\ \mathsf{N} \oplus [\mathsf{start} \leftarrow \mathsf{true}] \xrightarrow{\mathbf{exit}}_b \{B_6\}\ \mathsf{N_3}} \tag{4.92}$$

$$\frac{(4.96)}{\{B_6\}\ \mathsf{N_3} \xrightarrow{\alpha_j\,(\overline{x}_N, \overline{f}_N)}_b \{B_7\}\ \mathsf{N_3}} \tag{4.93}$$

$$\frac{\{start := true\}\ \mathsf{N} \xrightarrow{\mathbf{exit}}_s \mathsf{N} \oplus [\mathsf{start} \leftarrow \mathsf{true}] \quad (4.92) \quad (4.93)}{\{pr_{ge}\ [\alpha_1, \ldots, \alpha_q, \lambda]\ (vars\,(ge))\}\ \mathsf{N} \xrightarrow{2} \{B_7\}\ \mathsf{N_3}}, \tag{4.94}$$

where "$\xrightarrow{2}\!\doteq\!\xrightarrow{\mathbf{exit}}_b \circ \xrightarrow{\alpha_j\,(\overline{x}_N, \overline{f}_N)}_b$". Note that, by way of the standard semantics regarding the **trap** construct, B_7 can be substituted by B_4. Moreover, Rule (4.96) unfolds the following set of rules:

$$\frac{\{mod\,(ge)\}\ \mathsf{N_3} \rightarrow_e \mathsf{true} \quad \{\alpha_j\,(?\overline{x}, ?\overline{f})\}\ \mathsf{N_3} \xrightarrow{\alpha_j\,(\overline{x}_N, \overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N_3}}{\{\alpha_j\,(?\overline{x}, ?\overline{f})\ \mathbf{where}\ mod\,(ge)\}\ \mathsf{N_3} \xrightarrow{\alpha_j\,(\overline{x}_N, \overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N_3}} \tag{4.95}$$

$$\frac{(4.95)}{\{B_1\}\ \mathsf{N_3} \xrightarrow{\alpha_j\,(\overline{x}_N, \overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N_3}} \tag{4.96}$$

Note that evaluation "$[\![ge]\!]_3^{\mathsf{M}}(\mathsf{M}) = \mathsf{true}$" is, by force of Theorem 4.6.3, equivalent to "$\{mod\,(ge)\}\ \mathsf{N_3} \rightarrow_e \mathsf{true} \doteq \{mod\,(ge)\}\ \mathsf{N} \rightarrow_e \mathsf{true}$", since variable *start* cannot appear syntactically in *mod (ge)*. Furthermore, Rule (4.96) has no effect on LNT memory $\mathsf{N_3}$, as $\overline{x}_N$ and $\overline{f}_N$ are the respective values assigned by LNT memory N to vectors $\overline{x}$ and $\overline{f}$. From Rule (4.96), $\mathsf{M'} \approx \mathsf{N'}$, and $\mathsf{N_4} = \mathsf{N'} \oplus [\mathsf{start} \leftarrow \mathsf{false}]$, it follows directly that:

$$\langle (E', \mathsf{M'}),\ (B\,(E', true), \mathsf{N_4}) \rangle \in R.$$

If communication label $\alpha_j\,(\overline{x}_N, \overline{f}_N)$ is substituted by $\lambda(\overline{f}_N)$, the proof follows similar lines.

Similarly, if *start* is equal to *false*, then Rule (4.96) can be modified as follows:

$$\frac{\alpha_j\,(?\overline{x}, ?\overline{f})\}\ \mathsf{N_3} \xrightarrow{\alpha_j\,(\overline{x}_N, \overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N_3}}{\{B_1\}\ \mathsf{N_3} \xrightarrow{\alpha_j\,(\overline{x}_N, \overline{f}_N)}_b \{\mathbf{null}\}\ \mathsf{N_3}} \tag{4.97}$$

and the rest of the proof remains unchanged.

□

4.7 Conclusion

We proposed an approach for equipping the EB^3 method with formal verification capabilities by reusing already available model checking technology. Our approach relies upon a new translation from EB^3 to LNT, which provides a direct connection to all the state-of-the-art

verification features of the CADP toolbox. The translation, based on alternative memory semantics of EB^3 [VD13] instead of the original trace semantics [FSt03], was automated by the EB^{3}2LNT translator and validated on several examples of typical ISs. So far, we experimented only the model checking of MCL data-based temporal properties on EB^3 specifications. However, CADP also provides extensive support for equivalence checking and compositional LTS construction, which can be of interest to IS designers. We also provided a partial formal proof of the translation from EB^3 to LNT, which could serve as reference for translating EB^3 to other process algebras as well.

As future work, we plan to study abstraction techniques for verifying properties regardless of the number of entity instances that participate in the IS, following the approaches for parameterized model checking [ABJ$^+$99]. In particular, we will observe how the insertion of new functionalities into an IS affects this issue, and we will formalize this in the context of EB^3 specifications.

Chapter 5

Verification of Temporal Properties

5.1 Model Checking Language

In this section, we define the syntax and the formal semantics of the Model Checking Language (MCL) [MT08]. MCL is an extension of the alternation-free modal μ-calculus [Eme86] with action predicates enabling value extraction, modalities containing extended regular expressions on transition sequences, quantified variables and parameterized fixed point operators, programming language constructs, and fairness operators encoding generalized Büchi automata. These features make possible a concise and intuitive description of safety, liveness, and fairness properties involving data, without sacrificing the efficiency of on-the-fly model checking, which has a linear-time complexity for the dataless MCL formulas [MT08].

In particular, MCL consists of data expressions e, action formulas α and state formulas ϕ. Let a set of data variables $\mathcal{X}$ and a set of function identifiers $\mathcal{F}$ with standard interpretation. Data expressions e are defined as follows:

$$e \quad ::= \quad x \quad | \quad f(e_1, \ldots, e_n),$$

where $x \in \mathcal{X}$ and $f \in \mathcal{F}$ denoting typed function identifiers.

Action formulas α consist of:

- action patterns of the form $\{c\ !e_1\ \ldots\ !e_n\}$ such that, if action $\{c\ v_1\ \ldots\ v_n\}$ occurs, v_k matches expression e_k for $k \in \{1, \ldots, n\}$,
- action patterns of the form $\{c\ ?x_1 : T_1\ \ldots\ ?x_n : T_n\}$[1] such that, if action $\{c\ v_1\ \ldots\ v_n\}$ occurs, $\mathsf{v_k}$ is assigned to variable $\mathsf{x_k}$ for $k \in \{1, \ldots, n\}$, and
- usual Boolean operators.

To sum up, the abstract syntax of action formulas α is given by the following grammar:

$$\alpha \quad ::= \quad \{c\ !e_1\ \ldots\ !e_n\} \quad | \quad \{c\ ?x_1 : T_1\ \ldots\ ?x_n : T_n\} \quad | \quad \neg\alpha \quad | \quad \alpha_1 \wedge \alpha_2 \quad | \quad \alpha^*$$

Let propositional variables $Y \in \mathcal{Y}$ that denote functions $F : T_1 \times \ldots \times T_n \to 2^S \in \mathcal{F}$, where $T_1, \ldots, T_n$ are domains and S is the state space. State formulas consist of data expressions, propositional variables $Y \in \mathcal{Y}$, boolean operators, quantifiers and fixed point operators.

[1] T_k denotes x_k's type as well as the corresponding domain of elements that can be assigned to x_k

Note also that fixed point operations fall within the scope of an even number of negations (MCL formulas $\mu(\ldots)Y.\phi$ and $\nu(\ldots)Y.\phi$ are syntactically monotonic) [Koz83]. For the efficiency of model checking, mutual recursion between minimal and maximal fixed point operators is not allowed [EL$^+$86].

We assign values to propositional variables $x \in \mathcal{X}$ appearing in expressions e and action formulas α that are denoted as $\delta : \mathcal{X} \to T_1 \cup \ldots \cup T_n$. Moreover, we assign functions to all free propositional variables Y appearing in MCL state formulas that are denoted as $\rho : \mathcal{Y} \to \mathcal{F}$. Notations "$\mu Y.\phi$" and "$\nu Y.\phi$" denote the corresponding fixed points of monotonic functions over $Y \to 2^S$.

Other useful operators like **if** and **case** can also be used to construct MCL formulas. Their definition is found in [MT08].

Action patterns are enriched with additional features such as the wilcard clause **any** that matches any value and the "**where** V" clause denoting that the pattern-matching takes place on condition that condition V evaluates to true. Remark that parameters $?x : T$ can appear syntactically in V.

The abstract syntax of MCL state formulas is given by the following grammar:

$$
\begin{array}{rcl}
\phi & ::= & e \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \langle\, \alpha \,\rangle\, \phi \mid \mu Y.\ \phi \\
& \mid & \textbf{exists } x_1 : T_1, \ldots, x_n : T_n.\ \phi \mid Y(e_1, \ldots, e_n) \\
& \mid & \textbf{let } x_1 : T_1 := e_1, \ldots, x_n : T_n := e_n \textbf{ in } \phi \textbf{ end let} \\
& \mid & \textbf{if } \phi_1 \textbf{ then } \phi_1' \textbf{ elsif } \phi_2 \textbf{ then } \phi_2' \textbf{ else } \phi_3' \textbf{ end if} \\
& \mid & \textbf{case } e \textbf{ is } p_1 \to \phi_1 \mid \ldots \mid p_n \to \phi_n \textbf{ end case}
\end{array}
$$

The necessity modality is the dual of possibility modality "$[\,\alpha\,]\ \phi = \neg\langle\,\alpha\,\rangle\ \neg\phi$". The maximal fixed point operator is the dual of the minimal fixed point operator: "$\nu(\ldots)Y.\phi = \neg\mu(\ldots)Y.\neg\phi[Y/\neg Y]$", where $[Y/\neg Y]$ denotes the syntactic substitution of Y by $\neg Y$. Note that "$\langle\, \alpha^* \,\rangle\ \phi$" is syntactic sugar for "$\mu Y(\phi \vee \langle\, \beta \,\rangle\ Y)$".

The semantics of expressions, action formulas and state formulas is given below:

Expressions

$$
\begin{array}{rcl}
[\![x]\!]\delta & = & \delta(x) \\
[\![f(e_1, \ldots, e_n)]\!]\delta & = & \mathsf{f}([\![e_1]\!]\delta, \ldots, [\![e_n]\!]\delta)
\end{array}
$$

Action formulas

$$
\begin{array}{rcl}
[\![\{c\ !e_1 \ldots !e_n\}]\!]\delta & = & \{c[\![e_1]\!]\delta \ldots [\![e_n]\!]\delta\}\} \\
[\![\{c?x_1 : T_1 \ldots ?x_n : T_n\}]\!]\delta & = & \{cv_1 \ldots v_n \mid \forall 1 \leq i \leq n,\ v_i \in D_i\} \\
[\![\neg\alpha]\!]\delta & = & A \setminus [\![\alpha]\!]\delta \\
[\![\alpha_1 \vee \alpha_2]\!]\delta & = & [\![\alpha_1]\!]\delta \cup [\![\alpha_2]\!]\delta \\
env_\alpha(\{c\ ?x_1 : T_1 \ldots ?x_n : T_n\})\delta & = & [\mathsf{x_1} \leftarrow \mathsf{v_1}, \ldots, \mathsf{x_n} \leftarrow \mathsf{v_n}], \text{where } \alpha \doteq c\ v_1 \ldots v_n \\
env_\alpha(\{c\ ?x_1 : T_1 \ldots ?x_n : T_n\})\delta & = & [\,], \quad \text{otherwise}
\end{array}
$$

State formulas

$$
\begin{array}{rcl}
[\![e]\!]\rho\ \delta & = & \{s \in S \mid [\![e]\!]\delta\} \\
[\![\neg\phi]\!]\rho\ \delta & = & S \setminus [\![\phi]\!]\rho\ \delta \\
[\![\phi_1 \vee \phi_2]\!]\rho\ \delta & = & [\![\alpha_1]\!]\rho\ \delta \cup [\![\alpha_2]\!]\rho\ \delta \\
[\![\langle\, \alpha \,\rangle\ \phi]\!]\rho\ \delta & = & \{s \in S \mid \exists s \xrightarrow{\alpha} s'.\ a \in [\![\alpha]\!]\delta\ \wedge\ s' \in [\![\phi]\!]\rho\ \big(\delta \oplus env_a(\alpha)\big)\} \\
[\![\textbf{exists}\ x_1 : T_1, \ldots, x_n : T_n.\ \phi]\!]\rho\ \delta & = & \{s \in S \mid \exists v_1 : T_1, \ldots, v_n : T_n.\ a \in [\![\alpha]\!]\delta\ \wedge \\
 & & \qquad s \in [\![\phi]\!]\rho\ \big(\delta \oplus [\mathsf{x_1} \leftarrow \mathsf{v_1}, \ldots, \mathsf{x_n} \leftarrow \mathsf{v_n}]\big)\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![Y(e_1, \ldots, e_n)]\!]\rho\ \delta & = & \rho(Y)([\![e_1]\!]\delta, \ldots, [\![e_n]\!]\delta) \\
[\![\mu Y.\ \phi]\!]\rho\ \delta & = & \bigcap\{\ T \subseteq S \mid T \subseteq [\![\phi]\!](\rho \oplus [\mathsf{Y} \leftarrow \mathsf{T}])\delta\ \} \\
[\![\textbf{let}\ x_1 : T_1 := e_1, \ldots, x_n : T_n := e_n\ \textbf{in} & & \\
\quad \phi\ \textbf{end let}]\!]\rho\ \delta & = & [\![\phi]\!]\rho\ (\delta \oplus [\mathsf{x_1} \leftarrow [\![e_1]\!]\delta, \ldots, \mathsf{x_n} \leftarrow [\![e_n]\!]\delta),
\end{array}
$$

where $\subseteq$ is the set inclusion operator. A state s satisfies a formula ϕ denoted as $s \models \phi$ if and only if $s \in [\![\phi]\!][\ ][\ ]$. An LTS $M = \langle\, S, A, T, s_0 \,\rangle$ satisfies a formula ϕ denoted as $M \models \phi$ if and only if $s_0 \models \phi$.

MCL's expressiveness is not limited to *safety* and *liveness* properties. Deadlock freedom is formalized as "$[\, true^* \,]\, \langle\, true \,\rangle\ true$". The fair reachability [QS83] of an action a is formalized as "$[\, (\neg a)^* \,]\, \langle\, (\neg a)^*.a \,\rangle\ true$". For more complex fairness properties, we use the infinite looping operator $\Delta\beta$ of *PDL*-Δ [Str82], denoted as $\beta@$ in MCL, which states the existence of an infinite (unfair) sequence made by concatenating subsequences satisfying β.

5.2 Library Management System Revisited

We illustrate below the application of the EB^{3}2LNT translator in conjunction with CADP for analyzing an extended version of the IS library management system, whose description in EB^3 can be found in Annex C of [Ger06]. With respect to the simplified version presented in Chapter 2, the IS enables e.g., members to renew their loans and to reserve books, and their reservations to be cancelled or transferred to other members on demand. The desired behaviour of this IS was characterized in [FFC$^+$10] as a set of 15 requirements expressed informally as follows:

R1. A book can always be acquired by the library when it is not currently acquired.

R2. A book cannot be acquired by the library if it is already acquired.

R3. An acquired book can be discarded only if it is neither borrowed nor reserved.

R4. A person must be a member of the library in order to borrow a book.

R5. A book can be reserved only if it has been borrowed or already reserved by some member.

R6. A book cannot be reserved by the member who is borrowing it.

R7. A book cannot be reserved by a member who is reserving it.

R8. A book cannot be lent to a member if it is reserved.

R9. A member cannot renew a loan or give the book to another member if the book is reserved.

R10. A member is allowed to take a reserved book only if he owns the oldest reservation.

R11. A book can be taken only if it is not borrowed.

R12. A member who has reserved a book can cancel the reservation at anytime before he takes it.

R13. A member can relinquish library membership only when all his loans have been returned and all his reservations have either been used or cancelled.

R14. Ultimately, there is always a procedure that enables a member to leave the library.

R15. A member cannot borrow more than the loan limit defined at the system level for all users.

5.2.1 MCL Formulas for Requirements R1 to R15

We expressed all the above requirements using the property specification language MCL [MT08]. MCL is an extension of the alternation-free modal μ-calculus [Eme86] with action predicates enabling value extraction, modalities containing extended regular expressions on transition sequences, quantified variables and parameterized fixed point operators, programming language constructs, and fairness operators encoding generalized Büchi automata. These features make possible a concise and intuitive description of safety, liveness, and fairness properties involving data, without sacrificing the efficiency of on-the-fly model checking, which has a linear-time complexity for the dataless MCL formulas [MT08].

We show below the MCL formulation of two requirements from the list above, which denote typical safety and liveness properties. Requirement R2 is expressed in MCL as follows:

$$[\,\mathbf{true}^*.\{\texttt{ACQUIRE ?}B : \mathbf{string}\}.(\mathbf{not}\ \{\texttt{DISCARD !}B\})^*.\{\texttt{ACQUIRE !}B\}\,]\ \mathbf{false}$$

This formula uses the standard *safety* pattern "$[\beta]$ **false**", which forbids the existence of transition sequences matching the regular formula β. Here the undesirable sequences are those containing two *Acquire* operations for the same book B without a *Discard* operation for B in the meantime. The regular formula $\mathbf{true}^*$ matches a subsequence of (zero or more) transitions labeled by arbitrary actions. Note the use of the construct "$\texttt{?}B : \mathbf{string}$", which matches any string and extracts its value in the variable B used later in the formula. Therefore, the above formula captures all occurrences of books carried by *Acquire* operations in the model. Requirement R12 is formulated in MCL as follows:

$$\begin{array}{l}[\,\mathbf{true}^*.\{\texttt{RESERVE ?}M : \mathbf{string}\ \texttt{?}B : \mathbf{string}\}.\\ \ (\mathbf{not}\ (\{\texttt{TAKE !}M\ \texttt{!}B\}\ \mathbf{or}\ \{\texttt{TRANSFER !}M\ \texttt{!}B\}))^*\,]\\ \quad \langle\,(\mathbf{not}\ (\{\texttt{TAKE !}M\ \texttt{!}B\}\ \mathbf{or}\ \{\texttt{TRANSFER !}M\ \texttt{!}B\}))^*.\ \{\texttt{CANCEL !}M\ \texttt{!}B\}\,\rangle\ \mathbf{true}\end{array}$$

This formula denotes a *liveness* property of the form "$[\beta_1]\ \langle\beta_2\rangle$ **true**", which states that every transition sequence matching the regular formula β_1 (in this case, book B has been reserved by member M and subsequently neither taken nor transferred) ends in a state from

which there exists a transition sequence matching the regular formula β_2 (in this case, the reservation can be cancelled before being taken or transferred).

Requirement R7 can be formulated as follows:

[**true*** . {`RESERVE ?`M : **string** `?`B : **string**} . (**not** ({`TAKE !`M `!`B} **or** {`CANCEL !`M `!`B}))* . {`RESERVE !`M `!`B}] **false**

5.2.2 Verification of the Library Management System

Using EB^{3}2LNT, we translated the EB^3 specification of the library management system to LNT. The resulting specification was checked against all the 15 requirements, formulated in MCL, using the EVALUATOR 4.0 model checker of CADP. The experiments were performed on an Intel(R) Core(TM) i7 CPU 880 at 3.07GHz. Table 5.1 shows the results for several configurations of the IS for *NbLoans* = 1, obtained by instantiating the number of books (m) and members (p) in the IS. All requirements were shown to be valid on the IS specification. The second and third line of the table indicate the number of states and transitions of the LTS corresponding to the LNT specification. The fourth line gives the time needed to generate the LTS and the other lines give the verification time for each requirement. Note that the number of states generated increases with the size of m and p as EVALUATOR 4.0 applies explicit techniques for state space generation.

Comparison with [FFC$^+$10] We recall that CADP was used in [FFC$^+$10] for the verification of a version of the library management system with similar functionalities. According to the same paper, the time needed to generate the corresponding LTS for *NbLoans* = 1, was 970.2 sec approximately, the average time needed to verify the system requirements was 74.63 sec and no information is provided for the size of the LTS, which means that no reliable comparison of the two works is possible. Note also that the LNT specification, to whom the results of Table 5.1 are related, was automatically generated by EB^{3}2LNT, whereas the results of [FFC$^+$10] correspond to a manual LNT specification of the library management system.

5.3 File Transfer System Revisited

We express in MCL the system requirements related to the simplified file transfer system of Chapter 4 and model-check them with CADP.

5.3.1 MCL Formulas for Requirements P1 to P6

Requirement P1 "*A client C sends a file transfer request* `"Rqt !`F `!`C`"` *to the server interface regarding file F. Ultimately, if file F is stocked in server S, there is always a procedure that forwards the previous request to server S* `"Fwd !`F `!`C`"`. "

Table 5.1: Model checking results for the library management system

(m, p)	(3,2)	(3,3)	(3,4)	(4,3)
states	1,002	182,266	8,269,754	27,204,016
trans.	5,732	1,782,348	105,481,364	330,988,232
time	1.9s	14.4s	31'39s	140'22s
R1	0.3s	1.8s	5'19s	20'13s
R2	0.2s	2.9s	9'26s	36'7s
R3	0.2s	9.4s	97'46s	26'47s
R4	0.2s	1.7s	5'15s	18'40s
R5	0.2s	2.2s	6'46s	21'52s
R6	0.2s	4.1s	38'30s	10'19s
R7	0.2s	7.4s	65'22s	24'33s
R8	0.2s	2.2s	6'52s	22'27s
R9	0.2s	2.3s	6'38s	22'29s
R10	0.3s	13.3s	43'59s	62'07s
R11	0.3s	2.5s	6'36s	22'14s
R12	0.3s	4.0s	10'47s	45'09s
R13	0.4s	4.3s	11'46s	36'07s
R14	0.3s	3.6s	10'41s	37'33s
R15	0.2s	2.8s	7'53s	28'56s

macro $R_1\,(F, C) =$
  (
    $[\,(\textbf{not}\ (\{\texttt{Chown !}F\ \texttt{?any}\}\ \textbf{or}\ \{\texttt{Rqt !}F\ \texttt{?any}))^*.\ \{\texttt{Rqt !}F\ \texttt{!}C\}\,]$
    $\langle\,(\textbf{not}\ (\{\texttt{Fwd !}F\ \texttt{?any}))^*\ .\ \{\texttt{Fwd !}F\ \texttt{?}S : \textbf{string}$
      where $(((F = \text{"}F_1\text{"})\ \textbf{and}\ (S = \text{"}S_1\text{"}))\ \textbf{or}\ ((F = \text{"}F_2\text{"})\ \textbf{and}\ (S = \text{"}S_2\text{"})))\}\,\rangle\ \textbf{true}$
    and
    $[\,\textbf{true}^*\,.\,\{\texttt{Chown !}F\ \texttt{?}S : \textbf{string}\}\,.\,(\textbf{not}\ (\{\texttt{Chown !}F\ \texttt{?any}\}\ \textbf{or}\ \{\texttt{Rqt !}F\ \texttt{!}C\}))^*.$
     $\{\texttt{Rqt !}F\ \texttt{!}C\}\,]\,\langle\,\{\textbf{not}\ (\{\texttt{Fwd !}F\ \texttt{?any}))^*\ .\ \texttt{Fwd !}F\ \texttt{!}C\ \texttt{!}S\}\,\rangle\ \textbf{true}$
)
end_macro
$P_1(\texttt{"}F_1\texttt{"}, \texttt{"}C_1\texttt{"})\ \textbf{and}\ P_1(\texttt{"}F_2\texttt{"}, \texttt{"}C_2\texttt{"})$

We assume that file F_1 is initially stocked in server S_1 and that file F_2 is initially stocked in server S_2. Hence, the first conjunct in formula R_1 is a classical liveness property expressing the eventuality that the file transfer request reaches the server in question provided that the file does not change owner in the meantime. The second conjunct assumes that the file requested changes ownership first. The rest of the formula has a similar interpretation as usual.

Requirement P2 "*The message should only be sent to one client at a time. In particular, if request message* `"Fwd !F !C !S"` *from client C regarding file F reaches server S via the server interface, then file F eventually is sent from server S and reaches client C. Moreover, no pieces of file F can be sent from any other server* $S_1 \neq S$ *to any other client* $C1 \neq C$. "

macro $P_2\,(F, C) =$
(
[**true*** . {`Fwd ?`F : **string** `?`C : **string** `?`S : **string**} . (**not** ({`Trans !`F `!`S `!`C}))*]
⟨ {`Trans !`F `!`S `!`C} ⟩ **true**
and
[**true*** . {`Fwd ?`F : **string** `?`C : **string** `?`S : **string**} . (**not** ({`Trans !`F `!`S `!`C}))* .
{`Trans !`F `?`S_1 : **string** `?`C_1 : **string where** $(S_1 \neq S)$ **or** $(C_1 \neq C)$}] **false**
and
[**true*** . {`Fwd ?`F : **string** `?`C : **string** `?`S : **string**} . (**not** ({`Trans !`F `!`S `!`C}))*
{`Trans !`F `!`S `!`C} . (**not** ({`Trans !`F `!`S `!`C}))*]
⟨ `Trans !`F `!`S `!`C ⟩ **true**
and
[**true*** . {`Fwd ?`F : **st ring** `?`C : **string** `?`S : **string**} . (**not** ({`Trans !`F `!`S `!`C}))*
{`Trans !`F `!`S `!`C} . (**not** ({`Trans !`F `!`S `!`C}))* .
{`Trans !`F `?`S_1 : **string** `?`C_1 : **string where** $(S_1 \neq S)$ **or** $(C_1 \neq C)$}] **false**
)
$P_2()$

We assume that every file is truncated into 2 equal pieces. The first conjunct expresses the eventuality that a forward request for certain file will be followed by the transfer of a part of the requested file from the server that is the actual owner of this file to the client that requested it. Compared to the first conjunct, the second conjunct being a purely safety property predicates that no other server transfers pieces of the requested file to the client initially requesting it and that no other client receives pieces of the requested file. The third conjunct differs from the first conjunct in that it refers to the second part of the requested file. Similar is the relation of the fourth conjunct to the second conjunct.

A generic MCL formula (an independent from the size of the requested file formula) is impossible under the present MCL model checker capabilities. In order to achieve this, we would have to extract the exact size of each file from array $sizef$. However, the extraction of information from complex data structures is actually not supported.

Requirement P3 "*The server interface can set the owner of file F to server S_2 via communication label "`Chown !`F `!`S_2`" provided that F is not being transferred to any client.*
"

macro $P_3\,(F, C) =$
(
[(**not** ({`Chown !`F `?`S_1 : **string where** $S_1 \neq S$)})*) . {`Rqt !`F `?`C : **string**} .
(**not** ({`Ack !`F `!`S `!`C)*) . {`Chown !`F `?`S_2 : **string where** $S_2 \neq S$).
(**not** ({`Ack !`F `!`S `!`C)*) . (`Ack !`F `!`S `!`C)] **false**
and
[**true** . {`Chown !`F `!`S}) . {`Rqt !`F `?`C : **string**} .
(**not** ({`Ack !`F `!`S `!`C)*) . {`Chown !`F `?`S_2 : **string where** $S_2 \neq S$).
(**not** ({`Ack !`F `!`S `!`C)*) . (`Ack !`F `!`S `!`C)] **false**
)
end_macro

$$P_3(\texttt{"}F_1\texttt{"}, \texttt{"}C_1\texttt{"}) \textbf{ and } P_3(\texttt{"}F_2\texttt{"}, \texttt{"}C_2\texttt{"})$$

This property corresponds to the pattern "α_1 does not occur between α_2 and α_3", which is expressed by the following scheme, easily recognizable in this formula:

$$[\,\textbf{true}^*\,.\,\alpha_2\,.\,(\textbf{not}\ \alpha_3)^*\,.\,\alpha_1\,.\,(\textbf{not}\ \alpha_3)^*\,.\,\alpha_3\,]\ \textbf{false}$$

In particular, communication labels "*Chown* `!`F `?`S : **string**" cannot occur between communication labels "*Rqt* `!`F `?`C : **string**" and "*Ack* `!`F `!`S `!`C". File F_1 is initially stocked in server S_1 and file F_2 is initially stocked in server S_2 as usual.

Requirement P4 "*The maximum number of file requests is limited to 2.* "

macro $P_4() =$
(
$[\,\textbf{true}^*\,.\,\{\texttt{Rqt ?}F_1 : \textbf{string ?}C : \textbf{string}\}\,.\,(\textbf{not}\ (\{\texttt{Rqt ?}F_2 : \textbf{string ?}C : \textbf{string}\})^*)\,.$
$\{\texttt{Rqt ?}F_3 : \textbf{string !}C\}\,.\,(\textbf{not}\ (\{\texttt{Rqt ?}F_4 : \textbf{string !}C\}))^*\,.\,\{\texttt{Rqt ?}F_5 : \textbf{string !}C\}\,]$
false
)
end_macro
$P_4()$

Requirement P_4 is a classical safety property. The previous formula is equivalent is equivalent tot the MCL formula expressed in natural language as follows:

"*There is no execution path containing 3 commmunication labels of the form* `"Rqt ?`F : ***string*** `?`C : ***string***`"` .

Requirement P5 "*If file F is being transferred to client C, no other client can ask for F, unless the acknowledgement packet* `"Ack !`F `!`S `!`C`"` *has been sent to client C (denoting correct termination of the file transmission).*

macro $P_5\,(F, C) =$
(
$[\,(\textbf{not}\ (\{\texttt{Chown !}F\ \texttt{?}S_1 : \textbf{string}\})^*)\,.\,\{\texttt{Rqt !}F\ \texttt{?}C : \textbf{string}\}\,.$
$(\textbf{not}\ (\{\texttt{Ack !}F\ \texttt{!}S\ \texttt{!}C)^*)\,.\,\{\texttt{Rqt !}F\ \texttt{?}C_2 : \textbf{string where}\ C_2 \neq C).$
$(\textbf{not}\ (\{\texttt{Ack !}F\ \texttt{!}S\ \texttt{!}C)^*)\,.\,(\texttt{Ack !}F\ \texttt{!}S\ \texttt{!}C)\,]\ \textbf{false}$
and
$[\,\textbf{true}\,.\,\{\texttt{Chown !}F\ \texttt{?}S_1 : \textbf{string}\})\,.\,(\textbf{not}\ (\{\texttt{Chown !}F\ \texttt{?}S_2 : \textbf{string})^*)\,.$
$\{\texttt{Rqt !}F\ \texttt{?}C : \textbf{string}\}\,.\,(\textbf{not}\ (\{\texttt{Ack !}F\ \texttt{!}S\ \texttt{!}C)^*)\,.$
$\{\texttt{Rqt !}F\ \texttt{?}C_2 : \textbf{string where}\ C_2 \neq C)\,.$
$(\textbf{not}\ (\{\texttt{Ack !}F\ \texttt{!}S_1\ \texttt{!}C)^*)\,.\,(\texttt{Ack !}F\ \texttt{!}S_1\ \texttt{!}C)\,]\ \textbf{false}$
)
end_macro
$P_5(\texttt{"}F_1\texttt{"}, \texttt{"}C_1\texttt{"})$ **and** $P_5(\texttt{"}F_2\texttt{"}, \texttt{"}C_2\texttt{"})$

The first conjunct denotes that no communication label "`Rqt !`F `?`C_1" can occur between communication labels "`Rqt !`F `!`C" and "`Ack !`F `!`S `!`C". The second conjunct is similar.

Table 5.2: Model checking results for the Simplified File Transfer System

(c, s)	(3,2)	(3,3)	(3,4)	(4,3)	(4,4)	(4,5)	(5,4)
states	9,519	21,376	37,965	110,341	196,113	306,381	899,733
trans.	23,502	56,517	106,980	284,514	532,056	872,390	2,381,004
time	2.1s	2.3s	2.5s	3.1s	4.1s	5.4s	11.1
P1	0.6s	0.8s	0.9s	1.6s	2.4s	3.9s	13.5
P2	0.5s	0.7s	0.9s	2.1s	4.0s	6.7s	29.3
P3	0.6s	0.7s	0.8s	1.4s	2.1s	3.0	11.4
P4	0.3s	0.5s	0.7s	1.7s	3.4s	5.6	32.8
P5	0.6s	0.8s	1.1s	2.3s	3.8s	6.1	24.2
P6	0.5s	0.4s	0.4s	0.5s	0.6s	0.6	1.0

Requirement P6 "*While a file is being transferred to client C, C can ask for a second file. As a means not to exceed the upper limit on the total numbers of requests, we must make sure that no other requests are made in the meantime.*

macro $P_6\,(F_1, C_1) =$
  (
    [(**not** ({`Rqt !`F_1 `?`C_1 : **string**} **or** {`Rqt !`F_2 `?`C_1 : **string**})*))
     {`Chown !`F_1 `?`S : **string**)}
     (**not** ({`Chown !`F_1 `?`S_2 : **string**)} **or** {`Rqt !`F_1 `?`C_1 : **string**} **or** {`Rqt !`F_2 `?`C_1 : **string**})*) .
     {`Rqt !`F_1 `?`C : **string**}]
    ⟨ (**not** ({`Ack !`F_1 `!`S `!`C} **or** {`Rqt !`$F2?C1$: **string**})*) . {`Rqt !`F_2 `?`C} ⟩ **true**
)
end_macro
P_6("F_1", "C_1") **and** P_6("F_2", "C_2")

5.3.2 Verification of the Simplified File Transfer System

Table 5.2 shows the results for several configurations of the LNT simplified file transfer system Specificaton, obtained by instantiating the number of clients (c) and servers (s) in the IS. All requirements were shown to be valid on the IS specification as expected.

5.4 Conclusion

In this chapter, we presented the experimental results related to the verification with MCL of the LNT specifications that were generated by EB^{3}2LNT corresponding to the library management system and the simplified file transfer system. We also demonstrated how the functional requirements for each system were specified in MCL. The basic drawback of our approach in general is that the functional requirements cannot be generated automatically as is the case for LNT specifications. Instead, they should be expressed manually in MCL, which implies that EB^3 users should be familiar with MCL's syntax and expressivity.

Chapter 6

Parametric Model Checking (PMC)

6.1 Introduction

Parametric systems are systems, whose behaviour is scaled in keeping with the predefined value of a system parameter. The *parametric* model checking problem (PMCP) consists in deciding whether a temporal property is true for a *parametric* system regardless of the value assigned to its system parameter. In this paper, we study the PMCP for a particular class of information systems expressed in process algebra EB^3. We define the PARCTL, which covers a subset of the CTL* without the *next-time* operator X, as the temporal logic to describe the properties preserved in the *parametric* system. We then determine the conditions that render EB^3 models *parametric*.

6.2 Background

The combinatorial blow-up of the state-space representation of a concurrent system, also known as state-space explosion, is a usual limitation of conventional model checking techniques. Many approaches have been employed to combat state-space explosion, among which symbolic methods [BMC+92], bounded model checking [BCC+99], partial order reduction [GP93, KP88] and abstraction techniques [CC77, CGL92, DGG97, LGS+95]. The study of finite state concurrent systems with component-wise symmetry, such as caches, bus protocols and network protocols, gave birth to a new branch of abstraction techniques, the so-called symmetry-based model checking [CEF+96, ESW96, ID96]. Symmetry-based techniques exploit the system symmetry to define permutation groups that preserve the system's state labelling and transitions. Such groups specify bisimulation equivalences on the state space and the induced (by the bisimulation relations) quotient model, often smaller than the original one, is used to verify properties in CTL* [EH82] of the original model.

A wide range of concurrent systems, whose state-based models are susceptible to potential state-space explosion, can be classified as *parametric* systems. *Parametric* systems contain one or more system parameters that are natural numbers in most cases and can vary giving different instances of these systems. The *parametric* model checking problem (PMCP) is the model checking problem of *parametric* systems. Unluckily, it was proven to be undecidable in [AK86]. Since then, substantial model checking techniques [CGB86, EK00, EN95, HBR09] have aimed to limit the PMCP to frameworks that render the problem decidable. They focus on establishing an equivalence relation between instances N_n and N_{n+1} of the *parametric*

system[1]. This allows to find *cut-offs* c such that for any $n \geq c$ and temporal property ϕ, N_n satisfies ϕ if and only if N_c satisfies ϕ, denoted by $N_c \models \phi \Leftrightarrow \forall n \geq c : N_n \models \phi$. PMC techniques are closely-related to symmetry-based techniques; the difference being in the equivalence relation obtained. In the PMCP, instance N_{n+1} is related to N_n with *stuttering* bisimulation equivalence, which implies that the fragment of CTL* to be preserved, has to be insensitive to repeated occurences of the same state. In other words, only CTL$^*\setminus$X (CTL* without the *next-time* operator X) is preserved.

The rest of this chapter is organised as follows: Section 6.3 treats the PMC for the library management system directly on the LTS level. Section 6.4 generalizes the approach for EB^3 specifications. Section 6.5 evaluates our technique and compares it existing approaches. Finally, Section 6.6 summarizes the results and draws up lines for future work.

6.3 State-based PMC

The formal verification of information systems (ISs) with the use of model checking techniques is hampered by state space explosion [FFC$^+$10, VLD$^+$13]. Inspired by the PMC abstraction techniques developed in [CGB86, EN95], we apply similar reasoning to address the problem in the context of ISs. In this section, we apply our adapted abstraction technique on the state-based specifications describing ISs.

First, we introduce the notion of *parametric* transition systems (PTSs) defined on index sets. A PTS TS_n defined on index set $\mathcal{N}$ with cardinality n is an LTS enriched with special atomic propositions indexed with values from $\mathcal{N}$. We, then, define the notion of *stuttering* bisimulation equivalence for PTSs. *Stuttering* bisimulation equivalence preserves a fragment of the CTL* that we call *parametric* CTL (PARCTL). PARCTL interpreted over PTSs includes all of the CTL* with the exception of the next-time operator X. In addition, it permits formulas of the form $\vee_i \Phi_i$ and $\wedge_i \Phi_i$, where Φ_i is a formula of this logic and i belongs to the index set of the PTS in question.

We study a library management system the desired behaviour of which was characterized in [Ger06]. We specify the library management system as a PTS TS_p with index set MID denoting the IDs of potential members participating in the IS; p being equal to MID's cardinality. Alternatively, one could use book IDs to reason about the system. As a first step, we show that the temporal properties, in which we are interested, can be expressed in the PARCTL. Then, we prove that TS_p and TS_{p-1} are *stuttering* bisimilar. We move on to prove that TS_{p-1} and TS_{p-2} are *stuttering* bisimilar etc. Repeating this procedure and exploiting compositionality, model checking TS_p against PARCTL properties can be reduced to model checking some PTS TS_c with $c < p$ against the same PARCTL properties, whose special atomic propositions are indexed with values among c potential member IDs. We reason about the optimal (minimal) value of c depending on the system specifications. We call c the *cut-off* value for MID's cardinality. Since the cardinality of TS_c's index set is strictly less than the cardinality of TS_p's index set, the use of PMC on verifying temporal properties of TS_p oughts to be very effective.

6.3.1 Theoretical Framework

Definition 6.3.1. Parametric Transition Systems (PTSs)

[1] N_n denotes the instance for system parameter equal to $n \in \mathbb{N}$

A Parametric Transition System (PTS) TS_m is a tuple $(S, \rightarrow, s^0, AP, BP, \mathcal{N}, L)$, where:

- *S is a set of states,*
- *$\rightarrow \subseteq S \times S$ is a transition relation*[2]*,*
- *s^0 is the initial state,*
- *AP is a set of atomic propositions,*
- *$\mathcal{N}$ is a set of indexed values (index set) with cardinality m*
- *BP is a set of atomic propositions,*
- *$L : S \rightarrow \mathcal{P}(AP) \cup \mathcal{P}(BP \times \mathcal{N})$*[3] *is the labelling function assigning elements of AP and indexed elements of BP with values from $\mathcal{N}$ to states of TS_m*[4]*.*

Definition 6.3.2. Paths and Partitions of Paths
Let $TS = (S, \rightarrow, s^0, AP, BP, \mathcal{N}, L)$ be a PTS. A path π in TS is an (infinite) sequence $s_0 s_1 \ldots$ of states $s_i \in S$ such that $s_i \rightarrow s_{i+1}$ with $i \geq 0$. For integer $i \geq 0$, π_i stands for the $(i+1)$-th state of path π and $\pi_i..$ stands for the subsequence of π starting from π_i. A partition of path π is denoted by $S_1 S_2 \ldots$ such that if $s_j \in S_i$, then $|S_1|+\ldots+|S_{i-1}| < j \leq |S_1|+\ldots+|S_{i-1}|+|S_i|$, where $|S_i|$ denotes the number of states in S_i.

Definition 6.3.3. Stuttering Bisimulation for PTSs
Let $TS_1 = (S_1, \rightarrow_1, s_1^0, AP, BP, \mathcal{N}_1, L_1)$ and $TS_2 = (S_2, \rightarrow_2, s_2^0, AP, BP, \mathcal{N}_2, L_2)$ be two PTSs. Let $\mathcal{E} \subseteq \mathcal{N}_1 \times \mathcal{N}_2$ be a total relation over sets $\mathcal{N}_1$ and $\mathcal{N}_2$, which we call correspondence relation of PTSs TS_1 and TS_2. A stuttering bisimulation relation between TS_1 and TS_2 is defined as a set of relations $\mathcal{R} \doteq \{\mathcal{R}_{j,k} \subseteq S_1 \times S_2 \mid$ for $j \in \mathcal{N}_1,\ k \in \mathcal{N}_2$ with $(j,k) \in \mathcal{E}\}$ such that for every $s_1 \in S_1$, $s_2 \in S_2$, and $(s_1, s_2) \in \mathcal{R}_{j,k}$, the following conditions are satisfied:

i) $(s_1^0, s_2^0) \in \mathcal{R}_{j,k}$,

ii) for all $A \in AP$, it follows that $A \in L_1(s_1) \Leftrightarrow A \in L_2(s_2)$ and for all $B \in BP$, it follows that $B_j \in L_1(s_1) \cap (BP \times \{j\}) \Leftrightarrow B_k \in L_2(s_2) \cap (BP \times \{k\})$,

iii) if $s_1 \rightarrow_1 s_1'$ such that $(s_1', s_2) \notin \mathcal{R}_{j,k}$, then there exists a finite path fragment $s_2 u_1 \ldots u_n s_2'$ in TS_2 such that $n \geq 0$, $(s_1, u_i) \in \mathcal{R}_{j,k}$, $i = 1, \ldots, n$, and $(s_1', s_2') \in \mathcal{R}_{j,k}$,

iv) if $s_2 \rightarrow_2 s_2'$ such that $(s_1, s_2') \notin \mathcal{R}_{j,k}$, then there exists a finite path fragment $s_1 u_1 \ldots u_n s_1'$ in TS_1 such that $n \geq 0$, $(s_2, u_i) \in \mathcal{R}_{j,k}$, $i = 1, \ldots, n$, and $(s_1', s_2') \in \mathcal{R}_{j,k}$.

Definition 6.3.3.ii denotes, on the one hand, equality among unindexed atomic proposition labels for states s_1 and s_2. On the one hand, it introduces a notion of *correspondence* among indexed atomic labels of state s_1 and state s_2 based on relation $\mathcal{E}$ over $\mathcal{N}_1$ and $\mathcal{N}_2$. Notice that $\mathcal{E}$ oughts to be total over index sets $\mathcal{N}_1$ and $\mathcal{N}_2$, i.e. for all $c \in \mathcal{N}_1$, there exists $d \in \mathcal{N}_2$ such that $(c,d) \in \mathcal{E}$ and vice-versa. In the following, we say that TS_1 is *stuttering* bisimular to TS_2 if there is a *stuttering bisimulation relation* $\mathcal{R}$ between TS_1 and TS_2. Furthermore, for states $s_1 \in S_1$, $s_2 \in S_2$ such that $(s_1, s_2) \in \mathcal{R}_{c,d}$, we say that s_1 and s_2 are *stuttering bisimular* states with respect to $\mathcal{R}_{c,d}$.

[2] we write $s_1 \rightarrow s_2$ instead of $(s_1, s_2) \in \rightarrow$
[3] $\mathcal{P}(K)$ is the dynamic set of set K
[4] we write $B_j \in BP \times \mathcal{N}$ instead of $(B, j) \in BP \times \mathcal{N}$

Definition 6.3.4. CTL$^*\backslash$x *Let* $TS_m = (S, \rightarrow, s^0, AP, L)$ *be an LTS.* CTL$^*\backslash$x *formulas are defined as follows:*

$$\begin{array}{lcl} \Phi & ::= & \alpha \mid true \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\phi \\ \phi & ::= & \Phi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \boldsymbol{U} \phi_2. \end{array}$$

Definition 6.3.5. Parametric CTL (PARCTL)
Let $TS_m = (S, \rightarrow, s^0, AP, \mathcal{N}, BP, L)$ *be a PTS. Parametric CTL (*PARCTL*) formulas are defined as follows:*

$$\begin{array}{lcl} \Phi & ::= & \alpha \mid true \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\phi \mid \oplus\Psi \\ \Psi & ::= & \Phi \mid \beta \\ \phi & ::= & \Phi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \boldsymbol{U} \phi_2. \end{array}$$

Basic ingredients of PARCTL are atomic propositions $\alpha \in AP$, Boolean connectors like conjunction $\wedge$, disjunction $\vee$, and negation $\neg$. We may derive propositional logic operators, such as disjunction $\vee$, and $\rightarrow$ as usual. PARCTL consists mainly of two types of formulas:

- state formulas Φ expressing properties of states,
- intermediary formulas Ψ that are prefixed by operator $\oplus$, and
- path formulas ϕ expressing properties of paths.

In particular, complex state formulas are obtained by prefixing path formulas either with the path quantifier $\exists$ (for "some path") or the path quantifier $\forall$ (for "all paths"). The universal path quantifier $\forall$ can be defined by way of existential quantification and negation: $\forall\phi = \neg\exists\neg\phi$. The absence of the usual *next-time* operator X of the CTL* is to note among the usual path formula operators of CTL* used in the PARCTL. As opposed to CTL*, PARCTL formulas may contain the parametric operator $\oplus$ that is related to the atomic propositions $b \in BP$. The abstract syntax of PARCTL is further restricted as follows:

- Operator $\oplus$ cannot appear syntactically in formulas embedded within the scope of other operators $\oplus$, e.g. $\oplus(\alpha \wedge \beta \wedge \oplus\beta)$ is not a valid PARCTL state formula.
- Operator $\oplus$ cannot appear syntactically in path formulas ϕ_1 and ϕ_2 of complex path formulas $\phi_1 \mathbf{U} \phi_2$.

We derive the temporal operators $\Diamond$, and $\Box$ with the aid of the temporal operator $\mathbf{U}$ as follows: $\Diamond\phi = true\mathbf{U}\phi$, and $\Box\phi = \neg\Diamond\neg\phi$.

We define an auxiliary function $\mathcal{N}$, whose role is to associate a PARCTL formula to a CTL$^*\backslash$x formula.

Definition 6.3.6. From the PARCTL to the CTL$^*\backslash$x
Let TS *be a PTS* $(S, \rightarrow, s^0, AP, \mathcal{N}, BP, L)$ *and let* Φ *be a* PARCTL *state formula. The effect of function* $\mathcal{N}$ *on state formula* Φ *is defined as follows:*

$$\begin{array}{lcl|lcl|lcl} \mathcal{N}(\Phi ::= true) & = & true & \mathcal{N}(\alpha) & = & \alpha & \mathcal{N}(\oplus\Psi) & = & \bigvee_{i\in\mathcal{N}} \mathcal{N}(\Psi) \\ \mathcal{N}(\Phi_1 \wedge \Phi_2) & = & \mathcal{N}(\Phi_1) \wedge \mathcal{N}(\Phi_2) & \mathcal{N}(\neg\Phi) & = & \neg\mathcal{N}(\Phi) & \mathcal{N}(\exists\phi) & = & \exists\mathcal{N}(\phi) \end{array}$$

the effect of $\mathcal{N}$ on intermediary formula Ψ is defined as follows:

$$\mathcal{N}(\Psi ::= \Phi) = \mathcal{N}(\Phi) \mid \mathcal{N}(\beta) = \beta_i$$

and the effect of $\mathcal{N}$ on path formula ϕ is defined as follows:

$$\begin{array}{lcl|lcl} \mathcal{N}(\phi ::= \Phi) & = & \mathcal{N}(\Phi) & \mathcal{N}(\phi_i \wedge \phi_2) & = & \mathcal{N}(\phi_1) \wedge \mathcal{N}(\phi_2) \\ \mathcal{N}(\neg\phi) & = & \neg\mathcal{N}(\phi) & \mathcal{N}(\phi_1 \boldsymbol{U}\, \phi_2) & = & \mathcal{N}(\phi_1) \boldsymbol{U}\, \mathcal{N}(\phi_2). \end{array}$$

Definition 6.3.7. Satisfaction Relation for the CTL$^*\backslash$X
Let $TS = (S, \rightarrow, s^0, AP, L)$ be an LTS. The satisfaction relation $\models$ for CTL$^*\backslash$X *state formulas Φ to is defined inductively on the structure of Φ as follows:*

$TS_m, s \models true$
$TS_m, s \models \alpha \in AP \Leftrightarrow \alpha \in L(s) \quad TS_m, s \models \Phi_1 \wedge \Phi_2 \Leftrightarrow TS_m, s \models \Phi_i,\ i = 1, 2$
$TS_m, s \models \neg\Phi \Leftrightarrow TS_m, s \not\models \Phi \quad TS_m, s \models \exists\phi \Leftrightarrow TS_m, \pi \models \phi$ *for some* $\pi \in Paths(s)$,

where $Paths(s)$ denotes the paths in TS_m starting from state s. For path π, the satisfaction relation $\models$ for restricted PARCTL *path formulas ϕ is derived as below:*

$TS_m, \pi \models \Phi \Leftrightarrow TS_m, \pi_0 \models \Phi$
$TS_m, \pi \models \neg\phi \Leftrightarrow TS_m, \pi \not\models \phi$
$TS_m, \pi \models \Phi_1 \wedge \Phi_2 \Leftrightarrow TS_m, \pi \models \Phi_1$ *and* $TS_m, \pi \models \Phi_2$
$TS_m, \pi \models \phi_1 \boldsymbol{U} \phi_2 \Leftrightarrow$
there exists $j \geq 0$ *such that* $TS_m, \pi_j.. \models \phi_2$, *and for all* $0 \leq k < j$, $TS_m, \pi_k.. \models \phi_1$.

CTL$^*\backslash$X formulas are interpreted over the states and the paths of given LTS of the form:

$$TS_m = (S, \rightarrow, s^0, AP, L).$$

The notation $TS_m, s \models f$ denotes that state formula f holds at state s of TS_m, whereas $TS_m, \pi \models g$ denotes that path formula g holds along path π of TS_m. Formula $\exists\phi$ holds in state s if there exists *some* path satisfying ϕ that starts from s. Dually, formula $\forall\phi$ holds in s if *all* paths starting in s satisfy ϕ. Formula $\phi_1 \mathbf{U} \phi_2$ holds for a path if there is some state along the path for which ϕ_2 holds, and ϕ_1 holds in all states prior to that state.

Theorem 6.3.1. Stuttering bisimulation equivalence over paths of PTSs
Let $TS_{m_i} = (S_i, \rightarrow_i, s_i^0, AP, \mathcal{N}_i, BP, L_i)$, $i = 1, 2$ be stuttering *bisimilar PTSs. Let $\mathcal{E}$ be the* correspondence *relation and let $\mathcal{R}$ be the* stuttering *bisimilation relation between TS_{m_1}, and TS_{m_2}. We consider $j \in \mathcal{N}_1$ and $k \in \mathcal{N}_2$ such that $(j, k) \in \mathcal{E}$. For states $s_1 \in S_1$, $s_1' \in S_2$ such that $(s_1, s_1') \in \mathcal{R}_{j,k}$ and for every (infinite) path $\pi = s_1 s_2 \ldots$ in TS_{m_1}, there should be a corresponding path $\pi' = s_1' s_2' \ldots$ in TS_{m_2} such that:*

- *π can be partitioned into $S_1 S_2 \ldots$, and π' can be partitioned into $S_1' S_2' \ldots$*
- *for all $j \in 1..$, $s \in S_j$, and $s' \in S_j'$, s is* stuttering *bisimilar to s', i.e. $(s, s') \in \mathcal{R}_{j,k}$.*

Proof. We prove this by induction on the length of path π.

- Let π be a path of length one, i.e. trivially $\pi = s_1$. Then, $\pi' = s_1'$.

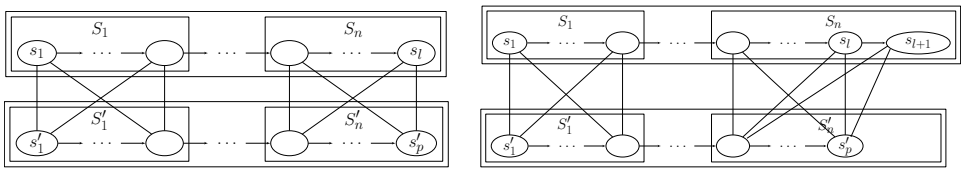


Figure 6.1: Partitioning of π and π' for $(s_{l+1}, s'_p) \in \mathcal{R}_{cd}$

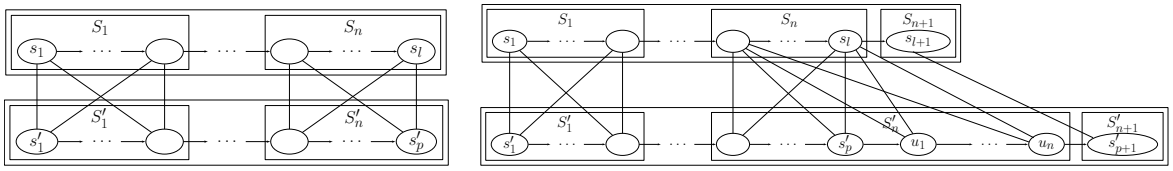


Figure 6.2: Partitioning of π and π' for $(s_{l+1}, s'_p) \notin \mathcal{R}_{cd}$

- Let $\pi = s_1 s_2 \dots s_l$. By the inductive hypothesis, there exists a partition for finite path π equal to $S_1 \dots S_n$, and there exists a path $\pi' = s'_1 s'_2 \dots s'_p$ in TS_{m_2} with corresponding partition $S'_1 \dots S'_n$ such that for all $j \in 1..n$, and $s \in S_j, s' \in S'_j$, we have $(s, s') \in \mathcal{R}_{j,k}$. Now, let's consider a valid path πs_{l+1}, i.e. $s_{l+1} \in \{s \mid s_l \rightarrow_1 s\}$. By way of TS_{m_1} and TS_{m_2} being *stuttering* bisimilar (Definition 6.3.3), we have that:

  1. if $(s_{l+1}, s'_p) \in \mathcal{R}_{j,k}$, then state s_{l+1} can be added to S_n, and S'_n remains unchanged. Hence, the partition for path πs_{n+1} becomes $\overline{S}_1 \dots \overline{S_n}$, where $\overline{S}_i = S_i$, for $i < n$ and $\overline{S}_n = S_n s_{n+1}$. The partition for π' becomes $\overline{S}'_1 \dots \overline{S}'_n$, $\overline{S}'_i = S'_i$, for $i \leq n$.
  2. if $(s_{l+1}, s'_p) \notin \mathcal{R}_{j,k}$, there exists a finite path fragment $s'_p u_1 \dots u_m s'_{p+1}$ in TS_2 with $m \geq 0$ and $(s_l, u_i) \in \mathcal{R}_{j,k}$, $0 < i \leq m$ and $(s_{l+1}, s'_{p+1}) \in \mathcal{R}_{j,k}$. Then, a new fragment $\overline{S}_{n+1}$ containing s_{l+1} can be concatenated to the existing partition for π. States u_1, ..., u_m can be added to the existing fragment S'_n. Hence, the partition for path πs_{l+1} becomes $\overline{S}_1 \dots \overline{S}_n \overline{S}_{n+1}$, where $\overline{S}_i = S_i$, for $i \leq n$ and $\overline{S}_{n+1} = s_{n+1}$. The partition for $\pi' u_1 \dots u_m s'_{p+1}$ becomes $\overline{S}'_1 \dots \overline{S}'_n \overline{S}'_{n+1}$, where $\overline{S}'_i = S'_i$, for $i < n$, $\overline{S}'_n = S'_n u_1 \dots u_m$, and $\overline{S}'_{n+1} = s'_{p+1}$. Notice that the *balance*[5] for both cases i) and ii) remains zero ensuring that the inductive procedure is sound.

□

Regarding Theorem 6.3.1, we say that paths π and π' are *stuttering* bisimilar with regards to $\mathcal{R}_{c,d}$. Notice that the converse of Theorem 6.3.1 is valid too. Figure 6.1 depicts the inductive construction of partitions S_n and S'_n $(s_{l+1}, s'_p) \in \mathcal{R}_{j,k}$, as well as the *stuttering* bisimilarity between corresponding states. Figure 6.4 treats the complementary case $(s_{l+1}, s'_p) \notin \mathcal{R}_{j,k}$.

Theorem 6.3.2. Preservation of the PARCTL
Let $TS_{m_i} = (S_i, \rightarrow_i, s_i^0, AP, \mathcal{N}_i, BP, L_i)$, $i = 1, 2$ be stuttering bisimilar PTSs. Let $\mathcal{E}$ be the correspondence *relation and let $\mathcal{R}$ be the stuttering bisimilation relation of TS_{m_1} and TS_{m_2}. For stuttering bisimilar states $s_1 \in S_1$, $s'_1 \in S_2$ with regards to $\mathcal{R}$, i.e. $(s_1, s'_1) \in \mathcal{R}_{c,d}$ and* stuttering *bisimilar paths $\pi = s_1 s_2 \dots$ in TS_{m_1}, $\pi' = s'_1 s'_2 \dots$ in TS_{m_2} with regards to $\mathcal{R}$, it follows that:*

[5] the difference between the number of state fragments into which π and π' are partitioned

- $TS_{m_1}, s_1 \models \mathcal{N}_1(\Phi) \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\Phi)$ *for every* PARCTL *state formula* Φ.
- $TS_{m_1}, \pi \models \mathcal{N}_1(\phi) \Leftrightarrow TS_{m_2}, \pi' \models \mathcal{N}_2(\phi)$ *for every* PARCTL *path formula* ϕ.

Proof. We proceed with structural induction on PARCTL state formula Φ and PARCTL path formula ϕ.

- $\Phi = true$. It holds trivially.
- $\Phi = \alpha \in AP$.

  $TS_{m_1}, s_1 \models \mathcal{N}_1(\alpha) = \alpha \Leftrightarrow \alpha \in L_1(s_1)$ (Def. 6.3.6, 6.3.7)
  for all $(s_1, s_1') \in \mathcal{R}_{c,d},\ \alpha \in AP,\ \alpha \in L_1(s_1) \Leftrightarrow \alpha \in L_2(s_1')$ (Def. 6.3.3.ii)
  $\alpha \in L_2(s_1') \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\alpha) = \alpha$ (Def. 6.3.7, 6.3.6)

  By way of transitivity, it follows that:

  $$TS_{m_1}, s_1 \models \mathcal{N}_1(\alpha) \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\alpha).$$

- $\Phi = \neg\Phi_1$.

  $TS_{m_1}, s_1 \models \mathcal{N}_1(\neg\Phi_1) = \neg\mathcal{N}_1(\Phi_1) \Leftrightarrow TS_{m_1}, s_1 \not\models \mathcal{N}_1(\Phi_1)$ (Def. 6.3.6, 6.3.7)
  $TS_{m_1}, s_1 \not\models \mathcal{N}_1(\Phi_1) \Leftrightarrow TS_{m_2}, s_1' \not\models \mathcal{N}_2(\Phi_1)$ (Induction Hypothesis)
  $TS_{m_2}, s_1' \not\models \mathcal{N}_2(\Phi_1) \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\neg\Phi_1) = \neg\mathcal{N}_2(\Phi_1)$ (Def. 6.3.7, 6.3.6)

  By way of transitivity, it follows that:

  $$TS_{m_1}, s_1 \models \mathcal{N}_1(\neg\Phi_1) \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\neg\Phi_1).$$

- $\Phi = \oplus\Psi$.

  - Let $\Psi = \beta$.

  $TS_{m_1}, s_1 \models \mathcal{N}_1(\oplus\beta) = \bigvee_{i \in \mathcal{N}_1} \beta_i \Leftrightarrow$ for some $c \in \mathcal{N}_1,\ \beta_c \in L_1(s_1)$ (Def. 6.3.6, 6.3.7)
  for all $(s_1, s_1') \in \mathcal{R}_{c,d},\ d \in \mathcal{N}_2,\ (c,d) \in \mathcal{E},\ \beta_c \in L_1(s_1) \Leftrightarrow \beta_d \in L_2(s_1')$ (Def. 6.3.3.ii)
  $\beta_d \in L_2(s_1') \Leftrightarrow TS_{m_2}, s_1' \models \beta_d = \bigvee_{i \in \mathcal{N}_2} \beta_i = \mathcal{N}_2(\oplus\beta)$ (Def. 6.3.7, 6.3.6)

  By way of transitivity, it follows that:

  $$TS_{m_1}, s_1 \models \mathcal{N}_1(\oplus\beta) \Rightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\oplus\beta).$$

  The proof for the opposite direction is similar.

  - Let $\Psi = \Phi'$.

  for some $c \in \mathcal{N}_1,\ TS_{m_1}, s_1 \models \mathcal{N}_1(\oplus\Psi') = \bigvee_{i \in \mathcal{N}_1} \mathcal{N}_1(\Psi') = \mathcal{N}_1(\Psi')[i \leftarrow c]$
  (Def. 6.3.6 and 6.3.7)
  for some $d \in \mathcal{N}_2$ such that $(c,d) \in \mathcal{E}$ and
  $TS_{m_1}, s_1 \models \mathcal{N}_1(\Psi')[i \leftarrow c] \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\Psi')[i \leftarrow d]$ (Induction Hypothesis)
  $TS_{m_2}, s_1' \models \mathcal{N}_2(\Psi')[i \leftarrow d] = \bigvee_{i \in \mathcal{N}_2} \mathcal{N}_2(\Psi') = \mathcal{N}_2(\oplus\Psi')$ (Def. 6.3.7, 6.3.6)

By way of transitivity, it follows that:

$$TS_{m_1}, s_1 \models \mathcal{N}_1(\oplus\Psi') \Rightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\oplus\Psi').$$

The proof for the opposite direction is similar.

- $\Phi = \Phi_1 \wedge \Phi_2$.

  $TS_{m_1}, s_1 \models \mathcal{N}_1(\Phi_1 \wedge \Phi_2) \Leftrightarrow$ for $i = 1, 2,\ TS_{m_1}, s_1 \models \mathcal{N}_1(\Phi_i)$ (Def. 6.3.6, 6.3.7)
  for $i = 1, 2,\ TS_{m_1}, s_1 \models \mathcal{N}_1(\Phi_i) \Leftrightarrow$ for $i = 1, 2,\ TS_{m_2}, s_1' \models \mathcal{N}_2(\Phi_i)$ (Induction Hypothesis)
  $TS_{m_2}, s_1' \models \mathcal{N}_2(\Phi_i),\ i = 1, 2 \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\Phi_1 \wedge \Phi_2)$ (Def. 6.3.7, 6.3.6)

  By way of transitivity, it follows that:

  $$TS_{m_1}, s_1 \models \mathcal{N}_1(\Phi_1 \wedge \Phi_2) \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\Phi_1 \wedge \Phi_2).$$

- $\Phi = \exists\phi$.

  $TS_{m_1}, \pi \models \mathcal{N}_1(\exists\phi) \Leftrightarrow$ for some $\pi \in Paths(s_1),\ TS_{m_1}, \pi \models \mathcal{N}_1(\phi)$ (Def. 6.3.6, 6.3.7)
  there exist struttering bisimilar paths π, π' such that:
  $TS_{m_1}, \pi \models \mathcal{N}_1(\phi) \Leftrightarrow TS_{m_2}, \pi' \models \mathcal{N}_2(\phi)$ (Induction Hypothesis)
  for some $\pi' \in Paths(s_1'),\ TS_{m_2}, \pi' \models \mathcal{N}_2(\phi) \Leftrightarrow TS_{m_2}, s_1' \models \mathcal{N}_2(\exists\phi)$ (Def. 6.3.7, 6.3.6)

  By way of transitivity, it follows that:

  $$TS_{m_1}, \pi \models \mathcal{N}_1(\exists\phi) \Rightarrow TS_{m_2}, \pi' \models \mathcal{N}_2(\exists\phi).$$

  The proof for the opposite direction is similar.

- $\phi = \phi_1 \wedge \phi_2$. This case is similar to $\Phi = \Phi_1 \wedge \Phi_2$.

- $\phi = \neg\phi_1$. This case is similar to $\Phi = \neg\Phi_1$.

- $\phi = \Phi$.

  $TS_{m_1}, \pi \models \mathcal{N}_1(\Phi) \Leftrightarrow TS_{m_1}, \pi_0 \models \mathcal{N}_1(\Phi)$ (Def. 6.3.6, 6.3.7)
  $TS_{m_1}, \pi_0 \models \mathcal{N}_1(\Phi) \Leftrightarrow TS_{m_2}, \pi_0' \models \mathcal{N}_2(\Phi)$ (Induction Hypothesis)
  $TS_{m_2}, \pi_0' \models \mathcal{N}_2(\Phi) \Leftrightarrow TS_{m_2}, \pi' \models \mathcal{N}_2(\Phi)$ (Def. 6.3.7, 6.3.6)

  The validity check of path formula ϕ over path π translates to the validity check of state formula Φ over the first state of π, i.e. π_0. The induction hypothesis is applicable over state formula Φ, and $\pi_0 = s_1$. By way of transitivity, it follows that:

  $$TS_{m_1}, \pi \models \mathcal{N}_1(\Phi) \Rightarrow TS_{m_2}, \pi' \models \mathcal{N}_2(\Phi).$$

  The proof for the opposite direction is similar.

- $\phi = \Phi_1 \mathbf{U} \Phi_2$. By force of Def. 6.3.6, it is found that $\mathcal{N}_1(\Phi_1 \mathbf{U} \Phi_2) = \mathcal{N}_1(\Phi_1) \mathbf{U} \mathcal{N}_1(\Phi_2)$. Then, by Def. 6.3.7, it follows that:

$$\begin{aligned} TS_{m_1}, \pi \models \mathcal{N}_1(\phi_1 \mathbf{U} \phi_2) \Leftrightarrow \text{there exists } j \geq 0 \text{ such that:} \\ TS_{m_1}, \pi_j.. \models \mathcal{N}_1(\phi_2) \text{ and } \forall k < j,\ TS_{m_1}, \pi_k.. \models \mathcal{N}_1(\phi_1) \end{aligned} \tag{6.1}$$

  Due to Theorem 6.3.1, there should be path $\pi' \in TS_{m_2}$ such that $\pi'_0 = s'_1$ permitting the partitioning of path π into $S_1 S_2 \ldots$, and the partitioning of path π' into $S'_1 S'_2 \ldots$ such that for all $j \in 1..$, $s \in S_j$, and $s' \in S'_j$, state s be *stuttering* bisimilar to state s', i.e. $(s, s') \in \mathcal{R}_{c,d}$. Conforming to (6.1), let π_j be the state belonging to S_r for which $TS_{m_1}, \pi_j.. \models \mathcal{N}_1(\phi_2)$. This means that it should be "$|S_1| + \ldots + |S_{r-1}| < j \leq |S_1| + \ldots + |S_{r-1}| + |S_r|$". $\pi_j..$ is obtained by eliminating the first j states in π. In other words, the prefix $S_1 \ldots S_{r-1}$, and the first "$j - |S_1| - \ldots - |S_{r-1}|$" states in S_j are removed from path π to obtain $\pi_j...$ Note that for $l = |S'_1| + \ldots + |S'_{r-1}|$, path $\pi'_l..$ will be *stuttering* bisimilar to $\pi_j...$ By way of the induction hypothesis, we have that $TS_{m_1}, \pi_j.. \models \mathcal{N}_1(\phi_2) \Leftrightarrow TS_{m_2}, \pi'_l.. \models \mathcal{N}_2(\phi_2)$. For $0 \leq m < l$, there is index k with $0 \leq k < j$ such that $\pi_k..$ be *stuttering* bisimilar to $\pi'_m...$ Note that k is strictly inferior to j, because l corresponds to the first state in S'_r. Then, $TS_{m_1}, \pi_k.. \models \mathcal{N}_1(\phi_1) \Leftrightarrow TS_{m_2}, \pi_m.. \models \mathcal{N}_2(\phi_1)$, for all $0 \leq m < l$ (Induction Hypothesis). As a result, we obtain the following:

$$\begin{aligned} TS_{m_2}, \pi' \models \mathcal{N}_2(\phi_1 \mathbf{U} \phi_2) \Leftrightarrow \text{ there exists } l \geq 0 \text{ such that:} \\ TS_{m_2}, \pi'_l.. \models \mathcal{N}_2(\phi_2), \text{ and } \forall m < l,\ TS_{m_2}, \pi'_m.. \models \phi_1, \end{aligned}$$

  which leads to $TS_{m_1}, \pi \models \mathcal{N}_1(\phi_1 \mathbf{U} \phi_2) \Rightarrow TS_{m_2}, \pi' \models \mathcal{N}_2(\phi_1 \mathbf{U} \phi_2)$. With similar reasoning, and by way of the converse of Theorem 6.3.2, we prove that the opposite direction of Theorem 6.3.2 for $\phi = \Phi_1 \mathbf{U} \Phi_2$.

□

6.3.2 System Specification

We provide here a succinct description in natural language of a simplified version of the library management system of [Ger06]. The following specifications have been selected among the standard specifications of the system on the grounds that they account for a PTS, which is amenable to PMC:

R1. A book can always be acquired by the library when it is not currently acquired.

R2. A book cannot be acquired by the library if it is already acquired.

R3. An acquired book can be discarded only if it is not borrowed.

R4. A person must be a member of the library in order to borrow a book.

R5. A member can relinquish library membership only when all his loans have been returned.

R6. Ultimately, there is always a procedure that enables a member to leave the library.

6.3.3 Formalization

Let the library management system be specified by the following PTS:

$$TS_p = (S_p, \rightarrow_p, s_p^0, AP, \mathcal{N}_p, BP, L_p).$$

Natural p, which serves as the system parameter, stands for the maximum number of persons entitled to register. Alternatively, one could reason over the maximum number of books m available for acquisition. The actual set of books is denoted by $\text{BID} = \{1, \ldots, m\}$ and the set of persons eventually obtaining membership in the library is denoted by $\text{MID} = \{1, \ldots, p\}$. Let also atomic proposition acq_i denote acquisition of book i by the library and let reg_j denote that j registers to become member of the library. Atomic proposition $bor_{i,j}$ signifies that member j borrows book i. The set of atomic propositions characterizing states in TS_p is defined as $AP = \{acq_1, \ldots, acq_m\}$ and the indexed set of atomic propositions is defined as $BP = \{reg, bor_1, \ldots, bor_m\}$. For state $s \in S_p$, we specify $A = \{i \mid acq_i \in L_p(s)\}$ that stands for the index set of books currently acquired by the library. The index set of registered members is denoted by $M = \{j \mid reg_j \in L_p(s)\}$ and the current loans are denoted by $B = \{(i,j) \mid bor_{i,j} \in L_p(s)\}$. Hence, the proposition labelling of TS_p is defined alternatively as follows:

$$L_p(s) = \{acq_i |\ i \in A\} \ \cup\ \{reg_j |\ j \in M\} \ \cup\ \{bor_{i,j} |\ (i,j) \in B\},$$

the set of states in TS_p is represented as $S_p = \{\ s \mid L_p(s) = \langle A, B, M\rangle\ \}$, the initial state of S_p is defined as $s_p^0 = \langle \emptyset, \emptyset, \emptyset \rangle$ and the transition relation $\rightarrow_p$ of TS_p is given by the following formula:

$$\begin{aligned}
\rightarrow_p \quad = \quad & \{\ (s,s') \mid L_p(s) = \langle A,B,M\rangle,\ L_p(s') = \langle A',B',M'\rangle \text{ such that} && (6.2)\\
& (\forall i \in 1..m,\ \{i\} \cap A = \emptyset\ \wedge\ mod_A(A \cup \{i\})) && (6.3)\\
\otimes \quad & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_A(A \setminus \{i\})) && (6.4)\\
\otimes \quad & (\forall j \in 1..p,\ \{j\} \cap M = \emptyset\ \wedge\ mod_M(M \cup \{j\})) && (6.5)\\
\otimes \quad & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_M(M \setminus \{j\})) && (6.6)\\
\otimes \quad & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_B(B \cup \{(i,j)\})) && (6.7)\\
\otimes \quad & (\forall i \in A,\ \forall j \in M,\ (i,j) \in B\ \wedge\ mod_B(B \setminus \{(i,j)\}))\ \} && (6.8)
\end{aligned}$$

Rules (6.3)-(6.8) describe all possible operations occurring in the system. Rule (6.3) refers to book acquisitions (ACQ). Rule (6.4) refers to book removals by the library (DIS). Notice that removals are possible only for books that are not being currently lent to members. Rule (6.5) refers to member registrations (REG). Rule (6.6) denotes the option for membership removal (UNREG) on condition that the member in question does not owe any book to the library. Rule (6.7) describes loans carried out by members of the library and (6.8) denotes returning books to the library. The use of the XOR operator $\otimes$ ensures asynchronous and unique execution of operations in every step. The expression $mod_x(x' = f(x))$ appearing in the body of the aforementioned rules expresses a special kind of predicate on the current and the primed values of set variable $x \in \{A, B, M\}$. The values of variables belonging to $\{A, B, M\} \setminus \{x\}$ remain unchanged after operation completion. For example, predicate $mod_A(A \cup \{i\})$ is equivalent to "$A' = A \cup \{i\} \wedge B' = B \wedge M' = M$". Figure 6.3 displays the PTSs simulating the library management system for configurations $(m = 1, p = 1)$ and $(m = 1, p = 2)$. Initial states $s_1^0 \in TS_1$ and $s_2^0 \in TS_2$ are denoted by blank (unlabelled) nodes in both graphs. Transitions are bidirectional.

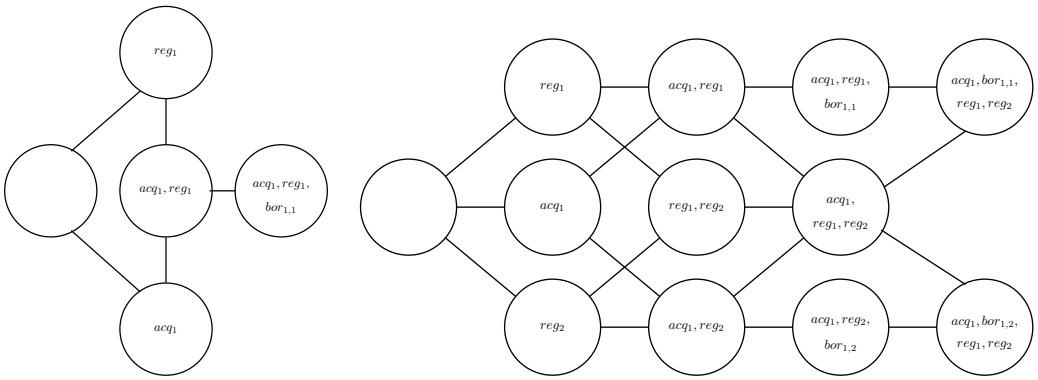


Figure 6.3: Library management system for $(m = 1, p = 1)$ and $(m = 1, p = 2)$

6.3.4 Temporal Properties over TS_p

We formalize the specifications of Section 6.3.2 in the PARCTL. The point is to make sure that the formalization of Section 6.3.3 conforms to the system's desired behaviour. Most specifications of Section 6.3.2 can be translated in the PARCTL as follows:

R1. $\wedge_i \forall\Box\ (\neg acq_i \rightarrow \exists\Diamond\ acq_i)$
R3. $\wedge_i \forall\Box\ (\wedge_j \neg bor_{i,j} \wedge acq_i \rightarrow \exists\ (\wedge_j \neg bor_{i,j} \wedge acq_i \mathbf{U} \neg acq_i))\ \wedge\ \wedge_i \forall\Box\ (\neg acq_i \rightarrow \wedge_j \neg bor_{i,j})$
R4. $\wedge_j \neg\forall\Box\ (\neg reg_j \rightarrow \exists\ (\neg reg_j \mathbf{U} \vee_i bor_{i,j}))$
R5. $\wedge_j \forall\Box\ (\wedge_i \neg bor_{i,j} \rightarrow \exists\ (\wedge_i \neg bor_{i,j} \mathbf{U} \neg reg_j))\ \wedge\ \wedge_j \forall\Box\ (\wedge_i \neg bor_{i,j} \rightarrow reg_j)$
R6. $\wedge_j \forall\Box\ (reg_j \rightarrow \exists\Diamond\ \neg reg_j)$

Among the Boolean expressions used in formulas **R**1, **R3**, **R4**, **R5**, **R6**, conjunctions (disjunctions) can be indexed with book IDS, i.e. $\wedge_i$ ($\vee_i$) or member IDs, i.e. $\wedge_j$ ($\vee_i$).

- Formulas **R1** and **R6** denote classical *liveness* properties of the form "$\forall\Box\ (a \rightarrow \exists\Diamond\ b)$" stating that for states satisfying formula a there exists a transition sequence leading to states satisfying formula b.

- Property **R2** cannot be expressed under the current formalization. The reason lies in the standard action-based nature of the library management system in total opposition to the state-based formalization of Section 6.3.3. In particular, there is no way to distinguish between the last executed action (in this case, the acquisition of book b_i) and a past action, whose effect is still present in the state label (in this case, the library having acquired book b_i, and still possessing it). To palliate this, an alternative formalization would cater for the possibility to deduce from the current state label the possibly enabled actions as well as the last executed action. However, one such formalization would not be interesting at this point. Formula **R5** is similar in nature to **R3**.

- Formula **R3** consists in two conjuncts. The first conjunct expresses a typical *liveness* property stating that a state s_1, in which acquired book b_i is not currently under loan, may transit into another state s_2, in which b_i has been discarded from the library. The transition sequence leading from s_1 to s_2 matches states, in which b_i is neither lent

to any member or discarded. The second conjunct constituting a *safety* property of the form "$\forall\Box\ a$" expresses the fact that a discarded book b_i cannot be lent to any member of the library. Notice that **R3** violates the second restriction on acceptable PARCTL formulas, because it contains indexed disjunctions over member IDs within the scope of the temporal operator $\forall\Box$, which derives from temporal operator **U**. However, supposing that we interchange the roles of members IDs with book IDs; thus, considering book IDs as the new system parameter, then, if we manage to establish the *stuttering* bisimulation relation between TS_m and TS_{m+1} for fixed value of p (the proof is fairly similar to the proof in Section 6.3.5), **R3** can be evaluated over TS_{m_c}, where m_c stands for the *cut-off* value of m.

- Property R4 is formalized as the negation of the eventuality that a non registered member borrows a book.

6.3.5 Stuttering Bisimulation Equivalence of TS_p and TS_{p+1}

In this section, we show that TS_p and TS_{p+1} are *stuttering* bisimilar PTSs. By way of Corollary 6.3.3 and exploiting compositionality, this result allows us to pass from TS_{p+1} to TS_p, then repeat this reasoning gradually descending from TS_p to TS_{p-1} ... etc. until a *cut-off* value is discovered for the system parameter p. The model checking of formulas **R**1, **R3**, **R4**, **R5**, and **R6** over TS_p is reduced to the model checking of formulas **R**1, **R3**, **R4**, **R5**, and **R6** over TS_c.

In practice, the step-by-step system reduction is completed as soon as it is of no more interest to reduce it further. In this case, this may occur if the system is already reduced to an IS of exactly one member, i.e. $p = 1$, or if the initial library management system specifications establish some sort of interplay among multiple members in the form of events (it is not the case for the simplified version of Section 6.3.2). As an intuitive example of the latter situation, consider operation TRANS($b_1 : BID, m_1 : MID, m_2 : MID$) of the classical library specification in [Ger06] associating member m_1 to member m_2. TRANS suggests that member m_1, while being the current borrower of book b_1, should hand it over to member m_2. In general, *cut-off* values for system parameters can be determined upon concrete criteria over the initial system specification.

TS_{p+1} has one more member than TS_p, or more formally $\mathcal{N}_{p+1} = \mathcal{N}_p \cup \{p+1\}$. We specify:

$$\mathcal{R} = \{\mathcal{R}_{j,k} \mid \forall j \in 1..p,\ \forall k \in 1..p,\ (j,k) \in \mathcal{E} \wedge j = k\} \cup \{\mathcal{R}_{p,p+1}\}, \tag{6.9}$$

where $\mathcal{E} \subseteq \mathcal{N}_p \times \mathcal{N}_{p+1}$, $\mathcal{R}_{j,k} \subseteq S_p \times S_{p+1}$ and $\mathcal{R}_{p,p+1} \subseteq S_p \times S_{p+1}$ are defined as follows:

$$\mathcal{E} = \{(j,k) \mid j \in 1..p,\ k \in 1..p \text{ such that } j = k\} \cup \{(p, p+1)\} \tag{6.10}$$

$$\mathcal{R}_{j,k} = \{(s_1,s_2) \mid s_1 \in S_p,\ s_2 \in S_{p+1} \text{ with } L_p(s_1) = \langle A_1, B_1, M_1\rangle, L_{p+1}(s_2) = \langle A_2, B_2, M_2\rangle$$
$$\text{such that for all } i \in 1..m,\ i \in A_1 \Leftrightarrow i \in A_2 \text{ and} \tag{6.11}$$
$$j \in M_1 \Leftrightarrow k \in M_2,\ (i,j) \in B_1 \Leftrightarrow (i,k) \in B_2\} \tag{6.12}$$

$$\mathcal{R}_{p,p+1} = \{(s_1,s_2) \mid \forall s_1 \in S_p,\ \forall s_2 \in S_{p+1} \text{ with } L_p(s_1) = \langle A_1, B_1, M_1\rangle, L_{p+1}(s_2) = \langle A_2, B_2, M_2\rangle$$
$$\text{such that for all } i \in 1..m,\ i \in A_1 \Leftrightarrow i \in A_2 \text{ and} \tag{6.13}$$
$$p \in M_1 \Leftrightarrow p+1 \in M_2,\ (i,p) \in B_1 \Leftrightarrow (i,p+1) \in B_2\} \tag{6.14}$$

For states $s_1 \in S_p$, and $s_2 \in S_{p+1}$ such that $(s_1,s_2) \in \mathcal{R}_{j,k}$, the definition of $\mathcal{R}_{j,k}$ is interpreted as follows:

- the library has acquired, and still possesses exactly the same books in states $s_1 \in S_p$ and $s_2 \in S_{p+1}$ (6.11); also, if j has registered to the library (and is still a member) in state $s_1 \in S_p$, then member k has also registered to the library (and is still a member) in state $s_2 \in S_{p+1}$ (6.12), and vice-versa,

- if the library is lending book i to registered member j in state s_1 of TS_p, then it must be lending book i to registered member k in state s_2 of TS_{p+1}, and vice-verca,

Similar is the interpretation of relation $\mathcal{R}_{p,p+1}$. The interplay among members p and $p+1$ in $\mathcal{R}_{p,p+1}$ is identical to the interplay among members j and k in $\mathcal{R}_{j,k}$. Relation $\mathcal{E}$ is total over sets $\mathcal{N}_1$ and $\mathcal{N}_2$ as expected. Notice, also, that TS_p is a sub-graph of TS_{p+1} in the sense that $S_p \subseteq S_{p+1}$ and $\rightarrow_p \subseteq \rightarrow_{p+1}$.

Theorem 6.3.3. *Relation* $\mathcal{R} = \{\mathcal{R}_{j,k} \mid j \in 1..p,\ k \in 1..p+1,\ (j,k) \in \mathcal{E}\}$ *as given in (6.9) defines a stuttering bisimulation relation between* TS_p *and* TS_{p+1} *of the library management system.*

Proof. We verify that relation $\mathcal{R}$ satisfies the criteria of *stuttering bisimulation* relations over PTSs as given in Definition 6.3.3.

- Let $s_p^0 \in S_p$ be the initial state of TS_p, for which $L_p(s_p^0) = \emptyset$ and let $s_{p+1}^0 \in S_p$ be the initial state of TS_{p+1}, for which $L_{p+1}(s_{p+1}^0) = \emptyset$. Trivially, we have that $(s_p^0, s_{p+1}^0) \in \mathcal{R}_{j,k}$ for all $j \in 1..p,\ k \in 1..p$, and $(j,k) \in \mathcal{E}$, which establishes Definition 6.3.3.i.

- Now, let $j \in 1..p$ and $k \in 1..p+1$ such that $(j,k) \in \mathcal{E}$. Let also state $L_p(s_1) = \langle A_1, M_1, B_1 \rangle \in S_p$ and state $L_{p+1}(s_2) = \langle A_2, M_2, B_2 \rangle \in S_{p+1}$ such that $(s_1, s_2) \in \mathcal{R}_{j,k}$.

  a. The set of atomic propositions AP is defined in Section 6.3.3 as $AP = \{acq_1, \ldots, acq_m\}$ for both PTSs TS_p and TS_{p+1}. If $j,k \in 1..p$ with $j = k$, then, due to (6.11), for all $i \in 1..m$, it is $i \in A_1 \Leftrightarrow i \in A_2$. If $j = p$ and $k = p+1$, then, due to (6.13), for all $i \in 1..m$, it is $i \in A_1 \Leftrightarrow i \in A_2$. Therefore, for all $(j,k) \in \mathcal{E}$ and $(s_1, s_2) \in \mathcal{R}_{j,k}$, it is $i \in A_1 \Leftrightarrow i \in A_2$ (or, equivalently, $A_1 = A_2$).

  b. The set of atomic propositions BP is defined as $BP = \{reg, bor_1, \ldots, bor_m\}$ for both PTSs TS_p and TS_{p+1} in Section 6.3.3. If $j,k \in 1..p$ such that $j = k$, then, due to (6.12), it follows that $j \in M_1 \Leftrightarrow k \in M_2$. If $j = p$ and $k = p+1$, then, due to (6.14), it follows that $p \in M_1 \Leftrightarrow p+1 \in M_2$. Therefore, for all $(j,k) \in \mathcal{E}$ and $(s_1, s_2) \in \mathcal{R}_{j,k}$, it follows that $j \in M_1 \Leftrightarrow k \in M_2$. By force of (6.12) and (6.14), it is $(i,j) \in B_1 \Leftrightarrow (i,k) \in B_2$ for all $(j,k) \in \mathcal{E}$ and $(s_1, s_2) \in \mathcal{R}_{j,k}$.

  The previous bullets illustrate that relation $\mathcal{R}$ meets the criteria of Definition 6.3.3.ii for *stuttering bisimulations* over PTSs. For $L_p(s_1) = \langle A_1, M_1, B_1 \rangle \in S_p$ and state $L_{p+1}(s_2) = \langle A_2, M_2, B_2 \rangle \in S_{p+1}$ such that $(s_1, s_2) \in \mathcal{R}_{j,k}$, we write $M_1 \sim M_2$ and $B_1 \approx B_2$. Obviously, relations $\sim$ and $\approx$ are reflexive and transitive.

- Let $j \in 1..p$ and $k \in 1..p+1$ such that $(j,k) \in \mathcal{E}$. Let also state $L_p(s_1) = \langle A_1, M_1, B_1 \rangle \in S_p$ and state $L_{p+1}(s_2) = \langle A_2, M_2, B_2 \rangle \in S_{p+1}$ such that $(s_1, s_2) \in \mathcal{R}_{j,k}$. We consider that state $s_1 \in S_p$ is characterized by the book acquisitions $acq_{i_1}, .., acq_{i_n}$ (or, equivalently, $A_1 = \{i_1, .., i_n\}$), by the registered members $reg_{j_1}, .., reg_{j_r}$ (or, equivalently, $M_1 = \{j_1, .., j_r\}$) and by the active loans carried out by the library $bor_{k_1,l_1}, .., bor_{k_q,l_q}$ (or,

equivalently, $B_1 = \{(k_1, l_1), .., (k_q, l_q)\}$). As a means to ensure the consistency of our approach, we assume that the index set of acquired books by the library constitutes a subset of the index set of possible book IDs, i.e. $A_1 \subseteq \{1, ..m\}$. We impose a similar restriction regarding the set of registered members, i.e. $M_1 \subseteq \{1, ..p\}$. As for the set of active loans, the books on loan should be a subset of the currently acquired books and the members, to whom these books are being lent, should be a subset of the currently registered members to the library. Formally, we write $\{k_1, ..k_q\} \subseteq A_1$ and $\{l_1, ..l_q\} \subseteq M_1$. Similar restrictions should be placed on $s_2 \in S_{p+1}$. The proof proceeds as follows:

a. We consider transition $s_1 \rightarrow_p s_1' \in TS_p$ starting from state s_1 such that $(s_1', s_2) \notin \mathcal{R}_{j,k}$ and $L_p(s_1') = \langle A_1', M_1', B_1' \rangle \in S_p$. In order to establish Definition 6.3.3.iii, we must prove that there exists state $L_{p+1}(s_2') = \langle A_2', M_2', B_2' \rangle \in S_{p+1}$ and finite path fragment $s_2 u_1 \ldots u_n s_2'$ in TS_{p+1} such that $n \geq 0$, $(s_1, u_i) \in \mathcal{R}_{j,k},\ i = 1, .., n$ and $(s_1', s_2') \in \mathcal{R}_{j,k}$.

b. The converse condition to a must be treated in order to establish Definition 6.3.3.iv.

In keeping with transition relation $\rightarrow_p$, we consider all possible forms of transition $s_1 \rightarrow_p s_1'$.

1. ($\Rightarrow$) Let $s_1 \rightarrow s_1' \in TS_p$ be of the form (ACQ). Due to (6.3) of Section 6.3.3, there should be $i \in 1..m$ such that $\{i\} \cap A_1 = \emptyset$, $A_1' = A_1 \cup \{i\}$, $B_1' = B_1$, and $M_1' = M_1$. By $A_1 = A_2$ and (6.12), it follows that $\{i\} \cap A = \emptyset$. As a consequence of the respective rule (6.3) for transition relation $\rightarrow_{p+1}$, there should be state $s_2' = \langle A_2', B_2', M_2' \rangle \in S_{p+1}$, and transition $s_2 \rightarrow_{p+1} s_2' \in T_{p+1}$ such that $A_2' = A_2 \cup \{i\}$, $B_2' = B_2$, and $M_2' = M_2$. Since $A_1 = A_2$, $A_1' = A_1 \cup \{i\}$, and $A_2' = A_2 \cup \{i\}$, we deduce that $A_1' \sim A_2'$. Since $M_1 \sim M_2$, $M_1' = M_1$, and $M_2' = M_2$, we deduce that $M_1' \sim M_2'$ and, similarly, from $B_1 \approx B_2$, $B_1' = B_1$, and $B_2' = B_2$, we deduce that $B_1' \approx B_2'$. Combining $A_1' \sim A_2'$, $M_1' \sim M_2'$, and $B_1 \approx B_2$, we prove that for every $(j, k) \in \mathcal{E}$, $(s_1', s_2') \in \mathcal{R}_{j,k}$. Hence, state s_2 executes an (ACQ) transition to state s_2' in TS_{p+1} without *stuttering*, i.e. $n = 0$.
($\Leftarrow$) The proof for the opposite direction is similar.

2. ($\Rightarrow$) Let $s_1 \rightarrow_p s_1' \in TS_p$ be of the form (DIS). Due to rule (6.4) of transition relation $\rightarrow_p$ in Section 6.3.3, there should be $i \in \mathrm{A}_1$ such that for all $j \in M_1$, $\{(i, j)\} \cap B_1 = \emptyset$, $A_1' = A_1 \setminus \{i\}$, $B_1' = B_1$, and $M_1' = M_1$.
   (a) We assume that for all $j \in M_2$, $\{(i, j)\} \cap B_2 = \emptyset$. As a consequence of the respective rule (6.4) for transition relation $\rightarrow_{p+1}$, there should be state $s_2' = \langle A_2', B_2', M_2' \rangle$, and transition $s_2 \rightarrow_{p+1} s_2' \in TS_{p+1}$ such that for all $j \in M_2$, $\{(i, j)\} \cap B_2 = \emptyset$, $A_2' = A_2 \setminus \{i\}$, $B_2' = B_2$, and $M_2' = M_2$. From $A_1 = A_2$, $A_1' = A_1 \setminus \{i\}$, and $A_2' = A_2 \setminus \{i\}$, we deduce that $A_1' \sim A_2'$. From $M_1 \sim M_2$, $M_1' = M_1$, and $M_2' = M_2$, we deduce that $M_1' \sim M_2'$ and, similarly, from $B_1 \approx B_2$, $B_1' = B_1$, and $B_2' = B_2$, we deduce that $B_1' \approx B_2'$. Combining $A_1' \sim A_2'$, $M_1' \sim M_2'$, and $B_1 \approx B_2$, we prove that for all $(j, k) \in \mathcal{E}$, it is $(s_1', s_2') \in \mathcal{R}_{j,k}$. Hence, state s_2 executes a (DIS) transition to state s_2' in TS_{p+1} without *stuttering*, i.e. $n = 0$.
   (b) i. Let $j = k$. Unluckily, it cannot be established from (6.12) that for all $j \in M_2$, $\{(i, j)\} \cap B_2 = \emptyset$. Therefore, it may be the case that for some $k' \in M_2$ such that $k' \neq k = j$, $(i, k') \in B_2$. However, discarding book b_i is not possible in state $s_2 \in TS_{p+1}$, unless member $m_{k'}$ returns book b_i to the

library beforehand. This has the implication that state s_2 transits to state $u_1 = \langle A_2, M_2, B_2 \setminus \{(i,k')\}\rangle \in TS_{p+1}$ via a (RET) operation. Then, state u_1 transits to state $s'_2 = \langle A_2 \setminus \{i\}, M_2, B_2 \setminus \{(i,k')\}\rangle \in TS_{p+1}$ via a (DIS) operation. Following similar arguments to those advanced in 2a and 1, we establish that for every $(j,k) \in \mathcal{E}$, $(s'_1, s'_2) \in \mathcal{R}_{j,k}$. Hence, state s_2 takes a *stuttering* path of length $n = 1$ to state u_1 via a (RET) operation before going to state s'_2 via a (DIS) operation.

ii. Let $j = p$ and $k = p + 1$. The proof is similar.

($\Leftarrow$) The proof for the opposite direction is similar.

3. ($\Rightarrow$) Let $s_1 \rightarrow s'_1 \in TS_p$ be of the form (REG). Due to (6.5) of Section 6.3.3, there should be $j' \in 1..p$ such that $\{j'\} \cap M_1 = \emptyset$, $A'_1 = A_1$, $B'_1 = B_1$, and $M'_1 = M_1 \cup \{j'\}$. We, then, need to reason on the possible values of j'.

   (a) Let $j = j'$.

      i. If $j = k$, then, by (6.12) and $\{j\} \cap M_1 = \emptyset$, it follows that $\{k\} \cap M_2 = \emptyset$. As a consequence of the respective rule (6.5) for transition relation $\rightarrow_{p+1}$, there should be state $s'_2 = \langle A'_2, B'_2, M'_2\rangle$, and transition $s_2 \rightarrow_{p+1} s'_2 \in TS_{p+1}$ such that $\{k\} \cap M_2 = \emptyset$, $A'_2 = A_2$, $B'_2 = B_2$, and $M'_2 = M_2 \cup \{k\}$. It is fairly easy to prove that for every $(j,k) \in \mathcal{E}$ such that $j = j'$, $(s'_1, s'_2) \in \mathcal{R}_{j,k}$ (see corresponding parts of proof in 1 and 2a. Therefore, state s_2 executes a (REG) transition to state s'_2 without *stuttering*, i.e. $n = 0$.

      ii. If $j = p$ and $k = p + 1$, then, by (6.14) and $\{p\} \cap M_1 = \emptyset$, it follows that $\{p + 1\} \cap M_2 = \emptyset$. Hence, there should be state $s'_2 = \langle A'_2, B'_2, M'_2\rangle$, and transition $s_2 \rightarrow_{p+1} s'_2 \in TS_{p+1}$ such that $\{p + 1\} \cap M_2 = \emptyset$, $A'_2 = A_2$, $B'_2 = B_2$, and $M'_2 = M_2 \cup \{p + 1\}$. The rest of the proof follows the lines of 3(a)i.

   (b) Let $j \neq j'$.

      i. If $j = k$, then $(s'_1, s_2) \in \mathcal{R}_{j,k}$, since (6.12) still holds for states s'_1 and s_2 after the execution of $s_1 \rightarrow s'_1 \in TS_p$.

      ii. If $j = p$ and $k = p + 1$, then $(s'_1, s_2) \in \mathcal{R}_{j,k}$, since (6.14) still holds for states s'_1 and s_2 after the execution of $s_1 \rightarrow s'_1 \in TS_p$.

   ($\Leftarrow$) The proof for the opposite direction is similar.

4. ($\Rightarrow$) Let $s_1 \rightarrow s'_1 \in TS_p$ be of the form (UNREG). Due to (6.6) of Section 6.3.3, there should be $j' \in 1..p$ such that for all $i \in A_1$, $\{(i,j')\} \cap B_1 = \emptyset$, $A'_1 = A_1$, $B'_1 = B_1$, and $M'_1 = M_1 \setminus \{j'\}$. We, then, need to reason on the possible values of j'.

   (a) Let $j = j'$.

      i. If $j = k$, then, relying on the fact that forall $i \in A_1$, $\{(i,j')\} \cap B_1 = \emptyset$ and (6.12), we deduce that forall $i \in A_2$, $\{(i,j)\} \cap B_2 = \emptyset$. As a consequence of the respective rule (6.6) for transition relation $\rightarrow_{p+1}$, there should be state $L_{p+1}(s'_2) = \langle A'_2, B'_2, M'_2\rangle$, and transition $s_2 \rightarrow_{p+1} s'_2 \in TS_{p+1}$ such that forall $i \in A_2$, $\{(i,j)\} \cap B_2 = \emptyset$, $A'_2 = A_2$, $B'_2 = B_2$, and $M'_2 = M_2 \setminus \{j\}$. The rest of the proof is straightforward.

      ii. If $j = p$ and $k = p + 1$, then, relying on the fact that forall $i \in A_1$, $\{(i,j')\} \cap B_1 = \emptyset$ and (6.14), we deduce that forall $i \in A_2$, $\{(i,p+1)\} \cap B_2 = \emptyset$. The rest of the proof follows the lines of 4(a)i.

(b) Let $j \neq j'$.

i. If $j = k$, then $(s'_1, s_2) \in \mathcal{R}_{j,k}$, since (6.12) holds for states s'_1 and s_2 after the execution of $s_1 \rightarrow s'_1 \in TS_p$.

ii. Let $j = p$ and $k = p + 1$. If $j' \neq p$, the rest of the proof is similar.

($\Leftarrow$) The proof for the opposite direction is similar.

5. ($\Rightarrow$) Let $s_1 \rightarrow s'_1 \in TS_p$ be of the form (LEND). Due to (6.6) of Section 6.3.3, there should be $i \in A_1$, and $j' \in M_2$ such that $\{(i, j')\} \cap B_1 = \emptyset$, $A'_1 = A_1$, $B'_1 = B_1 \cup \{(i, j')\}$, and $M'_1 = M_1$. We, then, need to reason on the possible values of j'.

(a) Let $j = j'$.

i. If $j = k$, then, by (6.12) and $\{(i, j')\} \cap B_1 = \emptyset$, it follows that $\{(i, j')\} \cap B_2 = \emptyset$. As a consequence of the respective rule (6.7) for transition relation $\rightarrow_{p+1}$, there should be state $L_{p+1}(s'_2) = \langle A'_2, B'_2, M'_2 \rangle$, and transition $s_2 \rightarrow_{p+1} s'_2 \in TS_{p+1}$ such that $\{(i, j')\} \cap B_2 = \emptyset$, $A'_2 = A_2$, $B'_2 = B_2 \cup \{(i, j')\}$, and $M'_2 = M_2$. The rest of the proof is straightforward.

ii. If $j = p$ and $k = p + 1$, then, by (6.14) and $\{(i, p)\} \cap B_2 = \emptyset$, we deduce that $\{(i, p + 1)\} \cap B_2 = \emptyset$. The rest of the proof follows the lines of 5(a)i.

(b) Let $j \neq j'$.

i. If $j = k$, then $(s'_1, s_2) \in \mathcal{R}_{j,k}$, since (6.12) holds for states s'_1 and s_2 after the execution of $s_1 \rightarrow s'_1 \in TS_p$.

ii. Let $j = p$ and $k = p + 1$. The rest of the proof is similar.

6. ($\Rightarrow$) Let $s_1 \rightarrow s'_1 \in TS_p$ be of the form (RET). Due to (6.7) of Section 6.3.3, there should be $i \in A_1$, and $j' \in M_2$ such that $\{(i, j')\} \in B_1$, $A'_1 = A_1$, $B'_1 = B_1 \backslash \{(i, j')\}$, and $M'_1 = M_1$. We, then, need to reason on the possible values of j'.

(a) Let $j = j'$.

i. If $j = k$, then, by (6.12) and $\{(i, j')\} \cap B_1 = \emptyset$, it follows that $\{(i, j')\} \in B_2$. As a consequence of the respective rule (6.7) for transition relation $\rightarrow_{p+1}$, there should be state $s'_2 = \langle A'_2, B'_2, M'_2 \rangle$, and transition $s_2 \rightarrow_{p+1} s'_2 \in TS_{p+1}$ such that $\{(i, j')\} \in B_2$, $A'_2 = A_2$, $B'_2 = B_2 \backslash \{(i, j')\}$, and $M'_2 = M_2$. The rest of the proof is straightforward.

ii. If $j = p$ and $k = p + 1$, then, by (6.14) and $\{(i, p)\} \in B_2$, we deduce that $\{(i, p + 1)\} \in B_2$. The rest of the proof follows the lines of 6(a)i.

(b) Let $j \neq j'$.

i. If $j = k$, then $(s'_1, s_2) \in \mathcal{R}_{j,k}$, since (6.12) holds for states s'_1 and s_2 after the execution of $s_1 \rightarrow s'_1 \in TS_p$.

ii. Let $j = p$ and $k = p + 1$. The rest of the proof is similar.

□

Figure 6.4 illustrates the *stuttering* bisimilar states of TS_1 and TS_2 with regards to $\mathcal{R}_{1,1}$ for $m = 1$ (one book). States in TS_2 of the same colour belong to the same partition class and they are *stuttering* bisimilar to the corresponding states of TS_1 of the same colour. For clarity reasons, each state $s \in S_2$ has been labelled with $L_2(s) \cap (BP \times \{1\})$. As a example,

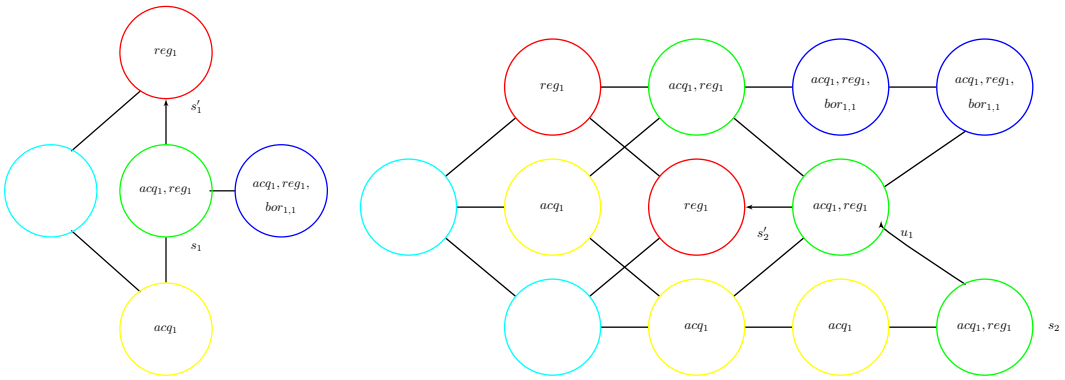


Figure 6.4: Partitioning of S_1 and S_2 into *stuttering bisimilar* classes of states w.r.t. $\mathcal{R}_{1,1}$

let state $s_1 \in S_1$ and state $s_2 \in S_2$ such that $(s_1, s_2) \in \mathcal{R}_{1,1}$ as depicted in Figure 6.4. State s_1 transits to s'_1 via a (DIS) operation of book b_1. Following the description of 2(b)i, state s_2 cannot mimic the behaviour of s_1 directly, because book 1 is currently lent to member 2, i.e. $bor_{1,2} \in L_2(s_2)$ (see Figure 6.3), which deters the book from being discarded by the library. This, however, triggers the *stuttering* of s_2 via a (RET) operation to state $u_1 \in S_2$, which suggests that member 2 return book 1 to the library. Once the book is returned to the library, it may now be discarded. In other words, the system may now transit from u_1 to $s'_2 \in S_2$ with $(s'_1, s'_2) \in \mathcal{R}_{1,1}$.

Note that the *cut-off* value for p is 1 as the lowest value that can be assigned to parameter p for which TS_p and TS_{p+1} is 1. Hence, the system requirements of Section 6.3.2 can be verified on TS_1, which turns out to be an important result from a model-checking point of view.

6.3.6 Modified System Specifications

The goal of this section is to bring some light on the area regarding the necessary conditions PTS specifications should fulfil so that consecutive instances of the system namely TS_p and TS_{p+1} can be proven *stuttering* bisimilar. To this end, we consider the following modifications of the library management system:

1. **Favourizing members**. As a first case, we assume that the library favourizes a registered member against the rest of its registered members. For argument's sake, we suppose that member $p+1$ is entitled to keep any book he borrows for an indeterminate period of time or, further simplifying the case, $p+1$ is entitled to keep any book he borrows forever. Notice that $p+1$ does not take part in TS_p at all. Transition relation $\rightarrow_p$ is, then, defined

as follows:

$$
\begin{array}{rl}
\rightarrow_p = & \{\ (s,s') \mid s = \langle A,B,M\rangle,\ s' = \langle A',B',M'\rangle \text{ such that} \\
& (\forall i \in 1..m,\ \{i\} \cap A = \emptyset\ \wedge\ mod_A(A \cup \{i\})) \\
\otimes & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_A(A \setminus \{i\})) \\
\otimes & (\forall j \in 1..p,\ \{j\} \cap M = \emptyset\ \wedge\ mod_M(M \cup \{j\})) \\
\otimes & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_M(M \setminus \{j\})) \\
\otimes & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_B(B \cup \{(i,j)\})) \\
\otimes & (\forall i \in A,\ \forall j \in M,\ (i,j) \in B\ \wedge\ \mathbf{j} \neq \mathbf{p+1}\ \wedge\ mod_B(B \setminus \{(i,j)\})) \\
\otimes & (\mathbf{A'} = \mathbf{A}\ \wedge\ \mathbf{B'} = \mathbf{B}\ \wedge\ \mathbf{M'} = \mathbf{M})\ \}
\end{array}
$$

The principal difference between this version of the library management system and the initial model of Section 6.3.3 lies in Rules (6.15) and (6.8), which constitute (RET) operations. In particular, member $p+1$ is banned from taking any (RET) operation. Notice also the presence of Rule (6.15), which is added to ensure that in case of blocking, the system state remains unchanged. This effect is represented by self-loops to all states in Figure 6.3. These simple modification alter the system behaviour considerably. Transition $s_1 \rightarrow_p s_1'$ of TS_p, which denotes discarding book b_1 from the library, cannot be mimicked by any *stuttering* path of TS_{p+1} starting from $s_2 \in TS_{p+1}$. The *stuttering* bisimulation relation of TS_p and TS_{p+1} cannot be established, and, thus, the preservation of PARCTL is not guaranteed. We conclude that concrete values of the system parameter affecting the control flow of the system may potentially violate the *stuttering* bisimulation relation of TS_p and TS_{p+1}.

2. **Extending loans**. As a second case, we assume that the library extends the time a book may be on loan. This may be viewed as a gesture of goodwill to the current borrower, whose total number of book loans has reached a predefined value. For simplicity's sake, we set this value to $n \in \mathbb{N}$ and we suppose that the book loan extension to the current borrower is for life. For this, we need a variable nbl to keep record of the total number of loans carried out by each member. The set $N = \cup_{j=1}^{p}\{nbl_j = k |\ k \in \mathbb{N}\}$ codes this information set-theoretically. Hence, the proposition labelling of TS_p is defined as follows:

$$L_p(s) = \{acq_i |\ i \in A\}\ \cup\ \{reg_j |\ j \in M\}\ \cup\ \{bor_{i,j} |\ (i,j) \in B\}\ \cup\ N,$$

the set of states in TS_p is represented as $S_p = \{\ s \mid L_p(s) = \langle A,B,M,N\rangle\ \}$ and the initial state of S_p is defined as $s_p^0 = \langle \emptyset,\emptyset,\emptyset,\{nbl_1 = 0,\ldots,\ nbl_p = 0\}\rangle$. Finally, transition relation $\rightarrow_p$ is modified appropriately:

$$
\begin{array}{rcl}
\rightarrow_p & = & \{\ (s,s') \mid s = \langle A,B,M,N\rangle,\ s' = \langle A',B',M',N'\rangle \text{ such that} \\
& & (\forall i \in 1..m,\ \{i\} \cap A = \emptyset\ \wedge\ mod_A(A \cup \{i\})\ \wedge\ \mathbf{N'} = \mathbf{N}) \\
& \otimes & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_A(A \setminus \{i\})\ \wedge\ \mathbf{N'} = \mathbf{N}) \\
& \otimes & (\forall j \in 1..p,\ \{j\} \cap M = \emptyset\ \wedge\ mod_M(M \cup \{j\})\ \wedge\ \mathbf{N'} = \mathbf{N}) \\
& \otimes & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_M(M \setminus \{j\})\ \wedge\ \mathbf{N'} = \mathbf{N}) \\
& \otimes & (\forall i \in A,\ \forall j \in M,\ \{(i,j)\} \cap B = \emptyset\ \wedge\ mod_B(B \cup \{(i,j)\})\ \wedge\ \mathbf{nbl'_j} = \mathbf{nbl_j + 1}) \\
& \otimes & (\forall i \in A,\ \forall j \in M,\ (i,j) \in B\ \wedge\ \mathbf{nbl_j} < \mathbf{n}\ \wedge\ mod_B(B \setminus \{(i,j)\})\ \wedge\ \mathbf{N'} = \mathbf{N}) \\
& \otimes & (\mathbf{A'} = \mathbf{A}\ \wedge\ \mathbf{B'} = \mathbf{B}\ \wedge\ \mathbf{M'} = \mathbf{M}\ \wedge\ \mathbf{N'} = \mathbf{N})\ \}
\end{array}
$$

Rule (6.15), which is a (RET) operation, increments the total number of book loans nbl_j taken by member j so far every time j takes a new loan. Finally, if nbl_j has reached value n, Rule (6.15) bans (RET) operations from taking place. Let's imagine that $p+1$ has taken n book loans so far and is the current borrower for book 1. Hence, as is the case for case 1, transition $s_1 \rightarrow_p s_1'$ cannot be mimicked by any *stuttering* path of TS_{p+1} starting from $s_2 \in TS_{p+1}$.

6.4 Parametric ISs and PARCTL

In this section, we consider the PMCP for ISs specified in EB^3. Based on the fact that the EB^3 models are translated to LTS models, we apply the results of Section 6.3.6 directly to the EB^3 models. First, we conclude that the control flow of the IS in question cannot be affected by concrete values of the system parameter (the book ID in the case of the library management system). Furthermore, the exact nature of the assigned values to the system parameter cannot change the control flow. In other words, the system cannot "look into" the values of the system parameter. To this respect, only equalities and inequalities with abstract values of the system parameter are allowed. For example, the classic function *card* (found in the library management specification) that returns the cardinality of sets relating to the system parameter is not allowed.

We sum up these results into the following precise criteria, which are necessary (but not sufficient) conditions for the preservation of PARCTL:

1. Only abstract values of system parameters can appear in the body of EB^3 expressions. The existence of concrete values would potentially break system symmetry.

2. Operations on system parameters can only involve abstract values. The outcome of these operations is by no means affected by concrete values.

3. The only predicates allowed containing abstract values of system parameters are equalities and inequalities.

4. Operations involving the size of sets pertaining to system parameters are strictly forbidden.

6.5 Evaluation and Related Work

The technique developed in this chapter permits efficient model-checking of PARCTL properties over EB^3 models. It is fairly straightforward to verify if an EB^3 specification satisfies the conditions of Section 6.4. However, the satisfaction of the aforementioned conditions does not mean that the EB^3 models in question constitute *parametric* ISs. This is one of the weak points of our approach. The proof that TS_p and TS_{p+1} are *stuttering bisimilar* has to be done manually. Once the proof is completed, the user must determine the *cut-off*. For the time being, there is no automated technique that calculates the *cut-off*, but it should be fairly easy to determine the *cut-off* by simple observation of the EB^3 specification and the devised proof of correspondence. For example, this should be equal to the different number of values of the system parameter appearing syntactically in EB^3's guards. In the guard "$m_1 = m_2 \wedge m_2 \neq m_3 \Rightarrow Lend(b_1, m_3)$" extracted from a modified version of the library

management system's specification, where m_1, m_2, m_3 are variables denoting members, the *cut-off* is three. As soon as the *cut-off* is determined, the user may use EB^{3}2LNT in order to translate the EB^3 model in question, whose system parameter is instantiated to the *cut-off* value, to its equivalent LNT model and verify any temporal properties expressed in MCL. Note that MCL allows the specification of action-based properties, whereas PARCTL is a state-based calculus. Thankfully, the correctness of this approach is supported by the theoretic results of [DV90] guarantee.

We note that our technique was inspired by the standard abstraction techniques of PMC [CGB86, EN95], which are nonetheless restricted to systems expressed as LTSs. The work closest to ours can be found in [Laz99], where the author addresses the PMCP in the context of process algebra CSP and determines *cut-offs* for ISs expressed in CSP. The model-checking tool related to CSP is FDR [Bro00] and the verification is based on refinement-checking, where the user expresses the temporal property as a process that *refines* the process related to the CSP model (the execution trace of the process related to the property is a subset of the execution trace of the CSP model). This is, in our humble view, the drawback of this approach, since temporal properties should all be expressed as CSP processes.

6.6 Conclusion

We addressed the PMCP in the context of ISs specified in EB^3. First, we showed how the standard theory on PMC can be applied to the library management system on a merely state-based level and specified the conditions under which EB^3 models, whose behaviour depends on a system parameter, preserve the the temporal logic PARCTL.

As future work, it is obviously of great practical importance to determine the *cut-offs* for these system parameters, and to find as small a threshold value as possible. It would also be interesting to investigate and develop abstraction techniques based on abstract interpretation [CC77], or even find a relation between PMC and abstract interpretation. Another interesting issue to study would be multi-parameter systems, i.e. parametric systems with multiple system parameters in the lines of [HSB$^+$10].

Chapter 7

Conclusion

In this thesis, we proposed an automatic approach for equipping the EB^3 method with formal verification capabilities by reusing already available model checking technology. Our work was motivated by the need for efficient model-checking techniques adapted to IS specifications. This task was a priory hard, given that ISs are complex systems by nature involving heavy data management and non-trivial concurrency. The project became all the more complicated as soon as we recognized the importance of pursuing a merely automatic approach, which would apply globally for every EB^3 specification without user intervention. In our humble opinion, this is the strong point of our method compared to other relevant approaches in the field that, despite being automatic, are restricted to the verification of *safety* properties [GFL05, GFL06] or, despite offering verification capabilities for the full spectrum of temporal properties, require user intervention [FFC$^+$10, ETL$^+$04].

Our approach relied upon a new translation from EB^3 to LNT, which provides a direct connection to all the state-of-the-art verification features of the CADP toolbox. Despite the relatively many common features between EB^3 and LNT, we were obliged to reconciliate the standard traced-based semantics of EB^3 [FSt03] with the typical state-based semantics of LNT, which led us to define EB^3's semantics alternatively [VD13]. The translation, based on EB^3's new alternative memory semantics, was automated by the EB^{3}2LNT translator and validated on several examples of typical ISs. So far, we experimented only the model checking of MCL data-based temporal properties on EB^3 specifications. However, CADP also provides extensive support for equivalence checking and compositional LTS construction, which can be of interest to IS designers. We also provided a formal proof of the translation from EB^3 to LNT, which could serve as reference for translating EB^3 to other process algebras in the future.

It should be noted that EB^{3}2LNT offers insight to interested readers on how to code global state as a process running parallel to the principal system specification, an approach that comes in very handy when state-based IS specifications are studied. On the other hand, the main limitation of the proposed translation is that model-checking becomes intractable for specifications describing multiple concurrent processes as is the case for the library specification for more than four members and four books. This inconvenience is attributed to the fact that CADP employs explicit techniques instead of symbolic techniques for state space generation. Another drawback of the proposed translation arises at the stage of property verification. Properties should be expressed in MCL, which implies that EB^3 users should be familiar with MCL's syntax. Also, user intervention is indispensable at this level.

Chapter 6 of the present thesis was concerned with improving the model checking of EB^3 specifications with the aid of abstraction techniques. Based on the inherent symmetry of EB^3 models, we concentrated on PISs namely ISs, whose behaviour is scaled in keeping with the predefined value of some system parameter. The goal of this approach was to effectively cut down on the number of components participating in the IS model, while making sure that the temporal property we wish to verify is preserved in the reduced model. For the time being, our approach requires user intervention so that the equivalence between initial and reduced model is established.

As future work, we will direct our attention to abstraction techniques with the purpose of improving and making automatic the PMC approach. In particular, we will study further how the insertion of new functionalities into an IS affects the issue, and we will formalize this in the context of EB^3 specifications.

Chapter 8

Appendix

8.1 LNT Code for the Simplified Library Management System

We give the optimized LNT code for the simplified Library Management System of Chapter 4, with 2 members, 1 book, and *NbLoans* set to 1.

```
module library is

type mId is m1, m2, m⊥ with "eq","ne","ord","val" end type
type BID is b1, b⊥ with "eq","ne","ord","val" end type
type NB is array[0..2] of NAT end type
type BOR is array[0..1] of MID end type

process M [Acquire, Discard, Register, Unregister, Lend, Return : any] is
  var mId : MID, bId : BID, bId' : BID,
      mId' : MID, borrower : BOR, nbLoans : NB in
      mId := m⊥;   borrower := BOR(m⊥);   nbLoans := NB(0);
  loop
    select
          Acquire (?bId)
    []    Discard (?bId, ?borrower)
    []    Register (?mId)
    []    Unregister (?mId)
    []    Lend (?bId, ?mId, !nbLoans, !borrower);
          borrower [ord (bId)] := mId;
          nbLoans [ord (mId)] := nbLoans [ord (mId)] + 1
    []    RET (?bId);
          mId' := borrower [ord (bId)];
          nbLoans [ord (mId')] := nbLoans [ord (mId')] − 1;
          borrower [ord (bId)] := m⊥
    end select
  end loop
  end var
end process

process loan [Lend, Return : any] (mId : mId, bId : BID) is
  var borrower : BOR, nbLoans : NB in (* NbLoans is set to 1 *) in
    Lend (bId, mId, ?nbLoans, ?borrower)
      where ((borrower [ord (bId)] eq m⊥) and (nbLoans [ord (mId)] eq 1));
    Return (bId)
  end var
end process

process book [Acquire, Discard : any] (bId : BID) is
  var borrower : BOR in
    Acquire(bId); Discard (bId, ?borrower) where (borrower [ord (bId)] eq m⊥)
  end var
end process
```

```
process member [Register, Unregister, Lend, Return : any] (mId : mId) is
  Register (mId);
  loop L in
    select break L [] loan [Lend, Return] (mId, b1)
    end select
  end loop;
  Unregister (mId)
end process
process Main [Acquire, Discard, Register, Unregister, Lend, Return : any] is
  par Acquire, Discard, Register, Unregister, Lend, Return in
    par
      loop L in
        select break L [] book [Acquire, Discard] (b1)
        end select
      end loop
    ||
      par
        loop L in
          select break L [] member [Register, Unregister, Lend, Return] (m1)
          end select
        end loop
      ||
        loop L in
          select break L [] member [Register, Unregister, Lend, Return] (m2)
          end select
        end loop
      end par
    end par
  ||
    M [Acquire, Discard, Register, Unregister, Lend, Return]
  end par
end process
end module
```

8.2 EB^3 Code for the Extended Library Management System

We give the EB^3 code for the extended Library Management System of Section 5.2, with 2 members ($MID = \{m_1, m_2\}$), 2 books ($BID = \{b_1, b_2\}$), and *NbLoans* set to 1. Functions "*add_reservation*","*cancel_reservation*", "*last_reservation*", "*nil_reservation*" and

"*is_reserved*" are standard functions on lists. They are not attribute functions. Their definitions are given in the corresponding LNT specification.

```
Acquire (bId : BID);
Discard (bId : BID);
Register (mId : mId);
Unregister (mId : mId);
Lend (bId : BID, mId : mId);
Take (bId : BID, mId : mId);
Transfer (mId : mId, bId : BID);
Reserve (mId : mId, bId : BID);
Cancel (mId : mId, bId : BID);
Return (bId : BID);
```

```
nbLoans (T : T, mId : MID) : Nat⊥ =
    match last (T) with
        ⊥T : ⊥
    |   Register (mId) : 0
    |   Lend (bId, mId) :
            nbLoans (front (T), mId) + 1
    |   Take (bId, mId) :
            nbLoans (front (T), mId) + 1
    |   Return (bId) :
          if mId = borrower (T, bId) then
            nbLoans (front (T), mId) − 1
    |   Unregister (mId) : ⊥
    |   Transfer (mId′, bId) :
          if mId = borrower (T, bId) then
            nbLoans (front (T), mId) + 1
    |   Unregister (mId) : nbLoans (front (T), mId)
    end match;
```

```
borrower (T : T, bId : BID) : mId⊥ =
    match last (T) with
        ⊥T : ⊥
    |   Lend (bId, mId) : mId
    |   Take (bId, mId) : mId
    |   Transfer (mId, bId) : ⊥
    |   _ : nbLoans (front (T), mId)
    end match;

acquired (T : T, bId : BID) : BOOL =
    match last (T) with
        ⊥T : false
    |   Acquire (bId) : true
    |   Discard (bId) : false
    |   _ : acquired (front (T), bId)
    end match;
```

```
reservation (T : T, bId : BID) =
    match last (T) with
        ⊥T : NIL
    |   Reserve (mId, bId) : add_reservation(mId, reservation (front (T), bId))
    |   Take (bId, mId) : cancel_reservation(mId, reservation (front (T), bId))
    |   Cancel (bId, mId) : cancel_reservation(mId, reservation (front (T), bId))
        _ : reservation (front (T), mId)
    end match;
```

$$
\begin{array}{l}
book\,(bId : BID) = Acquire\,(bId).\\
\quad (borrower\,(\mathsf{T},\ bId) = \bot) \wedge (nil\_reservation\ (reservation\ (\mathsf{T},\ bId))) \Rightarrow Discard\,(bId);\\
member\,(mId : MID) =\\
\quad (acquired\,(\mathsf{T},\ bId) = true) \wedge (borrower\,(\mathsf{T},\ bId) = \bot) \wedge\\
\quad (nbLoans\,(\mathsf{T}, mId) < NbLoans) \wedge\\
\quad (nil\_reservation\ (reservation\ (\mathsf{T},\ bId)) = true) \quad \Rightarrow \quad Lend\,(bId, mId).\\
\quad (\quad (acquired\,(\mathsf{T},\ bId) = true) \wedge (borrower\,(\mathsf{T},\ bId) = \bot) \wedge\\
\qquad (nil\_reservation\ (reservation\ (\mathsf{T},\ bId)) = true) \quad \Rightarrow \quad Renew\,(bId) \quad)^*.\\
\qquad\quad (acquired\,(\mathsf{T},\ bId) = true) \wedge (borrower\,(\mathsf{T},\ bId) = mId) \quad \Rightarrow \quad Return\,(bId)\\
\qquad\qquad\qquad\qquad\qquad \mid\\
\qquad\quad (\mid mId' : MID \setminus \{mId\} :\\
\qquad\qquad (acquired\,(\mathsf{T},\ bId) = true) \wedge (borrower\,(\mathsf{T},\ bId) \neq mId') \wedge\\
\qquad\qquad (nbLoans\,(\mathsf{T}, mId) > 0) \wedge (nbLoans\,(\mathsf{T}, mId) < NbLoans) \wedge\\
\qquad\qquad (nil\_reservation\ (reservation\ (\mathsf{T},\ bId)) = true) \quad \Rightarrow \quad Transfer\,(mId', bId))\\
\qquad\qquad\qquad\qquad \mid\\
\quad (borrower\,(\mathsf{T},\ bId) \neq \bot) \wedge (borrower\,(\mathsf{T}, bId) \neq mId) \wedge\\
\quad (acquired\,(\mathsf{T},\ bId) = true) \wedge\\
\quad (is\_reserved\ (bId, reservation\ (\mathsf{T},\ bId)) = false) \quad \Rightarrow \quad Reserve\,(mId, bId).\\
\qquad\quad (acquired\,(\mathsf{T},\ bId) = true) \wedge (borrower\,(\mathsf{T},\ bId) = \bot) \wedge\\
\qquad\quad (nbLoans\,(\mathsf{T},\ bId) < NbLoans) \wedge\\
\qquad\quad (last\_reservation\ (mId, reservation\ (\mathsf{T},\ bId)) = false) \quad \Rightarrow \quad Take\,(mId, bId)\\
\qquad\qquad\qquad\qquad\qquad \mid\\
\qquad\quad Cancel\,(mId, bId)\\
\qquad\qquad\qquad\qquad \mid\\
\quad (acquired\,(\mathsf{T},\ bId) = true) \wedge (borrower\,(\mathsf{T},\ bId) = mId) \quad \Rightarrow \quad Return\,(bId)\\
\qquad\qquad\qquad\qquad \mid\\
\quad (\mid mId' : MID \setminus \{mId\} :\\
\qquad (acquired\,(\mathsf{T},\ bId) = true) \wedge (borrower\,(\mathsf{T},\ bId) = mId) \wedge\\
\qquad (nbLoans\,(\mathsf{T}, mId) > 0) \wedge (nbLoans\,(\mathsf{T}, mId) < NbLoans) \wedge\\
\qquad (nil\_reservation\ (reservation\ (\mathsf{T},\ bId)) = true) \quad \Rightarrow \quad Transfer\,(mId', bId));
\end{array}
$$

$$
\begin{array}{l}
main =\\
\quad (\mid\mid\mid bId : BID : book\,(bId)^*) \mid\mid\mid\\
\quad (\mid\mid\mid mId : MID : (Register\,(mId).\ (\mid\mid\mid bId : BID : member\,(mId,\ bId)^*).\\
\qquad (nbLoans\,(\mathsf{T}, mId) = 0) \wedge (is\_reserved\ (mId,\ reservation\ (\mathsf{T},\ b_1)) = false) \wedge\\
\qquad (is\_reserved\ (mId,\ reservation\ (\mathsf{T},\ b_2)) = false) \quad \Rightarrow \quad Register\,(mId))^*)
\end{array}
$$

8.3 LNT Code for the Extended Library Management System

We give the LNT code for the extended Library Management System ($NbLoans = 1$). This code is generated by EB^{3}2LNT except the functions "*add_reservation*","*cancel_reservation*", "*last_reservation*", "*nil_reservation*" and "*is_reserved*", which are coded by hand, since our compiler does not support user-defined EB^3 types and (non-attribute) functions. We also define type "*RESERV*" to simplify the coding. By convention, for every type T we have $ord(\perp) = 0$, $ord(first_T) = 1$, etc.

```
module library is

type mId is m1, m2, m⊥ with "eq", "ne", "ord", "val" end type
type BID is b1, b2, b⊥ with "eq", "ne", "ord", "val" end type
type MIDLIST is
  NIL, CONS(HD : MID, TL : MIDLIST) with "eq","ne" end type
type ACQUIR is array[0..1] of BOOL end type
type BOR is array[0..1] of MID end type
type NB is array[0..1] of NAT end type
type RESERV is array[0..1] of MIDLIST end type

function add_reservation (m : MID, l : MIDLIST) : MIDLIST is
  case l in var temp_mem : MID, temp_list : MIDLIST in
      NIL → return CONS(m, NIL)
    | CONS(temp_mem, temp_list) →
        return CONS(temp_mem, add_reservation(m, temp_list))
  end case
end function

function cancel_reservation (m : MID, l : MIDLIST) : MIDLIST is
  case l in var temp_mem : MID, temp_list : MIDLIST in
      NIL → return NIL
    | CONS(temp_mem, temp_list) where (temp_mem eq m) → return temp_list
    | CONS(temp_mem, temp_list) →
        return CONS(temp_mem, cancel_reservation(m, temp_list))
  end case
end function

function nil_reservation (l : MIDLIST) : BOOL is
  case l in var temp_list : MIDLIST in
      NIL → return true
    | ANY MIDLIST → return false
  end case
end function
```

```
function is_reserved (m : MID, l : MIDLIST) : MIDLIST is
  case l in var temp_mem : MID, temp_list : MIDLIST in
      NIL → return false
    | CONS(temp_mem, temp_list) where (temp_mem eq m) → return true
    | CONS(temp_mem, temp_list) → return is_reserved(m, temp_list)
  end case
end function

function last_reservation (m : MID, l : MIDLIST) : BOOL is
  case l in var temp_mem : MID, temp_list : MIDLIST in
      NIL → return false
    | CONS(temp_mem, temp_list) where (temp_mem eq m) → return true
    | CONS(temp_mem, temp_list) → return false
  end case
end function
```

```
process M [Acquire, Discard, Register, Lend, Take, Renew, Return, Reserve,
           Cancel, Unregister, Transfer : any] is
  var mId : mId, bId : BID, acquired : ACQUIR,
      borrower : BOR, nbLoans : NB, reservation : RESERV in
      acquired := ACQUIR(false);   borrower := BOR(m⊥);
      nbLoans := NB(0);   reservation : RESERV(NIL);
  loop
    select
        Acquire (?bId);
        acquired [ord (bId)] := true
      [] Discard (?bId, !borrower, !reservation);
        acquired [ord ()] := false
      [] Register (?mId);
        nbLoans [ord (mId)] := 1
      [] Unregister (?mId !nbLoans !reservation)
        nbLoans [ord (mId)] := 0
      [] Lend (?bId, ?mId, !acquired, !borrower, !reservation, !nbloans);
        borrower [ord (bId)] := mId;
        nbLoans [ord (mId)] := nbLoans [ord (mId)] + 1
      [] Reserve (?mId, ?bId, !acquired, !borrower, !reservation);
        reservation [ord (bId)] := add_reservation(mId, reservation [ord (bId)])
      [] Take (?mId, ?bId, !acquired, !borrower, !reservation, !nbLoans);
        borrower [ord (bId)] := mId;
        nbLoans [ord (mId)] := nbLoans [ord (mId)] + 1;
        reservation [ord (bId)] := cancel_reservation(mId, reservation [ord (bId)])
      [] Cancel (?mId, ?bId);
        reservation [ord (bId)] := cancel_reservation(mId, reservation [ord (bId)])
      [] Transfer (?mId, ?bId, !acquired, !borrower, !reservation, !nbLoans);
        nbLoans [ord (borrower [ord (bId)])] := nbLoans [ord (borrower [ord (bId)])] − 1;
        borrower [ord (bId)] := mId;
        nbLoans [ord (mId)] := nbLoans [ord (mId)] + 1
      [] Renew (?bId, !acquired, !borrower, !reservation)
      [] Return (?bId, !acquired, !borrower);
        nbLoans [ord (borrower [ord (bId)])] := nbLoans [ord (borrower [ord (bId)])] − 1;
        borrower [ord (bId)] := m⊥
  end loop
  end var
end process
```

```
process book [Acquire, Discard : any] (bId : BID) is
  var acquired : ACQUIR, borrower : BOR, reservation : RESERV in
    Acquire(bId); Discard (bId, ?borrower, ?reservation)
      where ((borrower [ord (bId)] eq m⊥) and
             (nil_reservation (reservation[ord (bId)]) eq true))
  end var
end process

process member [Lend, Take, Renew, Return, Reserve, Cancel, Transfer : any]
                (mId : MID, bId : BID) is
  var acquired : ACQUIR, borrower : BOR, reservation : RESERV, nbLoans : NB in
    select
      Lend (!bId, !mId, ?acquired, ?borrower, ?reservation, ?nbloans)
        where ((acquired [ord (bId)] eq true) and (borrower[ord (bId)] eq m⊥) and
               (nbLoans[ord (mId)] lt 2) and
               (nil_reservation (reservation [ord (bId)] eq true))
      loop L in select break L []
          Renew (!bId, ?acquired, ?borrower, ?reservation)
            where ((acquired [ord (bId)] eq true) and (borrower[ord (bId)] eq mId) and
                   (nil_reservation (reservation [ord (bId)]) eq true))
      end select end loop;
      select
          Return (!bId, ?acquired, ?borrower)
            where ((acquired [ord (bId)] eq true) and (borrower[ord (bId)] eq mId))
      []  var mId' : MID in
            mId' := any MID where ((mId' ne m⊥) and (mId' ne m⊥));
            Transfer (!mId', !bId, ?acquired, ?borrower, ?reservation, ?nbLoans)
              where ((acquired [ord (bId)] eq true) and (borrower [ord (bId)] eq mId) and
                     (nbLoans [ord (mId')] gt 0) and (nbLoans [ord (mId')] lt 2) and
                     (nil_reservation (reservation [ord (bId)]) eq true))
          end var
      end select
```

```
[]  Reserve (!mId, !bId, ?acquired, ?borrower, ?reservation, ?nbLoans)
      where ((acquired [ord (bId)] eq true) and
             (borrower[ord (bId)] ne m⊥) and (borrower[ord (bId)] ne mId) and
             (is_reserved (mId, reservation [ord (bId)]) eq false);
    select
      Take (!mId, !bId, ?acquired, ?borrower, ?reservation, ?nbloans)
        where ((acquired [ord (bId)] eq true) and
               (borrower[ord (bId)] eq m⊥) and (nbLoans[ord (mId)] lt 2) and
               (last_reservation (mId, reservation [ord (bId)]) eq true))
    []  Cancel (!mId, !bId)
    end select
[]  Return (!bId, ?acquired, ?borrower)
      where ((acquired [ord (bId)] eq true) and (borrower[ord (bId)] eq mId))
[]  var mId' : MID in
      mId' := any MID where ((mId' ne m⊥) and (mId' ne m⊥));
      Transfer (!mId', !bId, ?acquired, ?borrower, ?reservation, ?nbLoans)
        where ((acquired [ord (bId)] eq true) and (borrower [ord (bId)] eq mId) and
               (nbLoans [ord (mId')] gt 0) and (nbLoans [ord (mId')] lt 2) and
               (nil_reservation (reservation [ord (bId)]) eq true))
    end var
  end select
end var
end process

process Main [Register, Lend, Take, Renew, Return, Reserve, Cancel, Unregister,
              Acquire, Discard : any] is
  par Register, Lend, Take, Renew, Return, Reserve, Cancel, Unregister, Acquire, Discard in
    par
      par
        loop L1 in select break L1 []
          book [Acquire, Discard] (b1)
        end select end loop
      ||
        loop L1 in select break L1 []
          book [Acquire, Discard] (b2)
        end select end loop
      end par
```

```
||
  par
    loop L1 in select break L1 [] 
      Register(m1);
      par
        loop L2 in select break L2 []
          member [Lend, Take, Renew, Return, Reserve, Cancel, Transfer] (m1, b1)
        end select end loop
      ||
        loop L2 in select break L2 []
          member [Lend, Take, Renew, Return, Reserve, Cancel, Transfer] (m1, b2)
        end select end loop;
      end par;
      var nbLoans : NB, bId : BID, reservation : RESERV in
        Unregister (m1, ?nbLoans, ?reservation)
          where ((nbLoans [ord (m1)] eq 1) and
                 (is_reserved (m1, reservation [ord (b1)]) eq false) and
                 (is_reserved (m1, reservation [ord (b2)]) eq false))
      end var
    end select end loop
  ||
    loop L1 in select break L1 []
      Register(m2);
      par
        loop L2 in select break L2 []
          member [Lend, Take, Renew, Return, Reserve, Cancel, Transfer] (m2, b1)
        end select end loop
      ||
        loop L2 in select break L2 []
          member [Lend, Take, Renew, Return, Reserve, Cancel, Transfer] (m2, b2)
        end select end loop
      end par;
      var nbLoans : NB, bId : BID, reservation : RESERV in
        Unregister (m2, ?nbLoans, ?reservation)
          where ((nbLoans [ord (m2)] eq 1) and
                 (is_reserved (m2, reservation [ord (b1)]) eq false) and
                 (is_reserved (m2, reservation [ord (b2)]) eq false))
      end var
    end select end loop;
  end par
```

```
  ||
    M [Acquire, Discard, Register, Lend, Take, Renew, Return, Reserve, Cancel, Unregister,
       Transfer]
  end par
end process

end module
```

8.4 MCL Formulas for Requirements R1 to R15

Requirement R1 *"A book can always be acquired by the library when it is not currently acquired"*

```
macro R1 (B) =
  (
    [ (not {ACQUIRE !B})* ] ⟨ {ACQUIRE !B} ⟩ true
    and
    [ true* . {DISCARD !B} . (not {ACQUIRE !B})* ] ⟨ {ACQUIRE !B} ⟩ true
  )
end_macro
R1 ("B1") and R1 ("B2") and R1 ("B3")
```

This is a classical liveness property. The second conjunct of "$R_1\,(B)$" expresses the eventuality that a book be withdrawn from the library before it is reacquired.

Requirement R3 *"An acquired book can be discarded only if it is neither borrowed nor reserved"*

```
[ true* . {?G : string ?any : string ?B : string where (G = "LEND") or (G = "TAKE")} .
  (not {RETURN !B})* . {DISCARD !B} ] false
and
[ true*.{RESERVE ?any : string ?B : string} .
  (not ({CANCEL ?any : string !B} or {RETURN !B}))* . {DISCARD !B} ] false
```

Requirement R4

"*A person must be a member of the library in order to borrow a book*"

macro $R_4\,(M) =$
(
[(**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
$R_4\,$(`"M1"`) **and** $R_4\,$(`"M2"`) **and** $R_4\,$(`"M3"`)

The first conjunct or "$R_4\,(M)$" 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 action `LEND` nor the reservation action `RESERVE`.

Requirement R5

"*A book can be reserved only if it has been borrowed or already reserved by some member*"

macro $R_5\,(B) =$
(
[(**not** ({`LEND ?any : string !`B} **or** {`TAKE ?any : string !`B}))* .
{`RESERVE ?any : string !`B}] **false**
and
[**true*** . {`RETURN !`B} .
(**not** ({`LEND ?any : string !`B} **or** {`TAKE ?any : string !`B}))* . {`RESERVE !`B}] **false**
and
[(**not** ({`LEND ?any : string !`B} **or** {`TAKE ?any : string !`B} **or**
{`TRANSFER ?any : string !`B} **or** {`RESERVE ?any : string !`B}))* .
{`RESERVE ?any : string !`B}] **false**
)
end_macro
$R_5\,$(`"B1"`) **and** $R_5\,$(`"B2"`) **and** $R_5\,$(`"B3"`)

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 "{ `RETURN !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.

Requirement R6 "*A book cannot be reserved by the member who is borrowing it*"

[**true***.{`LEND ?`M `: string ?`B `: string`} .
(**not** ({`RETURN !`B} **or** {`TRANSFER ?`$M2$ `: string !`B}))* . {`RESERVE !`M `!`B}] **false**

The difficulty here lies in the fact that the borrower may transfer the book to another member. For this reason, the following formula is false.

$$[\,\mathbf{true}^*\,.\,\{\texttt{LEND ?}M : \mathbf{string}\ \texttt{?}B : \mathbf{string}\}\,.\,(\mathbf{not}\ (\{\texttt{RETURN !}B\}))^*\,.\,\{\texttt{RESERVE !}M\ \texttt{!}B\}\,]\ \mathbf{false}$$

Requirement R8 *"A book cannot be lent to a member if it is reserved"*

$$\begin{array}{l}\mathbf{macro}\ R_8\,(B, M_1, M_2) = \\ \quad (\\ \quad\quad [\,\mathbf{true}^*\,.\,\{\texttt{RESERVE !}M_1\ \texttt{!}B\}\,.\,(\mathbf{not}\ (\{\texttt{TAKE !}M_1\ \texttt{!}B\}\ \mathbf{or}\ \{\texttt{CANCEL !}M_1\ \texttt{!}B\}))^*\,. \\ \quad\quad\quad \{\texttt{LEND !}M_2\ \texttt{!}B\}\,]\ \mathbf{false} \\ \quad) \\ \mathbf{end\_macro} \\ R_8\,(\texttt{"B1"}, \texttt{"M1"}, \texttt{"M2"})\end{array}$$

In this case (as well as for the subsequent requirements R9, R11, R13, R14, and R15), we only check the property for a specific book (`B1`) and members (`M1`, `M2`). So doing, we exploit the symmetry of the specification (all books and members have similar behaviour), which is crucial to avoid the exponential state space explosion.

Requirement R9 *"A member cannot renew a loan or give the book to another member if the book is reserved"*

$$\begin{array}{l}\mathbf{macro}\ R_9\,(B, M) = \\ \quad (\\ \quad\quad [\,\mathbf{true}^*\,.\,\{\texttt{RESERVE !}M\ \texttt{!}B\}\,.\,(\mathbf{not}\ (\{\texttt{TAKE !}M\ \texttt{!}B\}\ \mathbf{or}\ \{\texttt{CANCEL !}M\ \texttt{!}B\}))^*\,.\,\{\texttt{RENEW !}B\}\,]\ \mathbf{false} \\ \quad) \\ \mathbf{end\_macro} \\ R_9\,(\texttt{"B1"}, \texttt{"M1"})\end{array}$$

Requirement R10 *"A member is allowed to take a reserved book only if he owns the oldest reservation"*

$$\begin{array}{l}[\\ \quad \mathbf{true}^*\,. \\ \quad \{\texttt{RESERVE ?}M_1 : \mathbf{string}\ \texttt{?}B : \mathbf{string}\}\,. \\ \quad (\mathbf{not}\ (\{\texttt{TAKE !}M_1\texttt{!}B\}\ \mathbf{or}\ \{\texttt{CANCEL !}M_1\texttt{!}B\}\ \mathbf{or}\ \{\texttt{TRANSFER !}M_1\texttt{!}B\}))^*\,. \\ \quad \{\texttt{RESERVE ?}M_2 : \mathbf{string}\ \texttt{!}B\ \mathbf{where}\ M_2 \neq M_1\}\,. \\ \quad (\mathbf{not}\ (\{\texttt{TAKE !}M_1\texttt{!}B\}\ \mathbf{or}\ \{\texttt{CANCEL !}M_1\texttt{!}B\}\ \mathbf{or}\ \{\texttt{TRANSFER !}M_1\texttt{!}B\}))^*\,. \\ \quad \{\texttt{TAKE !}M_2\texttt{!}B\} \\ ]\ \mathbf{false}\end{array}$$

This property has been rephrased in the following way: If two members reserve a book, the first member to get it, is the first to have ordered it.

Requirement R11 "*A book can be taken only if it is not borrowed*"

macro $R_{11}\,(B, M) =$
(
[**true*** . ({LEND !M !B} **or** {TAKE !M !B}) . (**not** ({RETURN !B}))* .
({LEND !M !B} **or** {TAKE !M !B}) . (**not** ({RETURN !B}))* . {RETURN !B}] **false**
)
end_macro
R_{11} ("B1", "M1")

This property corresponds to the pattern "α_1 does not occur between α_2 and α_3", which is expressed by the following scheme, easily recognizable in this formula:

$$[\,\textbf{true}^*\,.\,\alpha_2\,.\,(\textbf{not}\ \alpha_3)^*\,.\,\alpha_1\,.\,(\textbf{not}\ \alpha_3)^*\,.\,\alpha_3\,]\ \textbf{false}$$

Requirement R13 "*A member can relinquish library membership only when all his loans have been returned and all his reservations have either been used or cancelled*"

macro $R_{13}\,(B, M) =$
(
[**true*** .
({LEND !M !B} **or** {TAKE !M !B}) .
(**not** ({RETURN !B} **or** {TRANSFER !"M2" !B} **or** {TRANSFER !"M3" !B}))* .
{LEAVE !M} . (**not** ({RETURN !B} **or** {TRANSFER !"M2" !B} **or** {TRANSFER !"M3" !B}))* .
({RETURN !B} **or** {TRANSFER !"M2" !B} **or** {TRANSFER !"M3" !B})] **false**
and
[**true*** . {RESERVE !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
R_{13} ("B1", "M1")

Requirement R14 "*Ultimately, there is always a procedure that enables a member to leave the library*"

macro $R_{14}\,(M) =$
  (
    $[\,\mathbf{true}^*\,.\,\{\texttt{JOIN !}M\}\,.\,(\mathbf{not}\ \{\texttt{LEAVE !}M\})^*\,]\ \langle\,(\mathbf{not}\ \{\texttt{LEAVE !}M\})^*\,.\,\{\texttt{LEAVE !}M\}\,\rangle\ \mathbf{true}$
)
end_macro
$R_{14}\,(\texttt{"M1"})$

Requirement R15 "*A member cannot borrow more than the loan limit defined at the system level for all users*"

macro $R_{15}\,(M) =$
  (
    $[\,\mathbf{true}^*\,.\,\mathbf{let}\ B_1 : \mathbf{string} := \texttt{"B1"}, B_2 : \mathbf{string} := \texttt{"B2"}\ \mathbf{in}$
      $(\{\texttt{LEND !}M\ \texttt{!}B_1\}\ \mathbf{or}\ \{\texttt{TAKE !}M\ \texttt{!}B_1\})\,.$
      $(\mathbf{not}\ (\{\texttt{TRANSFER ?}M_2 : \mathbf{string}\ \texttt{!}B_1\}\ \mathbf{or}\ \{\texttt{RETURN !}B_1\}))^*\,.$
      $(\{\texttt{LEND !}M\ \texttt{!}B_2\}\ \mathbf{or}\ \{\texttt{TAKE !}M\ \texttt{!}B_2\})\ \mathbf{end\ let}\,]\ \mathbf{false}$
)
end_macro
$R_{15}\,(\texttt{"M1"})$

This property is dependent on the maximum number *NbLoans* of books a member can have at any time in his/her possession. In the above, *NbLoans* was set to two.

Index

Bibliography

[ABJ+99] P.A. Abdulla, A. Bouajjani, B. Jonsson, M. Nilsson. Handling Global Conditions in Parameterized System Verification. In *Proceedings of Conference on Computer-Aided Verification*, LNCS vol. 1633, pages 134–145, Springer, 1999.

[Abr05] J.R. Abrial. *The B-Book - Assigning programs to meanings.* Cambridge University Press, 2005.

[AK86] K. R. Apt, D. C. Kozen. Limits for Automatic Verification of Finite-State Concurrent Systems. In *Journal of Information Processing Letters*, ACM vol. 22, pages 307–309, ACM, 1986.

[Bro00] N. Brownlee. Failures-divergence refinement. Formal Systems (Europe) Ltd. In Blount MetraTech Corp. Accounting Attributes and Record Formats, 2000.

[BCC+99] A. Biere, A. Cimatti, E. Clarke, Y. Zhu. Symbolic Model Checking without BDDs. In *Proceedings of Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS vol. 1579, pages 193–207, Springer, 1999.

[BPS01] J.A. Bergstra, A. Ponse, S.A. Smolka. *Handbook of Process Algebra*, Elsevier, 2001.

[BK84] J.A. Bergstra, J.W. Klop. Process Algebra for synchronous communication. In *Journal of Information Control*, vol. 60, pages 109–137, Elsevier, 1984.

[BK85] J. A. Bergstra, J. W. Klop. Algebra of Communicating Processes with Abstraction. In *Journal of Theoretical Computer Science*, vol. 37, pages 77–121, Elsevier, 1985.

[BMC+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang. Symbolic Model Checking: 10^{20} States and beyond. In *Journal of Information and Computation*, pages 142–170, Elsevier, 1992.

[Cl] ClearSy. *Atelier B.* http://www.atelierb.societe.com.

[Cho10] R. Chossart. Évaluation d'outils de vérification pour les spécifications de systèmes d'information. Master's thesis, Université de Sherbrooke, 2010.

[CC77] P. Cousot, R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of Symposium on Principles of Programming Languages*, pages 238-252, ACM, 1977.

[CCG+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. Springer, 2002.

[CCG+11] 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.

[CEF+86] E. M. Clarke, E. A. Emerson, A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In *Journal of Transactions on Programming Languages and Systems*, vol. 8, pages 244–263, ACM, 1986.

[CEF+96] E. M. Clarke, R. Enders, T. Filkorn, S. Jhay. Exploiting Symmetry in Temporal Logic Model Checking. In *Journal of Formal Methods in System Design*, vol. 9, pages 77–104, Springer, 1996.

[CGB86] E. M. Clarke, O. Grumberg, M. C. Browne. Reasoning about Networks with many identical Finite-State Processes. In *Proceedings of Symposium on Principles of Distributed Computing*, pages 240–248, ACM, 1986.

[CGL92] E. M. Clarke, O. Grumberg, D. E. Long. Model Checking and Abstraction. In *Proceedings of Symposium on Principles of Programming Languages*, vol. 16, pages 343-354, ACM, 1992.

[DGG97] D. Dams, R. Gerth, O. Grumberg. Abstract Interpretation of Reactive Systems. In *Journal of Transactions on Programming Languages and Systems*, vol. 19, pages 253–291, ACM, 1997.

[DV90] R. De Nicola, F. Vaandrager. Action versus state based logics for transition systems. In *Proceedings of the LITP spring school on Theoretical Computer Science on Semantics of systems of concurrent processes*, pages 407–419, Springer, 1990.

[Eme86] E. Allen Emerson, C-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of Symposium on Logic in Computer Science*, pages 267–278, 1986.

[EH82] E. A. Emerson, J. Y. Halpern. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. In *Proceedings of Symposium on Theory of Computing*, pages 169–180, ACM, 1982.

[EK00] E. A. Emerson, V. Kahlon. Reducing Model Checking of the Many to the Few. In *Proceedings of Conference on Automated Deduction*, LNCS vol. 1831, pages 236–254, Springer, 2000.

[EL+86] E. A. Emerson and C-L. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus. In *Proceedings of Symposium on Logic in Computer Science*, pages 267–278, 1986.

[EN95] E. A. Emerson, K. S. Namjoshi. Reasoning about Rings. In *Proceedings of Symposium on Principles of Programming Languages*, pages 85–94, ACM, 1995.

[ESW96] E. A. Emerson, A. P. Sistla, H. Weyl. Symmetry and Model Checking. In *Journal of Formal Methods in System Design*, vol. 9, pages 105–131, Springer, 1996.

[ETL+04] N. Evans, H. Treharne, R. Laleau, M. Frappier. How to verify dynamic properties of information systems. In *Proceedings of Workshop of Software Engineering and Formal Methods*, pages 416–425, 2004.

[FFC+10] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, M. Ouenzar. Comparison of model checking tools for information systems. In *Proceedings of International Conference on Formal Engineering Methods*, LNCS vol. 6447, pages 581–596, Springer, 2010.

[FM11] M. Frappier, A. Mammar. Proving Non-interference on Reachability Properties: A Refinement Approach. In *Proceedings of Software Engineering Conference*, pages 25–32, 2011.

[FSt03] M. Frappier, R. St.-Denis. EB^3: an entity-based black-box specification method for information systems. In *Journal of Software and System Modeling*, LNCS vol. 2, pages 134–149, Springer, 2003.

[Ger06] F. Gervais. *Combinaison de spécifications formelles pour la modélisation des systèmes d'information.* PhD thesis, Université de Sherbrooke, 2006.

[Gla87] R. J. van Glabbeek, F. W. Vaandrager. Petri Net Models for Algebraic Theories of Concurrency, in: In *Proceedings of Conference on Parallel Architectures and Languages Europe*, LNCS vol. 259, pages 224-242, Springer-Verlag, 1987.

[GFL05] F. Gervais, M. Frappier, R. Laleau. Synthesizing B Specifications from EB^3 Attribute Definitions. In *Proceedings of Conference on Integrated Formal Methods*, LNCS vol. 3771, pages 207–226 Springer, 2005.

[GFL06] F. Gervais, M. Frappier, R. Laleau. Refinement of EB^3 Process Patterns into B Specifications. In *Proceedings of Conference on Formal Specification and Development in B*, LNCS vol. 4355, pages 201–215, Springer, 2006.

[GLM+11] H. Garavel, F. Lang, R. Mateescu, W. Serwe. CADP 2011: A toolbox for the construction and analysis of distributed processes. In *International Journal on Software Tools for Technology Transfer*, LNCS, vol. 15, pages 89–107, Springer, 2011.

[GM84] U. Goltz, A. Mycroft. On the Relationship of CCS and Petri Nets. In *Proceedings of Colloquium on Automata, Languages and Programming*, LNCS, vol. 172, pages 196–208, Springer-Verlag, 1984.

[GSS+09] H. Garavel, G. Salaün, W. Serwe. On the semantics of communicating hardware processes and their translation into LOTOS for the verification of asynchronous circuits with CADP. In *Journal of Science of Computer Programming*, vol. 74, pages 100–127, 2009.

[GP93] P. Godefroid, D. Pirottin. Refining dependencies improves Partial Order Verification Methods. In *Proceedings of Conference on Computer-Aided Verification*, LNCS, vol. 697, pages 438–449, Springer, 1993.

[Hoa78] C. A. R. Hoare. Communicating Sequential Processes. In *Communication of the ACM*, vol. 21, pages 666–677, ACM, 1978.

[Hol04] G. J. Holzmann. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley 2004.

[HBR09] Y. Hanna, S. Basu, H. Rajan. Behavioral Automata Composition for Automatic Topology Independent Verification of Parameterized Systems. In *Proceedings of Symposium on the Foundations of Software Engineering*, pages 325–334, ACM, 2009.

[HSB+10] Y. Hanna, D. Samuelson, S. Basu, H. Rajan. Automating Cut-off for Multi-parameterized Systems. In *Proceedings of Conference on Formal Engineering Methods*, 2010, vol. 6447, pages 338–354, Springer, 2010.

[ID96] C. N. Ip, D. L. Dill. Better Verification through Symmetry. In *Journal of Formal Methods in System Design*, vol. 9, pages 97–111, ACM, 1996.

[Jac06] D. Jackson. Software Abstractions. MIT Press, 2006.

[JFG+10] M. E. Jiague, M. Frappier, F. Gervais, P. Konopacki, R. Laleau, J. Milhau, R. St-Denis. Model-Driven Engineering of Functional Security Policies. In *Proceedings of Conference on Enterprise Information Systems*, vol. 12, pages 374–379, 2010.

[Koz83] D. Kozen. Results on the Propositional -calculus. In *Journal of Theoretical Computer Science*, vol. 27, pages 333–354, 1983.

[KP88] S. Katz, D. Peled. An efficient Verification Method for Parallel and Distributed Programs. In *Proceedings of Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS vol. 354, pages 489–507, Springer, 1988.

[Laz99] R.S. Lazic. A Semantic Study of Data Independence with Applications to Model Checking PhD thesis, Oxford University Computing Laboratory, 1999.

[Lot01] ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard number 15437:2001, International Organization for Standardization – Information Technology, Genève, 2001.

[LB03] M. Leuschel, M. Butler. ProB: A model checker for B. In *Proceedings of Symposium on Formal Methods*, LNCS vol. 2805, pages 855–874, Springer-Verlag, 2003.

[LGS+95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, S. Bensalem, D. Probst. Property Preserving Abstractions for the Verification of Concurrent Systems. In *Proceedings of Conference on Formal Methods in System Design*, vol. 6, pages 11–44, Springer, 1995.

[LMC00] M. Leuschel, T. Massart, A. Currie. How to make FDR spin: LTL model checking of CSP by refinement. Technical report, 2000.

[LSH+09] F. Lang, G. Salaün, R. Hérilier, J. Kramer, J. Magee. Translating FSP into LOTOS and networks of automata. In *Journal of Formal Aspects of Computing*, vol. 22 pages 681–711, Springer, 2009.

[Mil80] R. Milner. A Calculus of Communicating Systems. In *Proceedings of Computer Science*, LNCS vol. 92, Springer, 1980.

[Mor98] C .C. Morgan. Programming from Specifications, Prentice Hall, 1998.

[MIL+11] J. Milhau, A. Idani, R. Laleau, M.A. Labiadh, Y. Ledru, M. Frappier. Combining UML, ASTD and B for the formal specification of an access control filter. In *Journal of Innovations in Systems and Software Engineering*, vol. 7, pages 303–313, Springer, 2011.

[MK+06] J. Magee, J. Kramer. Concurrency: State models and Java programs ACM, 2006.

[MPW+92] R. Milner, J. Parrow, D. Walker. A calculus of mobile processes. In *Journal of Information and Computation*, vol. 100, pages 1–40, ACM, 1992.

[MR05] M .R. Mousavi, M .A. Reniers. Congruence for Structural Congruences. In *Proceedings of Conference on Foundations of Software Science and Computational Structures*, vol. 3441, pages 47–62, Springer, 2005.

[MS10] R. Mateescu, G. Salaün. Translating Pi-calculus into LOTOS NT. In *Proceedings of Conference on Integrated Formal Methods*, LNCS vol. 6396, pages 229–244, Springer, 2010.

[MT08] R. Mateescu, D. Thivolle. A model checking language for concurrent value-passing systems. In *Proceedings of Symposium on Formal Methods*, LNCS vol. 5014, pages 148–164, Springer, 2008.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of Conference on Theoretical Computer Science*, LNCS vol. 104, pages 167–183, Springer-Verlag, 1981.

[Pet75] C. A. Petri. Concurrency. In *Advanced Course: Net Theory and Applications*, 1975.

[QS83] J-P. Queille and J. Sifakis. Fairness and Related Properties in Transition Systems-A Temporal Logic to Deal with Fairness. In *Journal of Acta Informatica*, vol. 19, pages 195–220, 1983.

[Ros98] B. A. W. Roscoe. The Theory and Practice of Concurrency. Prentice Hall PTR, 1998.

[Str82] R. Streett. Propositional Dynamic Logic of Looping and Converse. In *Journal of Information and Control*, vol. 54, pages 121–141, 1982.

[ST02] S. Schneider, H. Treharne. CSP Theorems for Communicating B Machines. Technical Report CSD-TR-02-12, Dept. of Computer Science, Royal Holloway, University of London, 2002.

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. In *Pacific Journal of Mathematics 5*, vol. 2, pages 285–309, 1955.

[Ten91] R. Tennent. Semantics of Programming Languages. Prentice Hall, 1991.

[VD13] D. Vekris, C. Dima. Efficient Operational Semantics for EB^3 for Verification of Temporal Properties. In *Proceedings of Conference on Fundamentals of Software Engineering*, LNCS vol. 8161, pages 133–149, Springer, 2013.

[VLD+13] D. Vekris, F. Lang, C. Dima, R. Mateescu. Verification of EB^3 Specifications using CADP. In *Proceedings of Conference on Integrated Formal Methods*, LNCS vol. 7940, pages 61–76, Springer, 2013.

[Win84] G. Winskel. A new definition of Morphism on Petri Nets. In *Proceedings of Symposium on Theoretical Aspects of Computer Science*, LNCS vol. 166, pages 140–150, Springer-Verlag, 1984.

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 entities in a system, and attribute functions that are recursive functions evaluated on the system execution trace describing entity attributes. The verification of EB^3 specifications against temporal properties is of great interest to users of EB^3. In this thesis, we focus on liveness properties of information systems, which express the eventuality that certain actions take place. The verification of liveness properties can be achieved with model checking.

First, we present an operational semantics for EB^3 programs, in which attribute functions are computed during program evolution and their values are stored into program memory. This semantics permits us to define an automatic translation from EB^3 to LNT, a value-passing concurrent language with classical process algebra features. Our translation ensures the one-to-one correspondence between states and transitions of the labelled transition systems corresponding to the EB^3 and LNT specifications. Then, we automate this translation with the EB^{3}2LNT tool, thus equipping the EB^3 method with the functional verification features available in the model checking toolbox CADP.

With the aim of improving the model checking results of this approach, we explore abstraction techniques for information systems specified in EB^3. In particular, we concentrate on a specific family of systems called parametric, whose behaviour is scaled in keeping with the predefined value of a system parameter. Finally, we apply this method on the EB^3 context.
Keywords: Information Systems, Process Algebras, Model Checking, Abstraction Techniques

Resumé: EB^3 est un langage de spécification développé pour la spécification des systèmes d'information. Le noyau du language EB^3 comprend des spécifications d'algèbre de processus afin de décrire le comportement des entités du système et des fonctions d'attributs qui sont des fonctions récursives dont l'évaluation se fait sur la trace d'exécution du système décrivant les attributs des entités. La vérification de propriétés temporelles en EB^3 est un sujet de grande importance pour des utilisateurs de EB^3. Dans cette thèse, on se focalise sur les propriétés de vivacité concernant des systèmes d'information exprimant l'éventualité que certaines actions puissent s'exécuter. La vérification des propriétés de vivacité se fait à l'aide de model checking.

Dans un premier temps, on presente une sémantique opérationnelle de EB^3, selon laquelle les fonctions d'attributs sont évaluées pendant l'exécution du programme puis stockées. Cette sémantique nous permet de définir une traduction automatique de EB^3 vers LNT, qui est une algèbre de processus. Notre traduction assure la correspondance un à un entre les états et les transitions des systèmes de transition étiquetés correspondent respectivement à des spécifications EB^3 et LNT. Ensuite, on automatise la traduction grâce à l'outil EB^{3}2LNT fournissant aux utilisateurs de EB^3 tous les outils de vérification fonctionnelle disponible dans CADP.

Dans le but d'améliorer les résultats de notre approche concernant le model checking, on explore des techniques d'abstraction dédiées aux systèmes d'information spécifiées en EB^3. En particulier, on se focalise sur une famille spécifique de systèmes appelés paramétriques dont le comportement varie en fonction de la valeur prédéfinie d'un paramètre du système. Enfin, on applique cette méthode dans le contexte de EB^3.
Mots clès: Systèmes d'Information, Algèbre de Processus, Model Checking, Techniques d'Abstraction