N° d'ordre : 3808

THÈSE

présentée à

L'UNIVERSITÉ BORDEAUX 1

ÉCOLE DOCTORALE DE
MATHÉMATIQUES ET INFORMATIQUE

par

Thế Quang Trần

pour obtenir

LE GRADE DE DOCTEUR

Spécialité: INFORMATIQUE

Unfolding Based Verification of Concurrent Infinite-State Systems

---

Soutenue le 19 juin, 2009

Après avis des rapporteurs :

MM.	Serge Haddad	PR. ENS de Cachan
	Claude Jard	PR. ENS de Cachan/Bretagne

Devant le jury composé de :

MM.	Bernard Berthomieu	CR. CNRS/LAAS
	Bruno Courcelle	PR. Univ. Bordeaux 1
	Jean-Michel Couvreur	PR. Univ. d'Orléans
	Serge Haddad	PR. ENS de Cachan
	Frederic Herbreteau	MC. ENSEIRB
	Claude Jard	PR. ENS de Cachan/Bretagne
	Grégoire Sutre	CR. CNRS/LaBRI
	Igor Walukiewicz	DR. CNRS/LaBRI

Directeur de thèse	Igor Walukiewicz
Co-encadrants	Frédéric Herbreteau et Grégoire Sutre
Laboratoire	LaBRI

Vérification des systèmes concurrents infinis par technique de dépliages

Résumé: Nous proposons une technique de dépliage pour vérifier les systèmes concurrents infinis bien structurés. Certaines propriétés d'intérêt comme la bornitude, la couverture et la terminaison sont décidables grâce à la bonne structure de ces systèmes. D'autre part, le dépliage réduit efficacement l'explosion combinatoire en exploitant l'ordre partiel entre les événements des systèmes concurrents. Nous proposons une modélisation par structure d'événements pour des systèmes bien structurés élémentaires, tels les compteurs et les files de communication. Le dépliage d'un réseau de structures d'événements, étant une structure d'événements, nous proposons ensuite une approche hiérarchique à la modélisation et à la vérification des systèmes, qui préserve la bonne structure. Enfin, nous proposons une technique d'élimination des événements redondants. La mise en œuvre de notre approche dans l'outil ESU nous permet de conclure à son efficacité.

Mots clés: algorithme de dépliage, ordre partiel, préordre, système infini, produit synchronizé, structure d'événements, prefixe fini, bornitude, terminaison, quasi-vivacité.

Acknowledgments

My foremost gratitude goes to my director Igor Walukiewicz for his guidance. Despite his crowded schedule, he always found some time to help me to overcome problems. His help was really significant to write this thesis. I wish to also thank my two first directors André Arnold and Jean-Michel Couvreur although we had not much time to work together.

I am very grateful to my co-directors Frédéric Herbreteau and Grégoire Sutre who initiated me to the research, and then have conducted me throughout this thesis. As their first PhD student, we have shared an excellent experience. They taught me a lot about not only how computers aid verification but also how competent and cheerful supervisors aid a student like me. It is not an overstatement to say that without their patience and guidance this thesis would not exist.

Many thanks to Bernard Berthomieu, Bruno Courcelle, Jean-Michel Couvreur, Serge Haddad, and Claude Jard for having kindly accepted to be members of the jury and review the thesis. This important task is always time consuming and deserve all my gratitude. I particularly thank Bruno Courcelle for having been my supervisor for six months.

Thanks to Anne Dicky, Alain Griffaut, and Olivier Ly for their friendship. I also would like to thank all the members of the *Formal Methods group* of *LaBRI*.

A special thanks has to be addressed to Michel Mouyssinat. He has looked after me and always been there when I needed help. I wish to thank Antoine Blascos, another french friend of mine that I appreciate very much his friendship.

Finally, I would like to switch to my mother tongue in the following page to thank my family and my vietnamese friends.

Biết ơn

Tôi muốn dành đôi dòng ngắn ngủi viết bằng tiếng mẹ đẻ cho những người mà chính họ đã tạo nên tôi với những kết quả đạt được trong nghiên cứu này.

Trước hết là với bố mẹ, những người đến giờ mới yên lòng vì đã lo xong cho con ăn học tới nơi tới chốn. Con trai của bố mẹ chỉ biết làm nhẹ bớt lo toan đó bằng những kết quả đẹp - cái mà bố mẹ luôn tự hào suốt quá trình học dài đằng đẵng của con. Cái đạt được ngày hôm nay, một lần nữa, chính là quả chín đền đáp công ơn nuôi nấng con ăn học.

Tiếp đến là với những người thân trong gia đình: anh Kỳ, chị Lê, anh Hà, chị Thảo, Quân, Minh, và cả Thỏ con nữa. Mọi người đã gánh giúp em, giúp cậu, những việc mà một người con trai lớn khi xa nhà không thể làm được. Hơn nữa, niềm tin của anh chị và các cháu với em đã không cho phép em buông xuôi, và luôn là động lực để em có đi có đến.

Mảnh đất Bordeaux cho rượu và con người làng Nho cho tình. Đếm năm tháng cứ ngỡ là dài nhưng nhắm mắt nhớ lại thì thấy quá ngắn. Từ anh Khuê già cho nhà đón ngày đầu, đến Đai fou cho nhà tù ngày cuối, biết bao người ở làng này, nếu không muốn nói là tất cả, đã giúp tôi ăn-ở-vui-chơi, nói chung là sống, để mà học tốt. Nợ chẳng trả đủ đành cười xoà nói lời cám ơn. Tôi cũng xin phép không kể hết tên chủ nợ vì quá dài.

Nếu coi bạn bè như chân tay thì mấy năm xa nhà cũng đủ làm tôi khác người (hy vọng không giống ngợm). Hơn nữa, tính rẻ hai năm có thêm một đầu đã là quá hời: anh Hoàng cong, Đức chích, Đại fou. Những cái đầu sẵn sàng chung vai mà không bao giờ làm đau đầu tôi, quả là đáng quý. Đáng quý hơn nữa khi biết rằng làm nghiên cứu là đã phải đau đầu.

Có những người đã gửi trái tim cho tôi lúc đang còn là nghiên cứu sinh, và tôi đã dùng phí họ trả để hoàn thành nghiên cứu này. Tôi chỉ xin nêu tên người duy nhất muốn được nêu tên, và cũng là người duy nhất, đến lúc viết những lời này, tôi vẫn giữ: Thuỷ.

Là giai chưa vợ, lời cuối cùng lại là vu vơ nhất, đó là dành cho vợ [tương lai của] tôi - sức ép vô hình giục tôi hoàn thành sớm nghiên cứu này.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

Below are the notations used in this thesis for important entities and constructions, together with the number of page in which each notation is defined or first appears.

$\emptyset$	empty set, *p. 20*
$\times$	Cartesian product, *p. 11*
$\lvert X\rvert$	cardinality of a set X, *p. 11*
$\lvert w\rvert$	length of a word w, *p. 12*
$\downarrow_i$	component restriction, Notation 2.4.11, *p. 16*
$\{x \,/\, \Phi\}$	set of x such that Φ, *p. 11*
$\leq$	causality relation, Definition 3.1.1, *p. 21*
$\geq(X)$	downward closure of X w.r.t. $\leq$, *p. 13*
$\leq(X)$	upward closure of X w.r.t. $\leq$, *p. 13*
$\lessdot$	minimal relation of which the transitive and reflexive closure is $\leq$, Notation 2.3.6, *p. 14*
$\lessdot(e)$	direct successors of e, *p. 22*
$\gtrdot(e)$	direct predecessors of e, *p. 22*
$\#$	conflict relation, Definition 3.1.1, *p. 21*
$\parallel$	concurrent relation, Notation 3.1.2, *p. 22*
$\vdash$	extension relation, Definition 3.1.6, *p. 23*
$C \vdash e$	event e is an extension of configuration C, *p. 23*
$C \Vdash X$	X is an extension set of configuration C, *p. 23*
$\approx$	isomorphic relation, Definition 3.2.7, *p. 29*
$\rightarrow$	transition relation, Definition 2.4.1, *p. 14*
$s \xrightarrow{a} s'$	s' is reachable from s by action a, *p. 14*
$\preccurlyeq$	well-preorder, Definition 2.3.1, *p. 12*
$\preccurlyeq_\otimes$	product (well-)preorder, Definition 4.1.7, *p. 64*
$\preccurlyeq^{\mathcal{C}}$	(well-)preorder on the configuration set, *p. 68*
$\preccurlyeq^{\mathcal{M}}$	marking preorder, Definition 4.2.2, *p. 69*
$\trianglelefteq$	adequate order, *p. 77*
$\trianglelefteq_l$	adequate order based on lexicography, Definition 4.2.17, *p. 78*
A^*	finite words over an alphabet A, *p. 12*
A^ω	infinite words over an alphabet A, *p. 12*
$\mathcal{B}$	bijection between two event sets, Definition 3.1.13, *p. 25*
b-$\mathcal{BC}^v$	v-initialized bounded counter, Definition 3.3.12, *p. 37*
b-$\mathcal{BP}$	b-bounded process, Definition 3.3.13, *p. 38*

$\mathcal{C}_{\mathcal{E}}$	configurations of $\mathcal{E}$, Notation 3.1.5, *p. 23*
$\mathcal{C}^{l}_{\mathcal{E}}$	local configurations of $\mathcal{E}$, *p. 23*
$\mathsf{Codom}(\mathcal{F})$	codomain of a function $\mathcal{F}$, *p. 11*
$\mathcal{CT}$, v-$\mathcal{CT}$	counter, v-initialized counter, Definition 3.3.6, *p. 34*
$\mathcal{CP}$	causality process, *p. 34*
k-$\mathcal{CP}$	k-causality process, Definition 3.3.9, *p. 35*
(k,v)-$\mathcal{CP}$	(k,v)-causality process, Definition 3.3.19, *p. 40*
M-$\mathcal{CP}$	M-causality process, Definition 3.3.27, *p. 46*
(M,v)-$\mathcal{CP}$	(M,v)-causality process, Definition 3.3.31, *p. 49*
$\mathcal{D}$	the depth function, Definition 3.3.34, *p. 51*
$\mathsf{Dom}(\mathcal{F})$	domain of a function $\mathcal{F}$, *p. 11*
E	events, Definition 3.1.1, *p. 21*
ε	empty word, *p. 12*
$\mathcal{E}$	labeled event structure, Definition 3.2.1, *p. 27*
$\mathcal{E}\|_F$	restriction of (labeled) event structure $\mathcal{E}$ onto event set F, Definition 3.1.10, *p. 24*
$\widehat{\mathcal{E}}$	prefix under construction, *p. 88*
$\mathcal{FF}$	FIFO channel, *p. 42*
(M,v)-$\mathcal{FF}$	v-initialized FIFO channel over M, Definition 3.3.22, *p. 42*
$\mathcal{FL}$	lossy FIFO channel, *p. 62*
$\mathcal{I}_X$	identity relation over X, *p. 12*
$\mathcal{L}$	label function, Definition 3.2.1, *p. 27*
$\mathcal{LET}$	labeled event tree, Definition 3.3.3, *p. 32*
$\mathcal{LTS}$	labeled transition system, Definition 2.4.1, *p. 14*
$\mathcal{LTS}^{\mathcal{E}}$	labeled transition system induced by $\mathcal{E}$, Definition 3.2.4, *p. 28*
$\mathcal{L}^{\mathcal{W}}$	function on words that is based on a label function $\mathcal{L}$, *p. 12*
M	FIFO channel's messages, *p. 42*
$!M$	sending actions, Notation 3.3.23, *p. 42*
$?M$	receiving actions, Notation 3.3.23, *p. 42*
$\mathsf{Max}_{\leq}(X)$	maximal elements of X w.r.t. $\leq$, *p. 13*
$\mathcal{M}$	marking function, Definition 3.2.1, *p. 27*
$\mathsf{Min}_{\leq}(X)$	minimal elements of X w.r.t. $\leq$, *p. 13*
$NR_{(\mathcal{E},\lhd)}$	the prefix without $\lhd$-redundant event of $\mathcal{E}$, *p. 143*
$\mathsf{post}^{*}_{\mathcal{LTS}}$	reachability set of $\mathcal{LTS}$, *p. 16*
$\mathcal{P}(X)$	power set of a set X, *p. 11*
pb	finite pred-basis, Definition 4.1.13, *p. 67*
PE	possible extensions, *p. 88*
$\Pi_{!M}$, $\Pi_{?M}$	M-letter morphisms, Definition 3.3.24, *p. 43*
$\not{\mathcal{R}}$ or $\overline{\mathcal{R}}$	complement of a relation $\mathcal{R}$, *p. 11*
$\mathcal{R}\|_X$	restriction of a relation $\mathcal{R}$ to X, Notation 2.1.1, *p. 11*
$\mathcal{R}^{*}$	reflexive and transitive closure of a relation $\mathcal{R}$, *p. 12*
$\mathcal{R}^{+}$	transitive closure of a relation $\mathcal{R}$, *p. 12*
$\mathcal{R}^{-1}$	converse relation of a relation $\mathcal{R}$, *p. 11*

S	set of states, Definition 2.4.1, *p. 14*
s^0	initial state, Definition 2.4.1, *p. 14*
$\mathcal{SP}$	a synchronized product of labeled transition systems, Definition 2.4.12, *p. 17*
Σ	set of actions, Definition 2.4.1, *p. 14*
Σ^τ	internal actions, *p. 63*
$\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$	the truncation of $\mathcal{E}$ w.r.t. the cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$, Definition 4.2.12, *p. 74*
$\mathcal{V}$	function representing synchronization in synchronized products of (labeled) event structures, Definition 3.3.39, *p. 54*
$(X, \leq)$	partially ordered set, Definition 2.3.2, *p. 13*
$\otimes(X_1, \ldots, X_n)$	n-dimension space, Notation 2.4.10, *p. 16*
$(\widehat{E}, \widehat{\leq}, \widehat{\#}, \widehat{\mathcal{L}}, \widehat{\mathcal{M}})$	structure variables, *p. 88*

Chapter 1

Introduction

Contents

Because of the success of embedded systems in automobiles, airplanes and other safety critical systems in our everyday life, we are likely to become more dependent on the proper functioning of computing devices. Bugs and errors may lead to dramatic consequences. Even when failure is not life-threatening, the consequences of having to replace critical code or circuitry can be a substantial economic loss. This fact emphasizes the necessity of confidence in such systems.

At the same time, the advances in computer science and especially in hardware led to an increase of systems' complexity, and consequently, makes it hard to design these systems without defects. This situation is more alarming since *concurrent systems* are customarily used. In fact, a concurrent system is composed of several components that run in parallel, possibly on different locations, and communicate with each other. Each component can be viewed as a *reactive system* that continuously interacts with its environment which may be another component. Hence, the effect of even very minor programming mistakes in a certain component can cause major system failures. Testing is also of limited help in concurrent system's design since it usually involves providing certain inputs and observing the corresponding outputs. Therefore, checking all of the potential behaviors resulting from all interactions between the different concurrent components of the system using testing techniques is rarely possible. Many errors can easily go through the testing phase undetected and show up only after a long period of operation. Moreover, even if some bug is found during a particular testing run, it may not occur during the next runs, and locating concurrency related bugs is a difficult task.

Formal verification has been proposed as a way to obtain guarantees on the correctness of safety critical systems. Verification means that a system description conforms to its expected properties. Therefore, all possible behaviors of the system have to be checked to determine if all of them are compatible with the given property. In order to be able to perform such a verification, one needs a formal modeling language in which

the system can be described, a formal specification language for the formulation of properties, and a deductive calculus or algorithm for the verification process. Hence, there are roughly two approaches to formal verification: *logical inference* and *model checking*. The first one consists of using a formal version of mathematical reasoning about the system, usually using theorem provers. Methods in this approach are more general but harder to use because they are usually only partially automated. Although there has been considerable research on the use of theorem provers, these methods are time consuming and often require a great deal of manual intervention. Only an experienced user with certain understanding of the system can perform a nontrivial proof, for instance, in which he has to find loop invariants or inductive hypotheses.

On the contrary, much of the success of *model checking*, firstly developed in the early 80's [CE81, QS82, EC82], is due to the fact that it performs a *fully automatic verification*. With model checking, all the user has to provide is a model of the system and a formulation of the property to be checked. The verification tool will either terminate with an answer indicating that the model satisfies the formula or show why the formula fails to hold in the model. These counter-examples are particularly helpful in locating errors in the model. If the model does not satisfy a given correctness specification, this is often connected to a mistake in the real system. Nevertheless, as an over-approximation of the real system, the model is sometimes too coarse and does not satisfy some correct properties, although the real system does. In such a case, the model of the system must be refined to get closer to the real system. Aiming at automatic verification methods, let us focus on model checking.

1.1 Model checking

Model checking is a verification technique that applies to a large class of systems and consists of three steps.

Modeling systems by mathematical models. A mathematical model of the semantics of a system or of a program is a mathematical structure, in general, an algebra, consisting of sets, functions, graphs, and possibly logical predicates. Such a model is an idealized abstraction that needs syntax to represent it directly. Examples of models often used are: (timed, hybrid) automata, Kripke structures, finite state machines, (labeled) transition systems, Petri nets, process algebra, (labeled) event structures.

Many powerful models have been introduced in order to incorporate some specific aspects, i.e. data view (in heterogeneous systems), state view (in reactive systems), or supporting the concept of hierarchical decomposition (in complex concurrent systems). Much effort of the theory of concurrency has been devoted to the study of suitable models for concurrent reactive systems, and to the formal understanding of their semantics. As each system has an implementation in terms of a state machine, it always has a state space, and so does the corresponding model. Such models have a common idea that they are based on atomic units of change - transition, actions, events - which are indivisible and allow the system to change its state.

The difference between the models for concurrent systems may be expressed w.r.t. three relevant parameters: behavior or system model, interleaving or noninterleaving model, and linear or branching time model [SNW96]. In other words, models for concurrency can be classified into the eight classes of models obtained by varying these three parameters in all the possible ways.

Representing the property in a specification language. Among specification languages, the first and probably the most successful one is first-order logic. Almost all interesting properties of programs can be formulated in this language. However, this classical logic is not well-suited for specifying properties of concurrent computations. Temporal logic, which can assert how the behavior of the system evolves over time, has proved to be suitable for this purpose. Because temporal logics, such as LTL, CTL, CTL* [Pnu77, CE81] can describe the ordering of events without introducing time explicitly.

In general, properties of interest go under the category of ordered executions. It relates to verification of event and state ordering. Properties such as *safety* and *liveness* belong to this category. When a system has to be verified, it often turns out that the property we are interested in is simply expressible in terms of reachability. This thesis is dedicated to reachability based problems.

Model checking algorithm. Once the model is built, one formally states the property to be checked by a logical formula, and uses an appropriate algorithm to verify if the formula holds in the model. *State space exploration* is one of the most successful approaches particularly when reachability-based properties have to be checked. It consists in exploring a global state graph representing all behaviors of the model/system. This is done by recursively computing all successor states of all states encountered during the exploration, starting from a given initial state, by executing all possible actions/transitions in each state. If the state space is finite, it can be explored entirely.

The first model checkers worked by constructing the whole state space prior to property checking, but modern tools are able to perform verification on-the-fly as the states are computed.

Verification by state space exploration has been studied by many researchers. The simplicity of the strategy lends itself to easy, and thus efficient, implementations. Moreover, the range of properties that state space exploration techniques can verify has been substantially broadened thanks to the development of model checking methods for various temporal logics [CES86, QS82, VW86]. As many verification tools have been developed, for example Caesar, Spin (see [FGM+92, Hol97, BBF+01]), the effectiveness of model checking, and particularly the state space exploration techniques, for debugging and proving correct systems is increasingly becoming established. The number of success stories about applying these techniques to industrial-size systems keeps growing.

However, model-checking suffers from two main drawbacks. Firstly, even a relatively small system model can (and often does) yield a very large state space. More precisely, owing to simple combinatorics, the size of the state space can be exponential in the size of the model being analyzed. This exponential growth is known as the *state explosion problem*. Secondly, when the system under study has an *infinite state space*, such as heterogeneous systems whose states consist of unbounded values (integers, channels in communication protocols), classical model checking no longer applies. Moreover, it does not allow to check for some essential properties such as the boundedness of the communication channels of a protocol (this is an important property when implementation is an objective).

1.2 Approaches to the state-space explosion: the unfolding technique

One category of techniques tackling state-space explosion are *symbolic methods* [BCM+92] that attempt to represent and manipulate sets of states implicitly with a help of specific data structures, rather than explicitly as enumerations of their components. The success of these methods is primarily due to the use of *binary decision diagrams (BDD)* [Bry86], for representing sets and relations over Boolean variables symbolically, making it possible to verify systems with a very large number of states (more than 10^{100} reachable states). Because of this and other technical advances in symbolic model checkers, it is now possible to verify some reactive systems of realistic industrial complexity.

Although symbolic representations have greatly increased the size of the systems that can be verified, many realistic systems are still too large to be handled. Thus, it is important to find techniques that can be used in conjunction with the symbolic methods to extend the size of the systems that can be verified. Such techniques are, for instance, *compositional reasoning* [CLM89, SG90], *abstraction* [CGL94, GS97], and *symmetry reduction* [CJEF96, ES96].

A collection of verification techniques attacking directly the sources of state-space explosion phenomenon on concurrent reactive systems have demonstrated that exploring all *interleavings* of concurrent events is not a priori necessary for verification. Indeed, interleavings corresponding to the same concurrent execution contain related information, e.g. the same reachable state. The intuition behind these technique is exploiting the independence between concurrent executed events, or in other words, the partial order over events. Hence, these techniques are called *partial-order methods*.

Many partial-order methods are based on *partial-order reductions* first appeared independently in [Val89] and [God90, GW91], and were developed further in [Val90, GHP92, WG93, Pel94]. The *stubborn sets* [Val89], the *persistent sets* [God90], and the *ample sets* [Pel94] differ in the actual details, but contain many similar ideas. Intuitively, rather than choosing to work with direct representations of partial orders, the model checking algorithms in these methods keep an interleaving representation of partial orders, but attempt to limit the expansion of each partial-order computation to just one of its interleavings, instead of all of them. A property to be checked needs to be verified only on a reduced part of the global state space. Partial-order methods yield results identical to those of verification methods based on classical interleaving. Thus they make it possible to perform the verification more efficiently.

Partial-order reductions described above are quite different from the partial-order method in our work, called *unfolding technique* [McM95a]. Unfolding technique is based on the results of the theory of *true concurrency* to replace the classical state/transition models by partially ordered graphs. More precisely, using the unfolding theory in [NPW80], the dynamics of a safe Petri net is captured by the dynamics of an acyclic net that lies in the category of *occurrence nets*. Occurrence nets as well as *event structures* - a more abstract representation - belong to so called *partial-order models* of concurrency that were discussed by many researchers in the 80's [Lam78, Maz86, Win86, Pra86]. Unfolding algorithms intuitively consist of computing some behavioral models of the system that preserve Mazurkiewicz's trace semantics [Maz86], and the properties are checked directly on these partial-order models. The verification process is generally done together with the construction of the behavior models.

Since its introduction in [McM95a], the unfolding technique has attracted considerable attention and inspired a fairly large number of works.

- The algorithm for constructing *finite prefixes* of the behavior model has been further analyzed and improved [ERV96, KK01, CGP01, ERV02, KKV03], parallelized [HKK02, SK04], and distributed [BHK06].

- The initial verification technique, that only allowed to check the reachability of a state or the existence of a deadlock [McM95a, MR97], has been extended to algorithms for (almost) arbitrary properties expressible in Linear Temporal Logic [CGP00, EH00, EH01].

- The unfolding technique, initially developed for systems modeled as safe Petri nets, has been extended to high-level Petri nets [KK03], symmetrical Petri nets [CGP01], unbounded Petri nets [AIN00], nets with read arcs [VSY98], time Petri nets [FS02, CJ06], products of transition systems [ER99], automata communicating through queues [LI05], networks of timed automata [BHR06, CCJ06], process algebras [LB99], and graph grammars [BCK04].

- The unfolding technique has been implemented in several model checkers [SSE03, HKK02, KK05, GB96, MRE96] that allow, among others: conformance checking [McM95b], analysis and synthesis of asynchronous circuits [KKY04], monitoring and diagnosis of discrete event systems [BHFJ03, CJ04], and analysis of asynchronous communication protocols [LI05].

1.3 Verification of infinite state systems

The verification of infinite-state systems is one of the most challenging research areas in formal and computer-aided verification. Being able to verify infinite-state systems is interesting not only due to the existence of complex systems that comprise unbounded variables, but also for several other reasons:

- Even though realizable, computer systems are finite in some sense, their size is often much larger than what can be handled by finite-state methods. Infinite-state models are good abstractions of large finite-state systems. Indeed, approximating a large finite domain by an unbounded one is usually more precise than imposing unrealistically small bounds on data values.

- Infinite-state systems are natural models of parameterized systems, when the range of parameter values is unbounded. It is often more comfortable to reason independently from any limit than to impose an arbitrary upper bound on the size of a system.

- The solutions developed for analyzing infinite-state systems are usually also applicable to systems whose state space is finite but very large.

In order to represent reactive systems as well as to extend the properties that could be checked, models for infinite-state systems were introduced. Most of them are based on a finite-state automata extended with unbounded data [AJ93, EFM99]: for instance, communicating finite-state machines [Boc78, BZ83] allow to model communication protocols, and Petri nets [Pet62] can represent systems with countably many resources. Most of the time, model checking is *undecidable* for infinite-state models since they can simulate Minsky machines.

However, for some classes of infinite-state systems, some problems remain decidable, such as the reachability problem for Petri nets [Kos82, May84]. This lead to many works on the identification of decidable subclasses of infinite-state models along with dedicated model-checking algorithms [Fin94, AJ96, CF97, HCF+02, Iba78, ISD+02, DJS99, EFM99, BM99, FS00b, FS00a, AD94, CJ99, LS02]. Another challenge is that most systems are heterogeneous: for instance checking the boundedness of communication channels may require to consider the channels themselves as well as the number of communicating processes, hence procedures specific to homogeneous systems do not apply. Fortunately, some of these techniques [KM69, AJ96] were found not to rely on the class of model, but rather on their structural properties, leading to the class of *well-structured transition systems* [Fin87, Fin90, AČJ00, FS01]. They form a subclass of infinite-state models including Petri nets and some of their extensions, lossy communicating machines, some process algebras, etc. The *well-preorder compatibility* over states/transitions in well-structured transition systems gives rise to a nice feature: only a finite prefix of the system's behaviors needs to be considered for concluding on the (in)satisfaction of several properties, particularly boundedness or termination.

1.4 Contributions

Our aim is the verification of concurrent systems that manipulate variables on unbounded domains, hence infinite-state systems. In this thesis, we explore the benefits of combining the efficient verification based on unfolding technique and decidability results on well-structured transition systems.

We provide a general framework for partial-order modeling of heterogeneous systems. We first show how labeled event structures [NPW80, Win82] can be used for efficient modeling of counters and FIFO channels. Moreover, the ideas that we used for the modeling of these two data types can be applied in other cases: they show how a collection of elements can be efficiently modeled, and how the order between these elements can be taken into account if needed. The modelization is thus no more on the system level, but on the behavioral level.

The labeled event structures as behavioral/branching/partially-ordered models are strictly related to system/linear/interleaving models generally used in model-checking such as labeled transition systems. We also give a strict correspondence between these two kinds of models by means of *coherence* property. However, the advantage of the first model, hence our choice, in comparison with the second one is that it can be directly used for verification. Moreover, because of the partial-order inside, labeled event structures are generally compact, and this fact makes such verifications more efficient. It is worth noticing that our labeled event structures without restraint on their labeling functions may be nondeterministic, i.e. a marking may correspond to a potentially infinite set of system's states. As a consequence, they are general enough to be combined with symbolic techniques.

The unfolding technique, initially developed for systems modeled as Petri nets [McM95a], requires a formalism having a notion of concurrent components; in particular, the formalism should allow us to determine for each action of the system which components participate in the action and which ones remain idle. The most straightforward approach in order to apply the unfolding technique for concurrent systems is Arnold and Nivat's synchronized products of labeled transition systems [AN82, Arn92]. Loosely speaking, in this formalism, components are modeled as labeled transition systems and may execute joint actions by means of a very general synchronization mechanism. The

result in [ER99] makes it clear that the unfolding technique is not tied to a particular formalism, although its details may depend on the formalism to which it is applied. However, this turns out not to be satisfactory: imagine that one models a counter by an interleaving model such as a labeled transition system, then if, say, three different processes want to increase the counter, their actions will get interleaved. As in principle those actions are independent, we lose a good deal of concurrency present in the original system. Our solution is to model a concurrent system by synchronized product of (heterogeneous) components where the semantics of components is given in terms of labeled event structures. Hence, when applying the unfolding technique, one takes advantage of the intrinsic concurrency in each component.

Our synchronized products of labeled event structures provide more information than Petri nets about the structure of the system. In particular, we show that a Petri net may be considered as the parallel composition of its places viewed as counters. Moreover, we show that synchronized products of (labeled) event structures conform to event structure semantics [Win82], and are thus (labeled) event structures themselves. Hence, one easily obtain a *hierarchical modelization* for complex systems.

Although labeled event structures are compact representations of systems, they are in general infinite due to the existence of infinite computations. However, research on verification of finite systems shows that properties can be checked using certain finite prefixes, called complete prefixes, of their state-space. Of course, there is no hope to have a notion of complete prefix for infinite systems. There is hope though when such systems have a weak simulation relation that is a well-preorder over states. We show how to adapt the results in [Fin87, FS01] to labeled event structures. We focus on the following four verification problems: *termination, boundedness, quasi-liveness* and *sub-covering*, that can be fully decided with algorithms. In other words, verification algorithms have theoretical termination guarantee.

We show a way to deduce a (well-)preorder over configurations of labeled event structures from the one over system states. Notice that, since a configuration may correspond to several states, these two preorders are quite different. They coincide only for deterministic labeled event structures. We give a notion of *compatibility* of transitions/events w.r.t. these preorders. Therefore, once the relation between preorders are determined, one may switch corresponding compatibilities back and forth between system/behavior models. We also show preservation of (well-)preorders as well as compatibilities under parallel composition using synchronized products.

Based on the truncation criteria in the unfolding technique, we propose a general definition of a *cutting-context* for well-preordered labeled event structures and show that they admit a finite prefix preserving one or more properties depending on the system's structure. Particularly, we give appropriate cutting-contexts for the verification problems that we focus on: termination, boundedness, quasi-liveness and sub-covering. Remark that these results cannot be directly obtained from previous techniques on well-structured systems since one needs to take into account the partial-order between events. We clarify the difference between global and local cutting-contexts. The former one is similar to techniques used in interleaving models [Fin91, FS01] while the other one respects the key idea of the unfolding technique, and is thus more suitable to partial-order verifications. Although our technique is based on forward partial-order analysis, we show that standard backward analysis techniques (see e.g. [AJ96]) could be embedded. The intuitive idea comes from the duality in the category of (labeled) transition systems.

As we use behavior models for systems, the model checking algorithms consist in constructing such models. In general, reachability-based properties may be checked

on-the-fly. We present a generalization of the unfolding algorithm in [McM95a, ER99] to parallel composition of labeled event structures. The idea is that, one iteratively tries to enlarge some prefix by adding new possible events without looking at global configurations/states. We detail our unfolding algorithm into two particular cases: for component labeled event structures and for synchronized products of them. In the first case, once a component is given by some labeled event structure, its events form somehow concrete patterns. By analyzing such patterns, one may obtain simple unfolding algorithms appropriate for the component. We propose algorithms for counters and FIFO channels and some of their variants. Other labeled event structures for standard components may have dedicated algorithms in the same manner. In the second case, the unfolding algorithm takes the notion of concurrent component given by synchronization constraints into account. The process of finding events to be added must consider the associated component prefixes.

It is worth noticing that, in unfolding algorithms for Petri nets or synchronized products of interleaving models, adding events depends on component states. But, in our algorithm for synchronized products of labeled event structures, we synchronize components' events. This task must be accompanied with an additional checking in order to see if such a synchronization satisfies the componentially downward-closed property in the synchronization product. The advantage of our technique is that it allows the intrinsic concurrency in components to be preserved. Moreover, we show that the labeled event structure for a complex system may be algorithmically constructed in a hierarchical way. It is not necessary to compute component labeled event structures prior to unfolding synchronized products. Indeed, the synchronized product's prefix and its component prefixes are constructed together on-the-fly, and component ones are extended when needed.

Finally, as a practical contribution, we have implemented a model checker, Esu, that runs our algorithms. It has been written in *OCaml*. To our knowledge, Esu is the first tool that combines the unfolding technique and decidable results on well-preordered systems. We also analyse the results obtained using Esu and some other well-established tools such as Pep and Tina to compare the benefits of different methods. In addition, a heuristic technique is integrated in Esu to generate more compact truncations. Although such truncations do not preserve behaviors of the system in terms of Mazurkiewicz's trace semantics, they are enough to check reachability-based properties that we are interested in. Experimental results show that this technique is promising since, for certain benchmark examples, we obtain truncations of which the size does not exceed the number of the system's reachable states.

1.5 Organization of the thesis

This thesis is organized as follows.

Chapter 2 provides basic notions that will be used along the thesis. We also introduce two well-known models: labeled transition systems and Petri nets, as well as their semantics.

Chapter 3 presents labeled event structures, based on prime event structures, together with their properties. We motivate the choice of this model and briefly compare it to other types of event structures. We define a strict correspondence between labeled event structures and labeled transition systems modeling the same system.

The major part of this chapter is dedicated to modeling concurrent systems. Labeled event structures for standard systems such as counters and FIFO channels are given. Their variants adapting to boundedness or different initial values intuitively demonstrate the ease of this modelization approach. We define synchronized products of labeled event structures and show how to use it for modeling concurrent systems as a hierarchical structure. In fact, a Petri net may be considered as some synchronized product of its places, and each place, in its turn, is similar to a counter.

Chapter 4 addresses the truncation technique to obtain complete prefixes of labeled event structures w.r.t. a given verification problem. We first define (well-)preorder and compatibility on labeled event structures that may be deduced from the one on labeled transition systems. We show that well-preorder and compatibility are preserved in parallel composition.

Cutting-contexts for interesting verification problems such as boundedness, termination, quasi-liveness and sub-covering, are given. The decidability of these problems on well-preordered labeled event structures are proved. We also study a technique for adapting our forward analysis in order to obtain the same results as some backward analysis.

Chapter 5 describes our general unfolding algorithm. We then detail it into two cases: for synchronized products of labeled event structures, and for their components that, of course, may be synchronized products. Appropriate algorithms for counters, FIFO channels, and even arbitrary systems (that have no local concurrency) are given. The correctness and termination of all these algorithms are proved.

In addition, we show that truncation technique can be integrated into the unfolding algorithm. Hence, the verification may be done together with the construction of labeled event structures.

Chapter 6 illustrates a methodology for modeling of heterogeneous systems on the example of the *Alternating Bit Protocol*, and explains how to verify interesting properties. Then, we briefly describe our model-checker Esu. Experimental results on standard benchmark examples are given. We also compare its results to the ones obtained by using well-established tools: Pep and Tina.

Moreover, the well-known problem, called auto-concurrency problem, of unfolding techniques for Petri nets is discussed. We show that its negative effect may be well reduced by using our heuristic technique and as a consequence, the obtained prefix is more compact but still preserves enough information for some reachability-based properties.

Chapter 7 concludes the thesis and presents some perspectives of our work.

Chapter 2

Preliminaries

Contents

2.1 Relations and functions

We use standard notations on sets. The *power set* of a set X, written $\mathcal{P}(X)$, is the set of all subsets of X, and X is called the *base set* of $\mathcal{P}(X)$. Any subset F of $\mathcal{P}(X)$ is called a *family of sets* over X. We denote $\mathcal{P}_f(X)$ the family which contains all finite subsets of X. A set of cardinal one, $X = \{x\}$ for some element x, is called a *singleton*. We notationally identify a singleton X by its only element x if there is no risk of confusion.

A *relation* $\mathcal{R}$ between two sets X and Y is a subset of the Cartesian product $X \times Y$. Let X' be a subset of X, the *(left-)restriction* of $\mathcal{R}$ to X' is another relation $\mathcal{R}'$ between X' and Y defined by $\mathcal{R}' = \{(x, y) \in \mathcal{R} \,/\, x \in X'\}$.

Notation 2.1.1. The restriction of $\mathcal{R}$ to X' is denoted by $\mathcal{R}|_{X'}$.

We denote $x \mathcal{R} y$ the fact that $(x, y) \in \mathcal{R}$. The *converse relation* of $\mathcal{R}$, denoted by either $\overleftarrow{\mathcal{R}}$ or $\mathcal{R}^{-1}$, is a relation between Y and X defined by $\{(y, x) \in Y \times X \,/\, x \mathcal{R} y\}$. The complement of $\mathcal{R}$ is denoted either by $\not\mathcal{R}$ or $\overline{\mathcal{R}}$, i.e. $\not\mathcal{R} = \overline{\mathcal{R}} = (X \times Y) \setminus \mathcal{R}$. For a given $x \in X$, the set of all elements $y \in Y$ satisfying $x \mathcal{R} y$ is denoted by $\mathcal{R}(x)$, and moreover it induces naturally the same notation on subsets of X.

Notation 2.1.2. For any subset $X' \subseteq X$, $\mathcal{R}(X') = \bigcup_{x \in X'} (\mathcal{R}(x))$.

A *relation* $\mathcal{R}$ between X and Y is *single-valued* if $\mathcal{R}$ pairs $x \in X$ with at most one $y \in Y$, i.e. for all $x \in X$, $|\mathcal{R}(x)| \leq 1$; and $\mathcal{R}$ is *total* for all $x \in X$, there exists $y \in Y$ such that $x \mathcal{R} y$. A function $\mathcal{F}$ from X to Y is any total and single-valued relation between

X and Y. We write $\mathcal{F} : X \to Y$ and call X, Y respectively the *domain* of $\mathcal{F}$, denoted by $\mathsf{Dom}(\mathcal{F})$, and the *codomain* of $\mathcal{F}$, denoted by $\mathsf{Codom}(\mathcal{F})$.

A function $\mathcal{F} : X \to Y$ is *bijective* if it is

- *injective*: $\forall x, x' \in X, \mathcal{F}(x) = \mathcal{F}(x') \Rightarrow x = x'$; and
- *surjective*: $\forall y \in Y, \exists x \in X : \mathcal{F}(x) = y$.

A bijective function is also called a *bijection* or *one-to-one* function.

A *binary relation* $\mathcal{R}$ on a set X is a relation between X and X, i.e. $\mathcal{R} \subseteq X \times X$. Binary relation $\mathcal{R}$ is

- *reflexive* (*irreflexive*) if $x \mathcal{R} x$ (not $x \mathcal{R} x$, resp.) for all $x \in X$,
- *transitive* if for all $x, y, z \in X$, $x \mathcal{R} y$ and $y \mathcal{R} z$ imply $x \mathcal{R} z$,
- *symmetric* (*asymmetric*) if for all $x, y \in X$, $x \mathcal{R} y$ implies $y \mathcal{R} x$ ($y \mathcal{R} x$, resp.),
- *antisymmetric* if for all $x, y \in X$, $x \mathcal{R} y$ and $y \mathcal{R} x$ imply $x = y$.

The *identity relation* over X is the set $\mathcal{I}_X = \{(x, x) \,/\, x \in X\}$. It is thus a reflexive, transitive, symmetric and antisymmetric binary relation. Given a subset $X' \subseteq X$, the *restriction* of $\mathcal{R}$ to X', denoted by $\mathcal{R}|_{X'}$, is the set of all pairs $(x, y) \in \mathcal{R}$ for which both x and y are in X'. Formally, $\mathcal{R}|_{X'} = \mathcal{R} \cap (X' \times X')$.

The *transitive closure* of a binary relation $\mathcal{R}$ on X, denoted by $\mathcal{R}^+$, is the smallest transitive relation on X which contains $\mathcal{R}$. Relation $\mathcal{R}^+$ exists and is unique (as stated) for any binary relation $\mathcal{R}$. The transitive closure $\mathcal{R}^+$ may be defined as follows: $\forall x, y \in X$, $x \mathcal{R}^+ y$ iff there exists a non-empty and finite sequence of element $x_1, \ldots, x_n \in X$ such that $x = x_1, x_1 \mathcal{R} x_2, \ldots, x_{n-1} \mathcal{R} x_n$, and $x_n \mathcal{R} y$ (this condition can be simply written as $x = x_1 \mathcal{R} x_2 \ldots \mathcal{R} x_n \mathcal{R} y$). The *reflexive and transitive closure* of a binary relation $\mathcal{R}$ on X, denoted by $\mathcal{R}^*$, is the binary relation $\mathcal{I}_X \cup \mathcal{R}^+$. Notice that the (reflexive and) transitive closure of a (reflexive and) transitive relation $\mathcal{R}$ is $\mathcal{R}$ itself.

2.2 Alphabet and words

Let A be an *alphabet*, i.e. a finite set of *symbols*, a finite (infinite) *word* w over A is any finite (infinite, resp.) sequence of symbols in A. We denote by A^* (A^ω) the set of all finite (infinite, resp.) words over A. The length of a word w is denoted by $|w|$, and we also use ω to denote an infinite length. And ε is the *empty-word*, whose length is equal to zero. Moreover, given any $I \subseteq (\mathbb{N} \cup \omega)$, we use notation A^I to denote a subset of words based on words' length. Formally, A^I is the set of all words in $(A^* \cup A^\omega)$ whose length is in I, i.e. $A^I = \{w \in (A^* \cup A^\omega) \,/\, |w| \in I\}$.

For two words $w \in A^*$, $w' \in (A^* \cup A^\omega)$, we let $w.w'$ denote the concatenation of w and w'. A finite word $w \in A^*$ is a *prefix* of a word $w' \in (A^* \cup A^\omega)$ if there exists another word $w'' \in (A^* \cup A^\omega)$ satisfying $w' = w.w''$. Similarly, the word w'' is then called a *suffix* of w', and we write $w'' = (w^{-1})w'$.

Let A, B be two alphabets, and $\mathcal{L}$ be a function $\mathcal{L} : A \to B$. We define the function $\mathcal{L}^{\mathcal{W}}$ on words over A which is based on $\mathcal{L}$, $\mathcal{L}^{\mathcal{W}} : (A^* \cup A^\omega) \to (B^* \cup B^\omega)$, as follows:

- $\mathcal{L}^{\mathcal{W}}(\varepsilon) = \varepsilon$, and
- $\mathcal{L}^{\mathcal{W}}(w.w') = \mathcal{L}^{\mathcal{W}}(w).\mathcal{L}^{\mathcal{W}}(w')$ for all $w \in A^*, w' \in (A^* \cup A^\omega)$.

Definition 2.2.1 (Subword order)**.** Let M be an alphabet. The *subword order* $\preccurlyeq$ over M^* is defined by: for all $w = m_1 m_2 \ldots m_n \in M^*$, for all $w' \in M^*$, $w' \preccurlyeq w$ iff $w' = m_{i_1} m_{i_2} \ldots m_{i_k}$ for some $k \leq n$ and $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

For every word $w \in M^*$, its prefixes as well as its suffixes are particular subwords of w itself.

2.3 Orders

A *preorder* $\leq$ on a set X is any reflexive and transitive binary relation on X.

Definition 2.3.1 (Well-preorder)**.** A preorder $\preccurlyeq$ on a set X is a *well-preorder* (*converse well-preorder*) if every infinite sequence $x_1, x_2, \ldots, x_k, \ldots$ of elements in X must contain an increasing (decreasing, resp.) pair $x_i \preccurlyeq x_j$ ($x_i \succcurlyeq x_j$, resp.) where $i < j$.

A *partial order* $\leq$ is an antisymmetric preorder. For example, one can deduce that the subword order over some finite alphabet M defined in the previous sub-section is a partial-order on M^*.

Definition 2.3.2 (poset)**.** A *partially ordered set (poset)* is a pair $(X, \leq)$ where X is a set and $\leq$ is a partial order on X.

Lemma 2.3.3. *Let $(X, \leq)$ be a poset. For every subset Y of X, $(Y, \leq|_Y)$ is a poset.*

Proof. By definition, $\leq|_Y$ is also a partial order on X as well as on $Y \subseteq X$ because the reflexivity, transitivity and antisymmetry of $\leq$ are all preserved on $\leq|_Y$. □

A *total order* $\trianglelefteq$ on X is a partial order such that for all $x, y \in X$, we have either $x \trianglelefteq y$ or $y \trianglelefteq x$. A *strict partial order* on X is any irreflexive and transitive (and therefore antisymmetric) binary relation on X. Every partial order $\leq$ on X corresponds to one and only one strict partial order on X, denoted by $<$, which is defined as $< = (\leq \setminus \mathcal{I}_X)$.

Given two binary relations $\mathcal{R}$ and $\mathcal{R}'$ on a set X, we say that $\mathcal{R}$ *refines* $\mathcal{R}'$ if whenever $x \mathcal{R}' y$ it also holds that $x \mathcal{R} y$. In other words, $\mathcal{R}$ contains $\mathcal{R}'$, i.e. $\mathcal{R}' \subseteq \mathcal{R}$. A *linear extension* of a partial order $\leq$ on X is any total order $\trianglelefteq$ on X which refines the partial order $\leq$.

Definition 2.3.4 (Linearisation)**.** Let $(X, \leq)$ be a poset and Y be a subset of X. A *linearisation* of Y w.r.t. $\leq$ is any sequence containing all elements $y_1, y_2, y_3, \ldots$, of Y such that $y_1 \lhd y_2, y_2 \lhd y_3, \ldots$, for some linear extension $\trianglelefteq$ of $\leq|_Y$.

Remark: We sometimes represent a linearisation $y_1, y_2, y_3, \ldots$ of Y by the corresponding word $w = y_1.y_2.y_3 \ldots \in (Y^* \cup Y^\omega)$.

Let $(X, \leq)$ be a poset and Y be a subset of X, then $y \in Y$ is a *minimal (maximal) element* of Y w.r.t. $\leq$ if for all $y' \in Y$, $y' \leq y$ ($y \leq y'$, respectively) implies $y' = y$. And $\leq$ is *well-founded (converse well-founded)* if every non-empty subset $Y \subseteq X$ has a minimal element w.r.t. $\leq$.

Minimal (maximal) elements of a subset Y need not exist (for example, when $\leq$ is not well-founded) and there may be many minimal (maximal) elements. We denote the set of minimal (maximal) elements of a subset Y (w.r.t. $\leq$) by $\mathsf{Min}_{\leq}(Y)$ ($\mathsf{Max}_{\leq}(Y)$, respectively). If $x, y \in \mathsf{Min}_{\leq}(Y)$, $x \neq y$ implies that neither $x \leq y$ nor $y \leq x$.

Since $\leq$ is a binary relation between X and itself, given a subset $Y \subseteq X$, $\leq(Y)$ contains all $x \in X$ satisfying: there exists $y \in Y$ where $y \leq x$. The set $\leq(Y)$ is called the *upward closure* of $Y \subseteq X$ w.r.t. the poset $(X, \leq)$. Similarly, we have the *downward closure* of Y, denoted by $\geq(Y)$. A set Y is *upward closed* (*downward closed*) w.r.t. $(X, \leq)$ if it is equal to its upward closure (downward closure, resp.), i.e. $Y = \leq(Y)$ ($Y = \geq(Y)$, resp.).

Definition 2.3.5 (DAG)**.** A *directed acyclic graph (DAG)* is a pair (V, E) in which:

- V is called the set of *vertices*,

- relation $E \subseteq V \times V$ is called the set of *directed edges*, and
- E^+ is irreflexive.

We say that there exists a *(directed) path* from a vertex v to another vertex v' if $v\, E^+ v'$. The last property in Definition 2.3.5 intuitively means that there is no (non-empty) directed path that starts and ends on a same vertex. Each DAG gives rise to a partial order $\leq$ on its vertices. Formally, if (V, E) is a DAG then $(V, (E^+ \cup \mathcal{I}_V))$ is a poset, where $\mathcal{I}_V$ is the identity relation on V.

Reversely, a poset $(X, \leq)$ is usually represented by and computed from one of DAGs (X, E) (there are many) which corresponds to the poset, i.e. $E^+ = < = (\leq \setminus \mathcal{I}_X)$. For the sake of simplicity, the relation E would be as small as possible. If $\leq(x)$ is finite for all $x \in X$, $\leq$ gives rise to a unique minimal relation $\lessdot$ so that $\lessdot^+ = <$. Therefore, in this thesis, we always choose the minimal relation $\lessdot$ for representing $(X, \leq)$ as well as graphically illustrating it afterward.

Notation 2.3.6. For a given poset $(\leq, X)$, we denote $\lessdot$ the relation

$$\bigcap_{\mathcal{R} \subseteq (X \times X) \,:\, \mathcal{R}^* = \leq} \mathcal{R}$$

Lemma 2.3.7. *If $\geq(x)$ is finite for all $x \in X$, then $\lessdot^* = \leq$.*

Proof. Let $\mathcal{S}$ denote the set of all binary relation $\mathcal{R}$ on X satisfying $\mathcal{R}^* = \leq$. Due to the transitive property of $\leq$, we have $\leq^* = \leq$, and consequently, $\leq \in \mathcal{S}$. Therefore, $\lessdot \subseteq \leq$ and $\lessdot^* \subseteq \leq$. We only need to prove that $\lessdot^* \supseteq \leq$, that means, for all $x, y \in X$, if $x \leq y$ then $x \lessdot^* y$ (1) by induction on the size of $\geq(y)$ because it is finite. In the base case, when $|\geq(y)| = 1$, we must have $x = y$ and obviously $x \lessdot^* y$. In general, if $x = y$ then (1) is also true, otherwise, i.e. $x < y$, there are two sub-cases.

First, if there does not exists any element $z \in X$ such that $x < z$ and $z < y$ (2), let $\mathcal{R}$ be any relation in $\mathcal{S}$. Since $\mathcal{R}^* = \leq \ni \langle x, y \rangle$, there exists a finite sequence $x = x_0 \,\mathcal{R}\, x_1 \,\mathcal{R}\, \ldots \,\mathcal{R}\, x_n = y$ where $n \in \mathbb{N}$ and $x_0, x_1, \ldots, x_n$ are pairwise different. Moreover, n is not equal to 0 because $x \neq y$. If $n = 1$ then $x \,\mathcal{R}\, y$. Otherwise, there exists an index i such that $i \in \{1, 2, \ldots, n-1\}$ and $x = x_0 \,\mathcal{R}^+\, x_i \,\mathcal{R}^+\, x_n = y$. As a consequence, $x = x_0 \leq x_i \leq x_n = y$. It follows from $x_0 \neq x_i \neq x_n$ that $x < x_i < y$. It contradicts to (2). Therefore, we have $x \,\mathcal{R}\, y$ for all $\mathcal{R} \in \mathcal{S}$, and consequently, $x \lessdot y$.

Second, if there exists $z \in X$ such that $x < z < y$. Assume that z is chosen so that there is no other element $z' \in X$ satisfying $z < z' < y$. Such an element z must exist because otherwise, one thus obtains an infinite sequence $z' < z'' < \ldots < y$. And it contradicts to the finiteness of the set $\geq(y)$. As in the previous sub-case, we have $z \lessdot y$. Moreover, since $|\geq(z)|$ is less than $|\geq(y)|$, by the induction hypothesis (1), we have $x \lessdot^* z$, and consequently, $x \lessdot^* y$.

Therefore, in both sub-cases, we have $x \lessdot^* y$. This lemma is proved. □

2.4 Labeled transition systems

Definition 2.4.1. A *labeled transition system* is a quadruple $\mathcal{LTS} = (S, \Sigma, \rightarrow, s^0)$ where:

- S is a (potentially infinite) set of *states*,
- Σ is a finite set of *actions*,
- the *labeled transition relation* $\rightarrow$ is any subset of $S \times \Sigma \times S$, and

- $s^0 \in S$ is the *initial state*.

Intuitively, one can evolve from a state $s \in S$ to another state $s' \in S$ due to a *transition* which is accomplished by an action $a \in \Sigma$, i.e. $\langle s, a, s' \rangle \in \rightarrow$. It is thus reasonable to write such a transition as $s \xrightarrow{a} s'$. Formally, for any action $a \in \Sigma$, $\xrightarrow{a}$ is a binary relation on S, defined as $\xrightarrow{a} = \{(s, s') \in S \times S \,/\, \langle s, a, s' \rangle \in \rightarrow\}$. We say that action a is *enabled* from the state s and simply write $s \xrightarrow{a}$.

Remark: The classical model called *transition system* can always be considered as labeled transition system without labeling transitions by actions. A labeled transition system $(S, \Sigma, \rightarrow_{\mathcal{LTS}}, s^0)$ gives rise to a transition system $(S, \rightarrow_{\mathcal{TS}}, s^0)$ where $\rightarrow_{\mathcal{TS}} \subseteq S \times S$ such that for all $s, s' \in S$, $s \rightarrow_{\mathcal{TS}} s'$ iff there exists $a \in \Sigma$ satisfying $s \xrightarrow{a}_{\mathcal{LTS}} s'$.

Example 2.4.2. The counter initialized by 1 is the labeled transition system $\mathcal{CT} = (\mathbb{N}, \{+, -\}, \rightarrow, 1)$ where its labeled transition relation $\rightarrow$ is defined by $\{\langle n, +, n+1 \rangle | n \in \mathbb{N}\} \cup \{\langle n+1, -, n \rangle | n \in \mathbb{N}\}$.

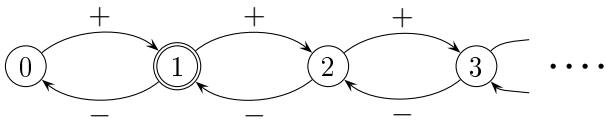


Figure 2.1: Graphical representation of the counter in Example 2.4.2

Figure 2.1 illustrates the counter initialized by 1 in Example 2.4.2 (see Section 3.3.2 for formal definition of counter's family and detailed explications) by a directed graph. States are represented by circles and every transition $\langle s, a, s' \rangle \in \rightarrow$ is represented by an arrow leading from s to s' labeled by a. The double frame of the circle corresponding to state 1 indicates that it is the initial state of the system.

Example 2.4.3. A FIFO (First-In-First-Out) channel in which we can send messages ranging over $M = \{a, b\}$ and receive messages in its sending order, can be modeled by $\mathcal{FF} = (S, \Sigma, \rightarrow, s^0)$ where:

- $S = M^*$: each state is a finite word over M,
- $\Sigma = \{!a \,/\, a \in M\} \cup \{?a \,/\, a \in M\}$: action $!a$ ($?a$) means sending (receiving, resp.) message a into (from, resp.) the channel,
- $\rightarrow = \{\langle w, !a, w.a \rangle \,/\, w \in M^*, a \in M\} \cup \{\langle a.w, ?a, w \rangle \,/\, w \in M^*, a \in M\}$
- $s^0 = a$: there are initially a message a and a message b in the channel.

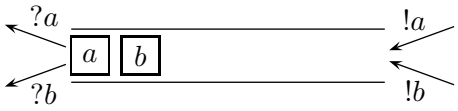


Figure 2.2: The FIFO channel in Example 2.4.3

Remark: By sending actions, the environment inserts messages into the channel, and conversely, it removes messages from the channel by receiving actions. *Sending* and *receiving* intuitively mean actions of the environment and not the ones of the channel. This naming is naturally convenient while using channels in modeling complex systems by synchronized product (see Section 3.3.4).

2.4.1 Behaviors and properties

Definition 2.4.4. Let $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ be a labeled transition system. A *finite path* (resp. *infinite path*) in $\mathcal{LTS}$ is any finite (resp. infinite) sequence $\pi = s_1 \xrightarrow{a_1} s'_1, s_2 \xrightarrow{a_2} s'_2, \ldots, s_k \xrightarrow{a_k} s'_k, \ldots$ of transitions such that $s'_{i-1} = s_i$ for every index $i > 1$ in the sequence.

We shortly write $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots s_k \xrightarrow{a_k} s_{k+1} \ldots$, and we say that π *starts in* s_1. The transition relation ($\rightarrow$) is extended to its transitive closure ($\twoheadrightarrow$).

Notation 2.4.5. Given a word $\sigma = a_1.a_2 \ldots a_k \in \Sigma^*$, $s_1 \xrightarrow{\sigma} \!\!\!\!\rightarrow s_{k+1}$ means that there is such a path π from s_1 to s_{k+1}.

Let $\pi = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots s_k \xrightarrow{a_k} s_{k+1} \ldots$ be a path of a labeled transition system $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$. π is called an *execution* if it starts in the initial state of $\mathcal{LTS}$, i.e. $s_1 = s^0$. The word $a_1.a_2 \ldots a_k$ is called a *firing sequence*, and s_{k+1} is called *reachable* by execution π.

The *reachability set* of a labeled transition system $\mathcal{LTS}$, denoted by $\mathsf{post}^*_{\mathcal{LTS}}$, is the set of all reachable states of $\mathcal{LTS}$.

Definition 2.4.6. Let $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ be a labeled transition system, and $S' \subseteq S$ be some subset containing the initial state s^0. The *restriction of* $\mathcal{LTS}$ *to* S', denoted by $\mathcal{LTS}|_{S'}$, is the labeled transition system $\mathcal{LTS}' = (S', \Sigma, \rightarrow', s^0)$ where $\rightarrow' = \rightarrow \cap (S' \times \Sigma \times S')$.

Given two labeled transition systems $\mathcal{LTS}$ and $\mathcal{LTS}'$, when their restrictions on their reachable states $\mathsf{post}^*_{\mathcal{LTS}}$ and $\mathsf{post}^*_{\mathcal{LTS}'}$ are the same, i.e. $\mathcal{LTS}|_{\mathsf{post}^*_{\mathcal{LTS}}} = \mathcal{LTS}'|_{\mathsf{post}^*_{\mathcal{LTS}'}}$, they are intuitively interchangeable.

Definition 2.4.7. Let $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ be a labeled transition system. $\mathcal{LTS}$ is:

- *finite* (*infinite*) if its set of states S is finite (infinite, respectively);
- *deterministic* if for all action $a \in \Sigma$, $\xrightarrow{a}$ is single-valued function, i.e. $s \xrightarrow{a} s'$ and $s \xrightarrow{a} s''$ imply $s' = s''$; and
- *finitely-branching* if for all state $s \in S$, the set $\{s' \in S \,/\, \exists a \in \Sigma, s \xrightarrow{a} s'\}$ is finite.

Because the set of actions Σ in Definition 2.4.1 is finite, we have:

Corollary 2.4.8. *Deterministic labeled transition systems are finitely-branching.*

Proof. Obvious due to the finiteness of the action set Σ in Definition 2.4.1. □

Example 2.4.9. The counter in Example 2.4.2 and FIFO channel in Example 2.4.3 are both finite, deterministic and finitely-branching labeled transition systems.

2.4.2 Synchronized products of labeled transition systems

We now present a composition primitive that we use to build complex systems from basic components: the synchronized product of labeled transition systems [ABC94]. In a synchronized product, components must behave according to so-called synchronization vectors.

Notation 2.4.10. Given a number $n \in \mathbb{N}$ and n sets $X_1, X_2, \ldots, X_n$, we denote the n-dimension space $X_1 \times X_2 \times \ldots \times X_n$ by $\otimes(X_1, X_2, \ldots, X_n)$. When $\varepsilon \notin X_i$ for all i in $\{1, 2, \ldots, n\}$, we denote the n-dimension space $(X_1 \cup \{\varepsilon\}) \times (X_2 \cup \{\varepsilon\}) \times \ldots \times (X_n \cup \{\varepsilon\})$ by $\otimes_\varepsilon(X_1, X_2, \ldots, X_n)$.

Notation 2.4.11 (Component restriction). Given n sets $X_1, X_2, \ldots, X_n$. For all tuple $x = \langle x_1, x_2, \ldots, x_n \rangle \in \otimes(X_1, X_2, \ldots, X_n)$ and for all $i \in \{1, 2, \ldots, n\}$, we call x_i the *component restriction* onto i of x, and denote it by $x\downarrow_i$. Therefore, the component restriction onto i of a subset $Y \subseteq X$ is the set $\{x\downarrow_i \ / x \in Y\}$, and is denoted by $Y\downarrow_i$.

Given n labeled transition systems $\mathcal{LTS}_1, \mathcal{LTS}_2, \ldots, \mathcal{LTS}_n$ where $\mathcal{LTS}_i = (S_i, \Sigma_i, \rightarrow_i, s_i^0)$, $i \in \{1, 2, \ldots, n\}$. A *synchronization vector* is any n-tuple v in $\otimes_\varepsilon(\Sigma_1, \Sigma_2, \ldots, \Sigma_n)$, and a *synchronization constraint* is any subset $\Sigma_{\mathcal{SP}} \subseteq \otimes_\varepsilon(\Sigma_1, \Sigma_2, \ldots, \Sigma_n)$ of synchronization vectors. Intuitively, a label a in a synchronization vector means that the corresponding component must take a transition labeled by a, whereas an ε means that the component must not move. The synchronized product is formally a labeled transition system in which the set of actions, called *global actions*, are determined by synchronization constraints.

Definition 2.4.12. Given n labeled transition systems $\mathcal{LTS}_i = (S_i, \Sigma_i, \rightarrow_i, s_i^0)$ where i ranges over $\{1, \ldots, n\}$ and a synchronization constraint $\Sigma_{\mathcal{SP}} \subseteq \otimes_\varepsilon(\Sigma_1, \Sigma_2, \ldots, \Sigma_n)$. The *synchronized product* of $\mathcal{LTS}_1, \mathcal{LTS}_2, \ldots, \mathcal{LTS}_n$ with respect to $\Sigma_{\mathcal{SP}}$ is the labeled transition system $\mathcal{SP} = (S_{\mathcal{SP}}, \Sigma_{\mathcal{SP}}, \rightarrow_{\mathcal{SP}}, s_{\mathcal{SP}}^0)$ defined by:

- $S_{\mathcal{SP}} = \otimes(S_1, S_2, \ldots, S_n)$,
- for all $s, s' \in S_{\mathcal{SP}}, a \in \Sigma_{\mathcal{SP}}$: $s \xrightarrow{a}_{\mathcal{SP}} s'$ iff, for every $i \in \{1, \ldots, n\}$, $s\downarrow_i \xrightarrow{a\downarrow_i} s'\downarrow_i$ (notice that $s\downarrow_i \xrightarrow{\varepsilon} s'\downarrow_i$ simply means that $s\downarrow_i = s'\downarrow_i$), and
- $s_{\mathcal{SP}}^0 = \langle s_1^0, \ldots, s_n^0 \rangle$.

Example 2.4.13. Let $\mathcal{CT}_1 = (\mathbb{N}, \{+, -\}, \rightarrow, 0)$, $\mathcal{CT}_2 = (\mathbb{N}, \{+, -\}, \rightarrow, 1)$, and $\mathcal{CT}_3 = (\mathbb{N}, \{+, -\}, \rightarrow, 1)$ be three counters defined as in Example 2.4.2 with only difference in its initial states. Let $\Sigma_{\mathcal{SP}}$ be the synchronization constraint defined as: $\Sigma_{\mathcal{SP}} = \{\langle -, +, + \rangle, \langle +, -, + \rangle, \langle \varepsilon, \varepsilon, - \rangle\}$. The semantics of the synchronized product of $\mathcal{CT}_1, \mathcal{CT}_2$, and $\mathcal{CT}_3$ with respect to $\Sigma_{\mathcal{SP}}$ is illustrated in Figure 2.3 where $+_{\mathcal{SP}}, +'_{\mathcal{SP}}, -_{\mathcal{SP}}$ are respectively abbreviations for global actions $\langle -, +, + \rangle, \langle +, -, + \rangle, \langle \varepsilon, \varepsilon, - \rangle$.

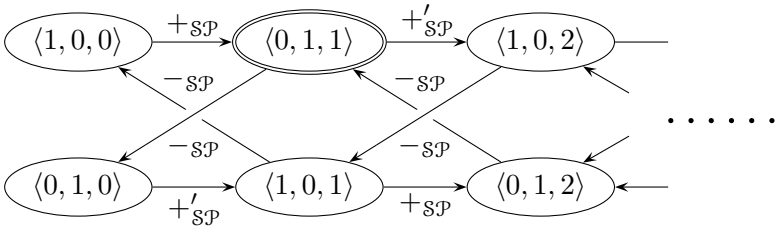


Figure 2.3: A synchronized product of three counters.

Remark: Synchronized product of labeled transition systems is a labeled transition system. This synchronized product can be a component labeled transition system of another synchronized product. In other words, synchronized products give us a way to hierarchically model complex systems.

Lemma 2.4.14. *Synchronized product of labeled transition systems is finite, deterministic, finitely-branching if its components are all finite, deterministic, finitely-branching, respectively.*

2.4.3 Simulation

A labeled transition system is an abstract model of some real system in which one is only interested in certain actions or behaviors. Hence, a real system gives rise to many labeled transition systems which may be pairwise different on the abstract level. Among them, a labeled transition system can simulates another one, it means that every behavior of the second one is also a behavior of the first one, called an abstraction. And both of them simulate the real system. The term "simulation" is used as in "this program simulates the process of people making decisions".

The definition of labeled transition systems immediately suggests a notion of simulation: initial states must be mapped to initial states, and for every action the first labeled transition system can perform in a given state, it must be possible for the second labeled transition system to perform the corresponding action, if any, from the corresponding state.

Definition 2.4.15 (Simulation)**.** Let $\mathcal{LTS}_1 = (S_1, \Sigma_1, \rightarrow_1, s_1^0)$ and $\mathcal{LTS}_2 = (S_2, \Sigma_2, \rightarrow_2, s_2^0)$ be two labeled transition systems. A *simulation relation* from $\mathcal{LTS}_1$ to $\mathcal{LTS}_2$ is a pair $\mathcal{R} = (\mathcal{R}_S, \mathcal{R}_\Sigma)$ where $\mathcal{R}_S \subseteq S_1 \times S_2$ and $\mathcal{R}_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ are two relations, such that:

- $s_1^0 \,\mathcal{R}_S\, s_2^0$, and
- for all $\langle s_1, a_1, s_1' \rangle \in \rightarrow_1, s_2 \in S_2,\ s_1 \,\mathcal{R}_S\, s_2$ implies that there exists a transition $\langle s_2, a_2, s_2' \rangle \in \rightarrow_2$ satisfying $s_1' \,\mathcal{R}_S\, s_2'$ and $a_1 \,\mathcal{R}_\Sigma\, a_2$.

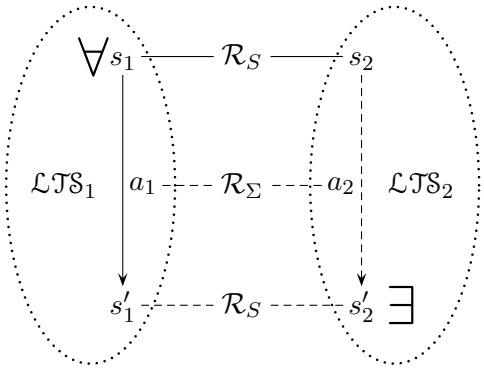


Figure 2.4: Simulation relation.

Figure 2.4 illustrates the intuitive idea of Definition 2.4.15. It is worth noticing that our definition of simulation concerns not only the states but also the actions of two labeled transition systems. The standard definition of simulation/bisimulation [Mil71, JP93, San04, San07] is thus a particular case in which two action sets Σ_1 and Σ_2 are the same and $\mathcal{R}_\Sigma$ is the identity relation on Σ_1. Our definition slightly differs from that of a morphism between labeled transition systems given by Sassone *et al.* [SNW96] in which $\mathcal{R}_\Sigma$ is a partial function, i.e. $\mathsf{Dom}(\mathcal{R}_\Sigma) \subseteq \Sigma_1$, hence some actions of a labeled transition system could be simulated by the ε-action of another labeled transition system, i.e. the other system does not move.

Example 2.4.16. Consider the FIFO channel in Example 2.4.3 $\mathcal{FF} = (M^*, \{!a, !b, ?a, ?b\}, \rightarrow_{\mathcal{FF}}, a)$, where $M = \{a, b\}$ and the synchronized product of three counters in Example 2.4.13 $\mathcal{SP} = (\mathbb{N}^3, \{+, +', -\}, \rightarrow_{\mathcal{SP}}, \langle 0, 1, 1 \rangle)$. Let us define:

- $\mathcal{R}_S = \{\langle v, w\rangle \in \mathbb{N}^3 \times M^* \text{ such that } |w| = v_3\}$, and
- $\mathcal{R}_\Sigma = \{\langle +, !a\rangle, \langle +', !b\rangle, \langle -, ?a\rangle, \langle -, ?b\rangle\}$.

It is easy to see that $(\mathcal{R}_S, \mathcal{R}_\Sigma)$ is a simulation relation from $\mathcal{SP}$ to $\mathcal{FF}$, and we say that $\mathcal{FF}$ simulates $\mathcal{SP}$. More precisely, a state $\langle i, j, k\rangle$ in $\mathcal{SP}$ is simulated by a word $w = abab\ldots$ where $|w| = k$.

Now let us restrict the state set of $\mathcal{FF}$ to the set of words in which the first message, if exists, is a and two consecutive messages are always different, and called it $S'_{\mathcal{FF}}$. It means that $S'_{\mathcal{FF}} = \{\varepsilon, a, ab, aba, abab, \ldots\}$. Let $\mathcal{R}'_S$ be the restriction of $\mathcal{R}_S$ to the new domain $S'_{\mathcal{FF}}$. Then $(\mathcal{R}'_S, \mathcal{R}_\Sigma)$ is still a simulation relation from $\mathcal{SP}$ to the new labeled transition system $\mathcal{FF}'$ and conversely, $(\mathcal{R}'^{-1}_S, \mathcal{R}^{-1}_\Sigma)$ is also a simulation relation from $\mathcal{FF}'$ to $\mathcal{SP}$.

Remark: When a labeled transition $\mathcal{LTS}_1$ is simulated by another one $\mathcal{LTS}_2$ w.r.t. to some simulation relation $(\mathcal{R}_S, \mathcal{R}_\Sigma)$, by Definition 2.4.15, we also have that $\mathcal{LTS}_1$ is simulated by $\mathcal{LTS}_2$ w.r.t. every relation $(\mathcal{R}_S, \mathcal{R}'_\Sigma)$ satisfying $\mathcal{R}'_\Sigma \supseteq \mathcal{R}_\Sigma$. And particularly, one can choose such a relation $\mathcal{R}'_\Sigma$ so that it is the maximal one w.r.t. the inclusion order, i.e. $\mathcal{R}'_\Sigma = \Sigma_1 \times \Sigma_2$.

Such a simulation relation $(\mathcal{R}_S, \Sigma_1 \times \Sigma_2)$ intuitively induces to simulations between transition systems without labeling actions in which one is interested in only systems' states. However, the smaller the relation $\mathcal{R}_\Sigma$ is, the more information about correspondence between systems' action that one can figure out. This fact is also true for bisimulations defined as follow:

Definition 2.4.17 (Bisimulation)**.** Two labeled transition systems $\mathcal{LTS}_1$ and $\mathcal{LTS}_2$ are *bisimilar*, or *in bisimulation*, if there exists a simulation relation $(\mathcal{R}_S, \mathcal{R}_\Sigma)$ from $\mathcal{LTS}_1$ to $\mathcal{LTS}_2$ such that $(\mathcal{R}^{-1}_S, \mathcal{R}^{-1}_\Sigma)$ is also a simulation relation from $\mathcal{LTS}_2$ to $\mathcal{LTS}_1$.

Notation 2.4.18. We denote the fact that $\mathcal{LTS}_1$ and $\mathcal{LTS}_2$ are bisimilar by $\mathcal{LTS}_1 \sim \mathcal{LTS}_2$.

Example 2.4.19. Given the counter in Example 2.4.2 $\mathcal{CT} = (\mathbb{N}, \{+, -\}, \rightarrow_{\mathcal{CT}}, 1)$ and the synchronized product of the three counters in Example 2.4.13 $\mathcal{SP} = (\mathbb{N}^3, \{+_{\mathcal{SP}}, +'_{\mathcal{SP}}, -_{\mathcal{SP}}\}, \rightarrow_{\mathcal{SP}}, \langle 0, 1, 1\rangle)$. These two labeled transition systems are in bisimulation w.r.t. the bisimulation relation $(\mathcal{R}_S, \mathcal{R}_\Sigma)$, where

- $\mathcal{R}_S = \{\langle v, k\rangle \in \mathbb{N}^3 \times \mathbb{N} \text{ such that } k = v_3\}$, and
- $\mathcal{R}_\Sigma = \{\langle +_{\mathcal{SP}}, +\rangle, \langle +'_{\mathcal{SP}}, +\rangle, \langle -_{\mathcal{SP}}, -\rangle\}$.

Intuitively, by grouping vertices corresponding to states $\langle 0, 1, k\rangle$ and $\langle 1, 0, k\rangle$ for every $k \in \mathbb{N}$ in the Figure 2.3, we directly obtains the graph representing the counter in Figure 2.1. The actions $+_{\mathcal{SP}}$, $+'_{\mathcal{SP}}$ are identical and both correspond to the same action '+' in the counter.

2.5 Petri nets

A *net* [NPW80, Rei85] is a triple $(P, T, \mathcal{F})$ where P is a set of *places*, T is a set of *transitions*, and $\mathcal{F} \subseteq (P \times T) \cup (T \times P)$ is its *flow relation* such that P and T are pairwise disjoint. The *preset* (*postset*) of a node $n \in P \cup T$, denoted by ${}^\bullet n$ (resp. $n^\bullet$) is the set of nodes $\{n' \in P \cup T \,/\, \langle n', n\rangle \in \mathcal{F}\}$ (resp. $\{n' \in P \cup T \,/\, \langle n, n'\rangle \in \mathcal{F}\}$).

A *multiset* over a set X is a function $\mu : X \rightarrow \mathbb{N}$. Notice that any subset of X may be viewed as a multiset over X. We denote $x \in \mu$ if $\mu(x) \geq 1$, and for two multisets μ, μ' over X we write $\mu \leq \mu'$ if $\mu(x) \leq \mu'(x), \forall x \in X$. The *sum* of two multisets μ and

μ' over X, denoted by $\mu + \mu'$, is given by $(\mu + \mu')(x) = \mu(x) + \mu'(x)$; and when $\mu' \leq \mu$, the *difference*, denoted by $\mu - \mu'$, is given by $(\mu - \mu')(x) = \mu(x) - \mu'(x)$.

A *marking* of a net $\mathcal{N} = (P, T, \mathcal{F})$ is simply a multiset over P.

Definition 2.5.1 (Petri net)**.** A *Petri net* is a quadruple $(P, T, \mathcal{F}, m_0)$ where $\mathcal{N} = (P, T, \mathcal{F})$ is a net and m_0 is a marking of $\mathcal{N}$. m_0 is called the *initial marking*.

Figure 2.5 illustrates a Petri net in which we use the standard rules about drawing nets: places are represented as circles, transitions as solid bars, flow relation $\mathcal{F}$ by arcs, and markings are shown by placing tokens within circles.

$$
\begin{aligned}
P &= \{p_1, p_2, p_3\} \\
T &= \{t_1, t_2\} \\
\mathcal{F} &= \{\langle p_1, t1\rangle, \langle t_1, p_2\rangle, \langle t_1, p_3\rangle\} \\
&\quad \cup \{\langle p_2, t_2\rangle, \langle t_2, p_1\rangle, \langle t_1, p_3\rangle\} \\
&\quad \cup \{\langle p_3, t_3\rangle\} \\
m_0 &= \{\langle p_1, 0\rangle, \langle p_2, 1\rangle, \langle p_3, 1\rangle\}
\end{aligned}
$$

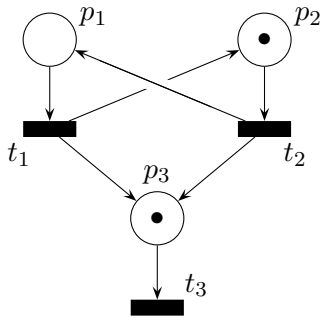


Figure 2.5: A Petri net

The semantic of a Petri net $(P, T, \mathcal{F}, m_0)$, is given by the one of its corresponding labeled transition system $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ where S is the set of all possible marking of $(P, T, \mathcal{F})$, $\Sigma = T$, $s^0 = m_0$ and for all marking m, m' and transition t, we have $m \xrightarrow{t} m'$ iff ${}^\bullet t \leq m$ and $m' = m - {}^\bullet t + t^\bullet$. Intuitively, firing a transition t from a marking m is removing a token in every place in ${}^\bullet t$ and then adding a token to every place in $t^\bullet$.

More interestingly, a place p initialized with k tokens may be seen as a particular Petri net $(\{p\}, \{t_+, t_-\}, \{\langle p, t_-\rangle, \langle t_+, p\rangle\}, \{\langle p, k\rangle\})$. This Petri net corresponds to a counter $\mathcal{CT} = (\mathbb{N}, \{+, -\}, \rightarrow_{\mathcal{CT}}, k)$. An arbitrary Petri net, on its turn, corresponds to a synchronized product of counters. For instance, let us consider the synchronized product $\mathcal{SP}$ of three counters in Example 2.4.13 and the Petri net $(P, T, \mathcal{F}, m_0)$ in Figure 2.5, places in P are mapped to the component counters of $\mathcal{SP}$, and transitions in T are mapped to the synchronization constraint $\{\langle -, +, +\rangle, \langle +, -, +\rangle, \langle \varepsilon, \varepsilon, -\rangle\}$ of $\mathcal{SP}$.

Definition 2.5.2. A *k-bounded* Petri net is a Petri net $(P, T, \mathcal{F}, m_0)$ in which every reachable marking m must satisfy that $m(p) \leq k$ for all $p \in P$.

A particular case of parameter k in Definition 2.5.2 is when $k = 1$. In this case, such a Petri net is called 1-*safe* or a *safe* Petri net. A Petri net is used to describe a wide range of systems. In [McM95a], McMillan has proposed a verification technique on Petri nets which is based on the concept of net unfolding [NPW80]. The unfolding of a Petri net is a net with simpler structure, called *(labeled) occurrence net*. However, for technical reasons, algorithms for constructing labeled occurrence nets of a Petri net $(P, T, \mathcal{F}, m_0)$ requires two following restrictions:

- there is no transition with empty preset, i.e. for all $t \in T$, ${}^\bullet t \neq \emptyset$, and
- the synchronization is finite, i.e. for every transition $t \in T$, ${}^\bullet t$ and $t^\bullet$ are finite sets.

It is recommended to have a look at [Sta89, McM95a, ERV96] for more details.

Chapter 3

Modeling concurrent systems by labeled event structures

Contents

Event structures [NPW80, Win82], abstract away from the cyclic structure of the process and consider only events, assumed to be the *atomic computational steps*, and the cause/effect relationships between them. Thus, we can classify event structures as *behavioral, branching* and *noninterleaving* models. In Section 3.1, we introduce prime event structures as well as associated classical notations. A brief comparison with general event structures is also given in this section.

In order to model a system's behavior, we are rather interested in labeled event structures. It allows us to represent states and different operations of a system by means of labeling functions. As a consequence, one will find in Section 3.2 some strict correspondence between such labeled event structures and labeled transition systems which model a same system. The advantage of the first model, in comparison with the second one, is that it can be directly used for verification (see Chapter 4). And moreover, because of the partial-order inside, labeled event structures are generally compact, and this fact makes verifications more efficient.

Concurrent labeled event structures for well-known systems, such as counters and FIFO channels, will be given in Section 3.3. They provide a set of examples of how

to exploit independence between a system's actions. This independence brings forth the concurrency of labeled event structures afterward. Then, we also define a class of synchronized products of labeled event structures that conforms to the synchronization idea on products of labeled transition systems given in Chapter 2. Such a synchronized product of labeled event structures inherits well the concurrency of its components.

3.1 Prime event structures

Definition 3.1.1 (Prime Event Structure)**.** A *prime event structure* is a triple $\mathcal{E} = (E, \leq, \#)$ where E is a set of *events*, $\leq \subseteq E \times E$ is a partial order on E, the *causality* relation, and $\# \subseteq E \times E$ is a symmetric, irreflexive relation, the *conflict* relation, satisfying:

- *finitary*: $\forall e \in E$, $\geq(e)$ is finite, and
- *conflict-inheritance*: $\forall e, e', e'' \in E$, $e \# e'$ and $e' \leq e''$ implies that $e \# e''$.

Intuitively, events (strictly speaking event *occurrences*) in a prime event structure are ordered w.r.t. the causality relation. This partial order means that an event must be preceded by, or occur after, some other one; and moreover, by only a finite number of events. It is worth noticing that this property of finitary in Definition 3.1.1 is fundamental from a computational point of view. The reason for this is that we assume that only finitely many events can occur in a finite amount of time. And therefore, only events with finitely many causes can occur.

Hence, a prime event structure is simply a poset $(E, \leq)$ equipped with a conflict relation $\#$ which means that two events, for example e and e', can not both occur. And naturally, events afterward are thus in conflict as the conflict-inheritance property states.

Notice that an event can not be in conflict with itself due to the irreflexivity of conflict relation. This condition is sometimes called *consistent* [Win82] or *non-self-conflict* property in similar concurrent structures, e.g. occurrence nets [McM95a]. Two events are *concurrent* if they are neither causal nor in conflict. This *concurrent relation* is thus a symmetric and irreflexive binary relation on E.

Notation 3.1.2. We denote $\|$ the concurrent relation $((E \times E) \setminus (\leq \cup \geq)) \setminus \#$.

We extend the relations of conflict and of concurrency to subsets of E, and respectively denote by $\#^s, \|^s$, as follows:

- $X \#^s Y$ iff $\# \cap (X \times Y) \neq \emptyset$, and
- $X \|^s Y$ iff $(X \times Y) \subseteq \|$.

In words, two subsets X, Y are conflict if there exists a pair of event $x \in X$ and $y \in Y$ which are in conflict; and these subsets are concurrent if all such pairs of events are concurrent. Given an event e, recall that e can stand for the singleton $\{e\}$, hence $e \|^s X$ means that e is concurrent with every event in X. An event set $X \subseteq E$ is called concurrent if $e \|^s (X \setminus e)$ for all events $e \in X$.

Thanks to Lemma 2.3.7, it follows from the finitary property that the binary relation $\lessdot$ corresponding to the poset $(E, \leq)$ (see Notation 2.3.6 on page 14) satisfies that $\lessdot^* = \leq$. Moreover, since $\lessdot$ is the intersection of all binary relations whose transitive closures are equal to $\leq$, $\lessdot$ is the minimal one w.r.t. the inclusion order.

As in graph theory, for two different events $e, f \in E$, if $e < f$ then we say that e is a *predecessor* of f, and reversely, that f is a *successor* of e. More precisely, e is a *direct predecessor* of f and f is a *direct successor* of e if there does not exists another event g such that $e < g$ and $g < f$. As shown in the proof of Lemma 2.3.7, the relation $\lessdot$

formally represents this direct predecessor/successor relation, i.e. $e \lessdot f$. Hence, the set of direct predecessors (successors) of an event e is the set $\gtrdot(e)$ ($\lessdot(e)$ respectively). In this work, we briefly call $\lessdot$ the *predecessor relation*.

3.1.1 Example and graphical representation

Example 3.1.3. The prime event structure $\mathcal{E} = (E, \leq, \#)$ where:

- $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$,
- $\leq = \mathcal{I}_E \cup (\{e_2\} \times \{e_3, e_4, e_5, e_6\}) \cup (\{e_4\} \times \{e_5, e_6\})$, and
- $\# = \emptyset$,

has no conflict. Its events are pairwise in causal or concurrent.

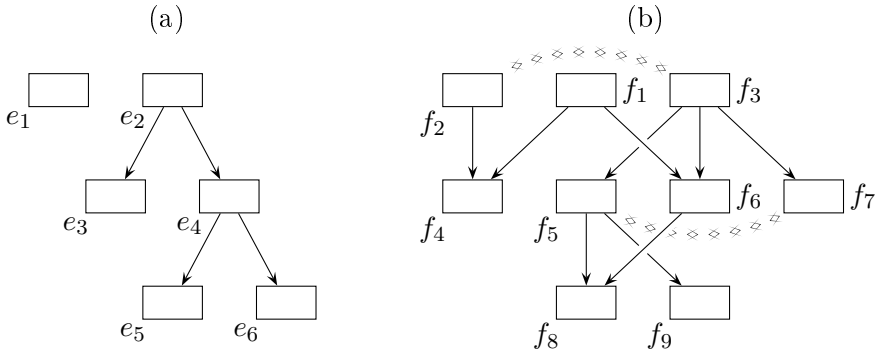


Figure 3.1: Graphical representation of prime event structures

Figure 3.1.a illustrates the prime event structure in Example 3.1.3. We adopt the standard rules about drawing occurrence nets. Events are drawn with boxes, and causal relation is the the transitive reflexive closure of the relation depicted by the oriented arcs. In other words, boxes and arcs in this figure correspond to vertices and edges of a DAG $(E, \lessdot)$ which give rise to the poset $(E, \leq)$.

For the purpose of adapting other works on the unfolding technique, causal events are in top-down direction w.r.t. causal relation.

The conflict relation is represented by arc drawn like: $f_2 \cdots f_3$, as we can see in another example of prime event structure $\mathcal{E}' = (E', \leq', \#')$ which is shown in Figure 3.1.b. It is worth noticing that our graphical representation depicts only a subset $\#''$ of the conflict relation $\#$. For clarity, this relation $\#''$ should be as small as possible, however, in addition of the conflict-inheritance, $\#''$ is enough for computing $\#$. Formally, $\#'' = \{\langle e, f\rangle \in \#' \,/\, {>}(e)\,\overline{\#'^s}\,{>}(f)\}$. In this example, it follows from $\#'' = \{\langle f_2, f_3\rangle, \langle f_3, f_2\rangle, \langle f_5, f_7\rangle, \langle f_7, f_5\rangle\}$ that the conflict relation $\#'$ is given by:

$$
\begin{aligned}
\#' &= \big(\leq'(f_2) \times \leq'(f_3)\big) \cup \big(\leq'(f_3) \times \leq'(f_2)\big) \\
&\quad \cup \big(\leq'(f_5) \times \leq'(f_7)\big) \cup \big(\leq'(f_7) \times \leq'(f_5)\big) \\
&= (\{f_2, f_4\} \times \{f_3, f_5, f_6, f_7, f_8, f_9\}) \cup (\{f_3, f_5, f_6, f_7, f_8, f_9\} \times \{f_2, f_4\}) \\
&\quad \cup (\{f_5, f_8, f_9\} \times \{f_7\}) \cup (\{f_7\} \times \{f_5, f_8, f_9\})
\end{aligned}
$$

3.1.2 Configurations and extensions

A subset of E is called *conflict free* if it does not contain events that are in conflict.

Definition 3.1.4 (Configuration)**.** Let $\mathcal{E} = (E, \leq, \#)$ be an prime event structure. A *configuration* of $\mathcal{E}$ is any finite subset C of E such that C is downward closed w.r.t. $(E, \leq)$ and conflict free.

For all event $e \in E$, the downward-closed set $\geq(e)$ is a conflict free set due to the conflict-inheritance and irreflexivity of the conflict relation in Definition 3.1.1. Hence, $\geq(e)$, finite by definition, is also a configuration and is called the *local configuration* of e.

Notation 3.1.5. Let $\mathcal{E} = (E, \leq, \#)$ be an prime event structure. We denote by $\mathcal{C}_\mathcal{E}$ and $\mathcal{C}^l_\mathcal{E}$ respectively the set of configurations and the set of local configurations of $\mathcal{E}$.

Definition 3.1.6 (Extension)**.** Let C be a configuration of a prime event structure $\mathcal{E} = (E, \leq, \#)$. An event $e \in E$ is an *extension* of C, denoted by, $C \vdash e$ if $e \notin C$ and $C \cup \{e\}$ is a configuration of $\mathcal{E}$.

The extension of a configuration by events directly gives rise to a notion of extension by a set of events. We call a subset $X \subseteq E$ an *extension set* of configuration C, and write $C \Vdash X$, if X and C are disjoint and $C \cup X$ is also a configuration.

Remark that our set extension $\Vdash$ on prime event structures slightly differs from the extension notion $\Vdash'$ on *local event structures* [HKT96] in which an extension set X must be a concurrent set, and it deduces that $X = \mathsf{Min}_{(E,\leq)}(X)$. The triple $(E, \mathcal{C}_\mathcal{E}, \Vdash')$ is thus a local event structure.

Lemma 3.1.7. *Given a configuration C of a prime event structure $\mathcal{E} = (E, \leq, \#)$. For all non-empty extension set $X \subseteq E$ of C we have:*

1. *$\exists e \in X$ such that $C \vdash e$ and $(C \cup e) \Vdash (X \setminus e)$, and*
2. *if X is finite, then $\exists f \in X$ such that $C \Vdash (X \setminus f)$ and $((C \cup X) \setminus f) \vdash f$.*

Proof. First item: Let g be any event in X. Due to the finitary property of $\mathcal{E}$ in Definition 3.1.1, $\geq(g)$ is finite, and so does $\geq(g) \setminus C$. There exists a minimal event e of $(\geq(g) \setminus C)$ w.r.t. $(E, \leq)$. Hence $C \cup e$ is downward closed w.r.t. $(E, \leq)$. Since $C \cup X$ is conflict free, and $(C \cup e) \subseteq (C \cup X)$, $C \cup e$ is thus a configuration. We have $C \vdash e$ and $(C \cup e) \Vdash (X \setminus e)$ by definition.

The second item can be proved by the same manner as the first one while choosing f as a maximal event w.r.t. $(E, \leq)$ of the finite set X. □

Corollary 3.1.8. *For every extension set X of a configuration C, i.e. $C \Vdash X$, for every linearisation $e_1, e_2, e_3, \ldots$ of X w.r.t. $(E, \leq)$ we have $C \vdash e_1, (C \cup \{e_1\}) \vdash e_2, (C \cup \{e_1, e_2\}) \vdash e_3, \ldots$*

The decidability of some verification problems on prime event structures (see later in Section 4.2) requires that prime event structures satisfy the following property.

Definition 3.1.9. A prime event structure $\mathcal{E} = (E, \leq, \#)$ is *finitely-branching* if every configuration $C \in \mathcal{C}_\mathcal{E}$ has a finite number of extension events, i.e. $\{e \in E \,/\, C \vdash e\}$ is finite.

3.1.3 Sub-structures

Definition 3.1.10. Let $\mathcal{E} = (E, \leq, \#)$ be a prime event structure, $F \subseteq E$ be a set of events. The *restriction* of $\mathcal{E}$ onto F, denoted by $\mathcal{E}|_F$, is the triple $(F, \leq|_F, \#|_F)$ where $\leq|_F, \#|_F$ are respectively the restrictions of $\leq, \#$ onto $(F \times F)$.

Lemma 3.1.11. *$\mathcal{E}|_F$ is a prime event structure for every subset $F \subseteq E$.*

Proof. $(F, \leq|_F)$ is a poset by Lemma 2.3.3 on page 13, let us denote $\leq|_F$ by $\leq'$. The symmetry and irreflexivity are preserved in binary relation $\#|_F$. Moreover, $\geq'(f) \subseteq \geq(f)$ is finite for all event $f \in F$, and conflict-inheritance is also guaranteed with $\leq|_F$ and $\#|_F$. $\mathcal{E}|_F$ is thus a prime event structure. □

Definition 3.1.12 (Prefixes)**.** Let $\mathcal{E} = (E, \leq, \#)$ be a prime event structure. Given a downward-closed set $F \subseteq E$ w.r.t. the causality $\leq$, the restriction of $\mathcal{E}$ onto F, i.e. $\mathcal{E}|_F$, is called the *F-prefix* of $\mathcal{E}$.

In practice, one only works on some finite prefix $\mathcal{E}|_F$ of $\mathcal{E}$, i.e. F is finite (see Chapter 5). The *prefix* $\mathcal{E}|_F$ gives not only a downward-closed set of events but also the causality and the conflict relation between these events. The notion of sub-structures, as a consequence, the notion of prefix, could be generalized to event structures of which event sets may be disjoint. It bases on so called *isomorphism* which is defined as follows:

Definition 3.1.13 (Isomorphism)**.** We say that two event structures $(E, \leq, \#)$ and $(E', \leq', \#')$ are *isomorphic* if there exists a bijection $\mathcal{B}$ between E and E' such that, for all events $e, f \in E$,

- $e \leq f$ iff $\mathcal{B}(e) \leq \mathcal{B}(f)$, and
- $e \# f$ iff $\mathcal{B}(e) \#' \mathcal{B}(f)$.

Then, an event structure $\mathcal{E}'$ is also called a prefix of another one $\mathcal{E} = (E, \leq, \#)$, w.r.t. isomorphism, if $\mathcal{E}'$ is isomorphic with some prefix $\mathcal{E}|_F$ of $\mathcal{E}$. In this case, we say that $\mathcal{E}'$ is smaller than or equal to $\mathcal{E}$ w.r.t. isomorphism. It intuitively defines a partial-order on the set of all event structures, called the *prefix-order*. In this poset, the event structure without event, i.e. $(\emptyset, \emptyset, \emptyset)$, is the minimal one.

Another particular kind of sub-structures concerning $\mathcal{E}$ is its *suffix*. A suffix is based on some configuration $C \in \mathcal{C}_\mathcal{E}$. Recall that C is intuitively a set of events, certainly downward-closed and conflict-free, which can occur together. One is interested in events that can occur afterward, or together with events in C. The set of such events is the set $((E \setminus C) \setminus \#(C))$ which may be determined in another way by the union set of all extension sets of C in $\mathcal{E}$.

Definition 3.1.14 (Suffixes)**.** Given a configuration C of a prime event structure $\mathcal{E} = (E, \leq, \#)$, the restriction of $\mathcal{E}$ onto $((E \setminus C) \setminus \#(C))$ is called the *C-suffix* of $\mathcal{E}$.

As detailed in Section 3.3, when aiming at modeling a system by certain event structure $\mathcal{E}$, each configuration C in $\mathcal{E}$ corresponds somehow to a system's state. The C-suffix of $\mathcal{E}$ intuitively models another system that is different from the first one only on its initial state. Therefore, in a theoretical view, suffixes of an event structures allow us to model a family of systems. As a direct consequence of Lemma 3.1.11, suffixes as well as prefixes of a prime event structure are prime event structures.

Example 3.1.15. Figure 3.2 depicts the $\{f_1, f_3\}$-suffix of the prime event structure $\mathcal{E} = (E, \leq, \#)$ shown in Figure 3.1.b. The event set of this suffix is computed as follows:

$$\begin{aligned} E' &= ((E \setminus \{f_1, f_3\}) \setminus \#(\{f_1, f_3\})) \\ &= ((E \setminus \{f_1, f_3\}) \setminus \{f_2, f_4\}) \\ &= \{f_5, f_6, f_7, f_8, f_9\} \end{aligned}$$

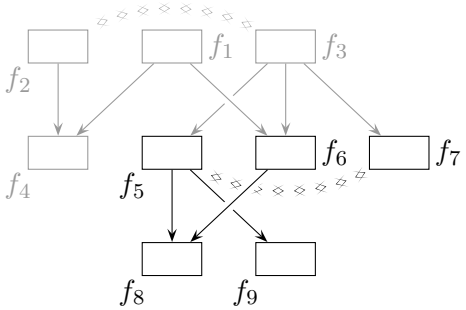


Figure 3.2: The $\{f_1, f_3\}$-suffix of $\mathcal{E}$ given in Figure 3.1.b.

Lemma 3.1.16. *Let $\mathcal{E} = (E, \leq, \#)$ be a prime event structure, and C be a configuration in $\mathcal{E}$. X is an extension set of C, i.e. $C \Vdash X$, iff X is a configuration in the C-suffix of $\mathcal{E}$.*

Proof. Let S denote the event set of the C-suffix of $\mathcal{E}$, i.e. $S = (E \setminus C) \setminus \#(C)$. By definition of event set, $X \cap \#(C) = \emptyset$, and consequently, X is thus a subset of S.

($\Rightarrow$) Since $(C \cup X)$ is downward closed w.r.t. $(E, \leq)$, X is downward closed w.r.t. $(E \setminus C, \leq|_{(E \setminus C)})$. Moreover, it follows from the conflict-freeness of $(C \cup X)$ that X is also conflict-free w.r.t. both $\#$ and $\#|_S$. Therefore, X is a configuration in $\mathcal{E}|_S$.

($\Leftarrow$) It follows from $X \cap \#(C) = \emptyset$ that $(C \cup X)$ is thus a conflict-free set w.r.t. $\#$. Let e be any event in $\geq(C \cup X)$, thanks to the conflict-freeness of $(C \cup X)$, e is not in $\#(C)$, and consequently, e is either in C or in S. Notice that $\geq(C \cup X) = \geq(C) \cup \geq(X)$, if $e \in S$ then e must be in $\geq(X) \cap S$. Since X is downward-closed w.r.t. $\leq|_S$, we have $e \in (\geq(X) \cap S) = \geq|_S(X) = X$. Hence, we always obtain either $e \in \geq(C) = C$ or $e \in X$. As a consequence, $\geq(C \cup X) = (C \cup X)$ that means $(C \cup X)$ is downward-closed w.r.t. the causality $\leq$. Therefore, X is an extension set of C.

□

Corollary 3.1.17. *Let C be a configuration of a prime event structure $\mathcal{E} = (E, \leq, \#)$.*

- *if $\mathcal{E}$ is finitely-branching then the C-suffix of $\mathcal{E}$ is finitely-branching; and*
- *if every event in E is not in conflict with C, i.e. $\#(C) = \emptyset$, and the C-suffix of $\mathcal{E}$ is finitely-branching then $\mathcal{E}$ is finitely-branching.*

Proof. Let S denote the event set of the C-suffix of $\mathcal{E}$, i.e. $S = (E \setminus C) \setminus \#(C)$. Thanks to Lemma 3.1.16, every configuration X in the C-prefix gives rise to a configuration $(C \cup X)$ in $\mathcal{E}$. The left-to-right implication is obvious.

For the right-to-left implication, let C' be any configuration in $\mathcal{E}$ and let $X = C' \setminus C$. Since $\#(C) = \emptyset$, S and C are disjoint sets and $E = C \cup S$. The event set X, maybe empty, is thus an extension set of C and is a configuration in the C-suffix $\mathcal{E}|_S$ due to Lemma 3.1.16. An extension of C' in $\mathcal{E}$ must be either an event in C or, otherwise, an extension of $C \cup X$. Once again, thanks to Lemma 3.1.16, in the second case, such an extension corresponds to another extension of X in $\mathcal{E}|_S$. Since C is finite and X has finitely many extensions in $\mathcal{E}|_S$, C' has a finite number of extensions in $\mathcal{E}$ too. Therefore, $\mathcal{E}$ is finitely-branching. □

3.1.4 Prime vs general event structures

Prime event structure is a subclass of event structure [NPW80, Win82] which is generally defined by a couple $(E, \mathcal{C})$ where E is a set of events, $\mathcal{C}$ is a family of sets over event sets E, or set of configurations. There is an equivalence between this definition of prime event structures and Definition 3.1.1. However, we are interested in the second one because, for many standard systems, the conflict and causality relations may be naturally defined. Then these relations serve to compute configurations, and not conversely.

Moreover, aiming at constructing event structure for systems, we have no concern in event structures not belonging to this subclass. That means event structures which do not satisfying the following properties of prime event structures:

- *full*: every event is associated to at least one configuration, and
- *causality relation is global*: order between two events, if exists, is not varied in accordance with some configuration which contains these events.

From now on, we will say *event structures* for short, always meaning prime event structures.

3.2 Labeled event structures

Definition 3.2.1 (Labeled event structures)**.** A *labeled event structure* is a tuple $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ where $(E, \leq, \#)$ is an event structure, and

- $\mathcal{L}$, called *label function*, is a function from event set E to some alphabet Σ, and
- $\mathcal{M}$, called *marking function*, is a function from configuration set $\mathcal{C}_\mathcal{E}$ to the power set of some (maybe infinite) set S.

Recall that the co-domain is part of the definition of a function. Although the sets Σ and S are not explicitly given in the tuple $(E, \leq, \#, \mathcal{L}, \mathcal{M})$ representing a labeled event structure, we always denote Σ the co-domain of $\mathcal{L}$, i.e. $\Sigma = \mathsf{Codom}(\mathcal{L})$, and S the base set of the co-domain of $\mathcal{M}$, i.e. $\mathcal{P}(S) = \mathsf{Codom}(\mathcal{M})$. These sets Σ, S are called respectively the *set of actions* and the *set of states*.

A labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ is simply an event structure $(E, \leq, \#)$ equipped with two additional labeling functions in order to model the behavior of a system. Hence, all notations and definitions on its event structure $(E, \leq, \#)$ previously defined in Section 3.1, such as configuration, extension, prefix and suffix, are generalized for the labeled event structure itself. Labeling functions for a sub-structure $\mathcal{E}'$ based on subset event E' are thus its left-restrictions to E', i.e. $\mathcal{L}|_{E'}$ and $\mathcal{M}|_{\mathcal{C}'}$ where $\mathcal{C}' = \mathcal{C}_{\mathcal{E}'}$ is the configuration set of $\mathcal{E}'$.

Definition 3.2.2. A labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ is *deterministic* if the co-domain of the marking function $\mathcal{M}$ contains only singletons of the set of states S, i.e. $\mathsf{Codom}(\mathcal{M}) = \{\{s\} \,/\, s \in S\}$.

Remark: For deterministic labeled event structures, marking functions $\mathcal{M}$ may be simply defined as a function from configuration set $\mathcal{C}_\mathcal{E}$ to the set of states S, and its co-domain is then extended to the power set $\mathcal{P}(S)$.

3.2.1 Semantics of labeled event structures

Labeled event structure is used for modeling behaviors of a system. Intuitively, the label function $\mathcal{L}$ tells which events are occurrences of which system's action, while the marking function $\mathcal{M}$ associate a configuration to some states of the system.

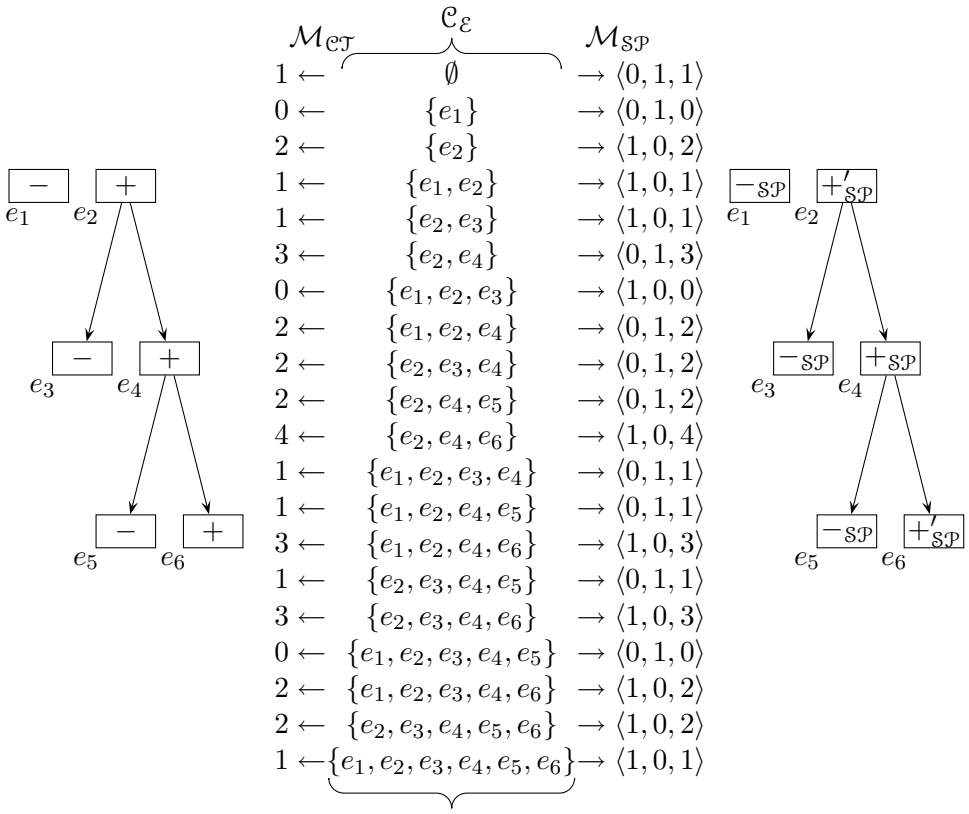


Figure 3.3: Examples of labeled event structures

Example 3.2.3. Let's consider the event structure $(E, \leq, \#)$ depicted in Figure 3.1.b. We can have two different labeled event structures which are deterministic and are respectively defined by two pairs of labeling functions $(\mathcal{L}_{\mathcal{CT}}, \mathcal{M}_{\mathcal{CT}})$ and $(\mathcal{L}_{\mathcal{SP}}, \mathcal{CT}_{\mathcal{SP}})$ where:

- $\mathcal{L}_{\mathcal{CT}} : E \to \{+, -\}$, $\mathcal{M}_{\mathcal{CT}} : \mathcal{C}_\mathcal{E} \to \mathbb{N}$, and
- $\mathcal{L}_{\mathcal{SP}} : E \to \{+_{\mathcal{SP}}, +'_{\mathcal{SP}}, -\}$, $\mathcal{M}_{\mathcal{SP}} : \mathcal{C}_\mathcal{E} \to \mathbb{N}^3$.

These labeled event structures are illustrated in Figure 3.3.

Labeled event structures are graphically represented like event structures (see Section 3.1.1). In addition, the label of an event is shown inside the box corresponding to the event. In Figure 3.3, the marking function $\mathcal{M}_{\mathcal{CT}}$, as well as $\mathcal{M}_{\mathcal{SP}}$, is individually defined for each element (configuration) in its domain $\mathcal{C}_\mathcal{E}$.

Definition 3.2.4. Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a labeled event structure. A *labeled transition system induced by* $\mathcal{E}$, denoted by $\mathcal{LTS}^\mathcal{E}$, is defined as follows:

- the set of states is the base set S of co-domain of the marking function $\mathcal{M}$,
- the set of actions is the co-domain Σ of the label function $\mathcal{L}$,
- the transition relation $\to$ satisfying that for all $s, s' \in S, a \in \Sigma$, $s \xrightarrow{a} s'$ iff there exists a configuration $C \in \mathcal{C}_\mathcal{E}$ and an event $e \in E$ such that $C \vdash e, s \in \mathcal{M}(C), a = \mathcal{L}(e)$ and $s' \in \mathcal{M}(C \cup \{e\})$.

- the initial state $s^0 \in \mathcal{M}(\emptyset)$.

Remark: Due to the last item in Definition 3.2.4, a labeled event structure does not induce an unique labeled transition system.

Moreover, since a configuration corresponds to a set of states, induced labeled transition systems are generally not deterministic. Although of the determinism of a labeled event structure (Definition 3.2.2), its induced labeled transition systems are deterministic only if, from every configuration C, all extensions whose labels are the same, give a same marking. The following lemma is straightforward.

Lemma 3.2.5. *Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a deterministic labeled event structure. If for every configuration $C \in \mathcal{C}_\mathcal{E}$ and for all extensions e, f of C, $\mathcal{L}(e) = \mathcal{L}(f)$ implies $\mathcal{M}(C \cup e) = \mathcal{M}(C \cup f)$; then $\mathcal{E}$ has one and only one deterministic induced labeled transition system.*

Example 3.2.6. The labeled transition system induced by the labeled event structure $(E, \leq, \#, \mathcal{L}_{\mathcal{CT}}, \mathcal{M}_{\mathcal{CT}})$ in Example 3.2.3 is $(\mathbb{N}, \{+, -\}, \rightarrow_{\mathcal{CT}}, 1)$ where

$$\begin{aligned} \rightarrow_{\mathcal{CT}} = &\{\langle 0, +, 1\rangle, \langle 1, +, 2\rangle, \langle 2, +, 3\rangle, \langle 3, +, 4\rangle\} \\ &\cup \{\langle 1, -, 0\rangle, \langle 2, -, 1\rangle, \langle 3, -, 2\rangle, \langle 4, -, 3\rangle\} \end{aligned}$$

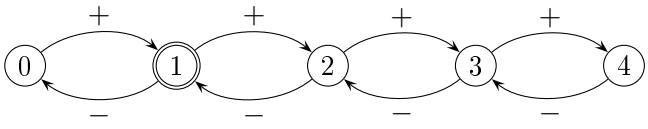


Figure 3.4: Graphical representation of the induced labeled transition system in Example 3.2.6. Notice that it is similar to the one in Figure 2.1 on page 15. But here, there are only 5 reachable states.

We have now an intuitive relation between labeled event structures and labeled transition systems - a classic model: configurations correspond to states, events correspond to actions, extension $\vdash$ and set extension $\Vdash$ respectively correspond to transition relation $\rightarrow$ and execution relation $\longrightarrow\!\!\!\!\!\rightarrow$. In Example 3.2.6, both firing sequences $+-$ and $-+$ are represented by configuration $\{e_1, e_2\}$ in which there is no interleaving due to the the concurrence between these (occurrence) events. Therefore, due to the independence between actions or concurrence between events, labeled event structures give us somehow a way of compactly representing possible firing sequences of a system.

Definition 3.2.7. Two labeled event structures $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ and $\mathcal{E}' = (E', \leq', \#', \mathcal{L}', \mathcal{M}')$ are *isomorphic* and we write $\mathcal{E} \approx \mathcal{E}'$, if $(E, \leq, \#)$, $(E', \leq', \#')$ are isomorphic w.r.t. some bijection $\mathcal{B}$ and

1. $\mathcal{L}(e) = \mathcal{L}'(\mathcal{B}(e))$ for every event $e \in E$, and
2. $\mathcal{M}(C) = \mathcal{M}'(\mathcal{B}(C))$ for every configuration $C \in \mathcal{C}_\mathcal{E}$.

Remark: When the underlying event structures of $\mathcal{E}$ and $\mathcal{E}'$ are isomorphic *w.r.t.* to $\mathcal{B}$ (see Definition 3.1.13), the bijection $\mathcal{B}$ gives rise to a bijection between configuration sets $\mathcal{C}_\mathcal{E}$ and $\mathcal{C}_{\mathcal{E}'}$ in which a configuration $C \in \mathcal{C}_\mathcal{E}$ is associated to the configuration $\mathcal{B}(C) \in \mathcal{C}_{\mathcal{E}'}$.

Corollary 3.2.8. *Let $\mathcal{LTS}^{\mathcal{E}}$ and $\mathcal{LTS}^{\mathcal{E}'}$ be respectively induced labeled transition systems of two labeled event structures $\mathcal{E}$ and $\mathcal{E}'$. If $\mathcal{E}$ and $\mathcal{E}'$ are isomorphic, then $\mathcal{LTS}^{\mathcal{E}}$ and $\mathcal{LTS}^{\mathcal{E}'}$ are bisimilar.*

Proof. Obvious by Definition 2.4.17 and Definition 3.2.4. □

Example 3.2.9. Induced labeled transition systems of the labeled event structures in Example 3.2.3 are bisimilar w.r.t. the bisimulation relation $(\mathcal{R}_S, \mathcal{R}_\Sigma)$ defined in Example 2.4.19.

Moreover, one can find out that the labeled transition system induced by $\mathcal{E} = (E, \leq, \#, \mathcal{L}_{\mathcal{C}_\mathcal{T}}, \mathcal{M}_{\mathcal{C}_\mathcal{T}})$ in Example 3.2.3 is simulated by the counter in Example 2.4.2 by simply observing its semantics depicted in Figure 2.1 and Figure 3.4. The formal reason is that $\mathcal{E}$ is just a prefix of the labeled event structure for this counter latterly defined in Section 3.3.2.

Lemma 3.2.10. *Let C be a configuration of a labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$. Let $(S, \Sigma, \rightarrow, s^0)$, $(S', \Sigma', \rightarrow', s^{0'})$, and $(S'', \Sigma'', \rightarrow'', s^{0''})$ be induced labeled transitions of $\mathcal{E}$, its C-prefix $\mathcal{E}|_{\geq(C)}$, and its C-suffix $\mathcal{E}|_{(E\setminus C\setminus \#(C))}$, respectively. We have:*

- $S = S' \cup S''$,
- $\Sigma = \Sigma' \cup \Sigma''$,
- $\rightarrow = \rightarrow' \cup \rightarrow''$, *and*
- $s^0 \in \mathcal{M}(\emptyset)$, $s^{0'} = \mathcal{M}(\emptyset)$, $s^{0''} \in \mathcal{M}(C)$.

Proof. Obvious due to Lemma 3.1.16 and Definition 3.2.4. □

3.2.2 Properties of labeled event structures

As previously mentioned, we will construct labeled event structures representing systems' behavior. More precisely, suppose that a system is implicitly defined by a labeled transition system where reachable states are represented by configurations' markings and firable actions are represented by extensions. Therefore, if two configurations concerns a same state s, i.e. its markings contain s, such configurations should have extensions in accordance with all firable actions from s.

Definition 3.2.11 (Coherence)**.** A labeled event structure is *coherent* if for all configurations $C, C' \in \mathcal{C}_\mathcal{E}$,

- if $\mathcal{M}(C) \cap \mathcal{M}(C') \neq \emptyset$ then for every $a \in \Sigma = \mathsf{Codom}(\mathcal{L})$ we have

$$\bigcup_{e \in E, \mathcal{L}(e)=a, C \vdash e} \mathcal{M}(C \cup \{e\}) = \bigcup_{e \in E, \mathcal{L}(e)=a, C' \vdash e} \mathcal{M}(C' \cup \{e\})$$

- if $\mathcal{M}(C) = \mathcal{M}(C')$ then for every extension $e \in E$ of C, there exists an extension $e' \in E$ of C' such that $\mathcal{L}(e) = \mathcal{L}(e')$ and $\mathcal{M}(C \cup \{e\}) = \mathcal{M}(C' \cup \{e'\})$.

Figure 3.5 illustrates the first property of the coherence. A simple consequence of coherence is that if the markings of two configuration C and C' are not disjoint sets, they are extended by the same set of labels/actions a in $\mathsf{Codom}(\mathcal{L})$. One can say that the second property which is a particular case of the first one when configurations have a same marking, sounds more reasonable. However, we are in favor of the first one when working on non-deterministic labeled event structures as well as on non-deterministic systems. It offers further some possibility of marking abstraction for labeled event structures.

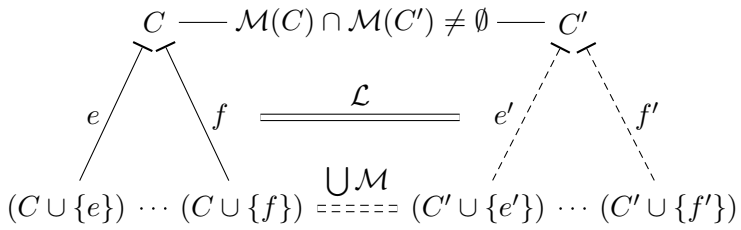


Figure 3.5: Coherence of labeled event structures

Once again, look at the labeled event structure in Example 3.2.6. It represents only some finite executions of its induced labeled transition system. The coherence property is not satisfied in this example because, for instance, configurations $\emptyset$ and $\{e_1, e_2, e_3, e_4, e_5, e_6\}$ have a same marking, however, the first one can be extended while the second one can not. This example explains well that coherent labeled event structures are generally not finite. In fact, a configuration in a labeled event structure represents not only reachable states by means of its marking, but also firing sequences of some induced labeled transition system. Hence, if the induced system has an infinite execution, then the corresponding labeled event structure should be infinite too.

Lemma 3.2.12. *Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a coherent labeled event structure and $\mathcal{LTS}^{\mathcal{E}} = (S, \Sigma, s^0, \rightarrow)$ be an induced labeled transition system of $\mathcal{E}$. For all $s \in S$ and $\sigma \in \Sigma^+$, we have $s^0 \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} s$ iff there exists a non-empty configuration $C \in \mathcal{C}_{\mathcal{E}}$ and a linearisation l of C w.r.t. $(E, \leq)$ such that $\sigma = \mathcal{L}^{\mathrm{W}}(l)$ and $s \in \mathcal{M}(C)$.*

Proof. This lemma is a direct consequence of the following property: $s^0 \stackrel{a_1}{\longrightarrow} s_1, s_1 \stackrel{a_2}{\longrightarrow} s_2, \ldots$ is a path of $\mathcal{LTS}^{\mathcal{E}}$ iff there exists a sequence of events $e_1, e_2, \ldots$ such that $a_1 = \mathcal{L}(e_1), \emptyset \vdash e_1$ and $s_1 \in \mathcal{M}(\{e_1\})$; $a_2 = \mathcal{L}(e_2), \{e_1\} \vdash e_2$ and $s_2 \in \mathcal{M}(\{e_1, e_2\})$; .... This property could be easily proved by induction on the length of the path and using Lemma 3.1.16, Definition 3.2.4 and Lemma 3.2.10. □

Lemma 3.2.12 only states about non-empty firing sequences. However, the empty word ε is of course a firing sequence of $\mathcal{LTS}^{\mathcal{E}}$ where its reachable state is equal to the initial state s^0. Other states s concerning the empty configuration $\emptyset$, i.e. $s \in (\mathcal{M}(\emptyset) \setminus s^0)$ does not mean that s is reachable. The following corollary is straightforward from Lemma 3.2.12.

Corollary 3.2.13. *Given a labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, if $\mathcal{M}(\emptyset)$ is a singleton s^0 then the reachable state set of its unique labeled transition system $\mathcal{LTS}^{\mathcal{E}}$, i.e. $\mathsf{post}^*_{\mathcal{LTS}^{\mathcal{E}}}$, is equal to $\bigcup_{C \in \mathcal{C}_{\mathcal{E}}} \mathcal{M}(C)$.*

As a consequence of the Lemma 3.2.12, one configuration of the labeled event structure may represent several executions of its induced labeled transition system. These executions are simply different interleavings of events of the configuration, or in other words, interleavings of actions' occurrences. The more concurrency between events of the configuration, the more corresponding executions it has. Therefore, in the view of modeling systems by labeled event structures, it is worth noticing that for two labeled event structures whose induced labeled transition systems are the same, the one in which there are more concurrency, seems to be the more compact.

Definition 3.2.14 (Redundancy)**.** A labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ is *redundant* if there exists a configuration $C \in \mathcal{C}_\mathcal{E}$ which has two different extensions $e, e' \in E$ such that $e \# e'$, $\mathcal{L}(e) = \mathcal{L}(e')$ and $(\mathcal{M}(C \cup \{e\}) \cap \mathcal{M}(C \cup \{e'\})) \neq \emptyset$.

Definition 3.2.14 is similar to the one used in labeled occurrence nets [McM95a]. However, non-redundancy in labeled event structures does not give rise to the notion of *unique* labeled event structure for some system. In other words, we can have many non-redundant labeled event structures which model a same system, i.e. correspond to the labeled transition system modeling it. More details on redundancy will be given in Section 6.2.2.

3.3 Modeling concurrent systems

3.3.1 Labeled event trees

Definition 3.3.1. An *event tree* is an event structure $\mathcal{E} = (E, \leq, \#)$ satisfying that all events are pairwise in causality or in conflict, i.e. $(\leq \cup \geq \cup \#) = (E \times E)$.

Corollary 3.3.2. *Let $\mathcal{E} = (E, \leq, \#)$ be a tree,*

1. *for every configuration $C \in \mathcal{C}_\mathcal{E}$, the restriction of $\leq$ onto C, i.e. $\leq|_C$ is a total order, and C has thus one and only one linearisation w.r.t. the causality, and*
2. *every non-empty configuration C is the local configuration of some event $e \in E$ where e is the maximal event in C w.r.t. the causality, i.e. $\{e\} = \mathsf{Max}_\leq(C)$ and $C = \geq(e)$.*

Proof. Since every configuration C is conflict-free, i.e. $\#|_C = \emptyset$, all its events are pairwise in causality. The partial-order $\leq|_C$ is thus a total order on C. It defines the unique linearisation of C w.r.t. $\leq$ by Definition 2.3.4. The first item is proved.

As a consequence, the finite set C with its total order $\leq|_C$ admits a unique maximal event w.r.t. $\leq$. Let us denote this maximal event by e, $\{e\} = \mathsf{Max}_\leq(C)$ if C is not empty. It follows from the downward-closure of C that $\geq(e) \subseteq C$. Because of the uniqueness of e, one has $e' \leq e$ for all $e' \in C$, and consequently, $C \subset \geq(e)$. Therefore, $\geq(e) = C$, and the second item is also proved. □

One can find different ways of defining a tree in other works [Fin87, Fin91, SNW96]. For an intuitive comparison, in our definition, non-empty configurations C (or events whose local configuration is C) correspond to *nodes*, the empty configuration $\emptyset$ corresponds to a particular node, called *root*; and the acyclic property says that there exists one and only one *path* from the root to any node of the tree. The notion of path is represented by the linearisation of events or equally by the extension of configurations (see Corollary 3.1.8) over event structures. Such an acyclic property corresponds to the first item in Corollary 3.3.2.

Figure 3.6 illustrates an event tree in which events are labeled by either '$-$' or '$+$', and there is, in addition, an added root $\emptyset$ representing the empty configuration $\emptyset$. By giving a simple marking function $\mathcal{M} : \mathcal{C}_\mathcal{E} \to \mathbb{N}$ defined as follow:

$$\mathcal{M}(C) = 1 + |\{e \in C \,/\, \mathcal{L}(e) = +\}| - |\{e \in C \,/\, \mathcal{L}(e) = -\}|$$

one can obtain a deterministic labeled event structure $\mathcal{LET}$ whose induced labeled transition system is the same as the one induced from the labeled event structure $\mathcal{E}$ in Example 3.2.3 on page 28. However, $\mathcal{LET}$ is much bigger than $\mathcal{E}$ because $\mathcal{LET}$ has no concurrency between its events while $\mathcal{E}$ does.

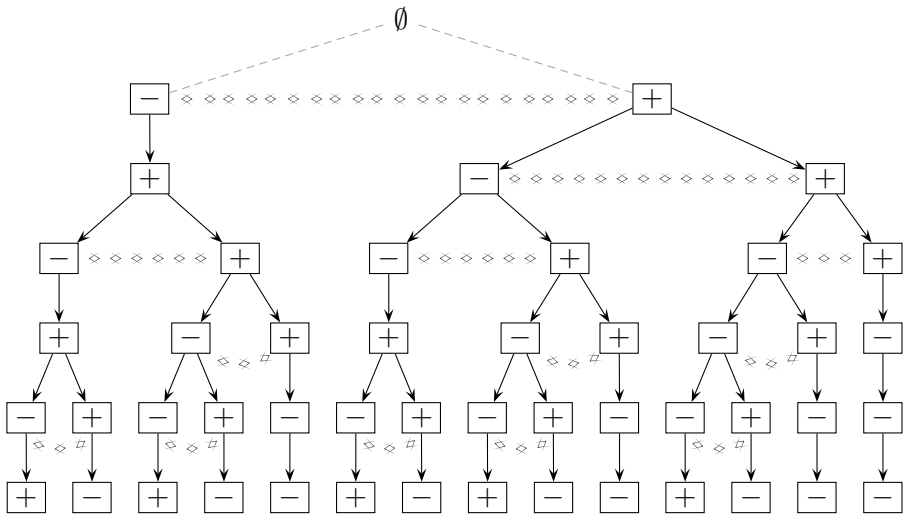


Figure 3.6: Tree with labeled events

Definition 3.3.3 (Labeled event tree)**.** A *labeled event tree* is a coherent and non-redundant labeled event structure $\mathcal{LET} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ where $(E, \leq, \#)$ is an event tree.

Proposition 3.3.4. *Let $\mathcal{LET} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, $\mathcal{LET}' = (E', \leq', \#', \mathcal{L}', \mathcal{M}')$ be two labeled event trees whose induced labeled transition systems are the same. If $\mathcal{LET}, \mathcal{LET}'$ are deterministic then $\mathcal{LET}$ and $\mathcal{LET}'$ are isomorphic.*

Proof. Let $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ denote the induced labeled transition system of both $\mathcal{LET}$ and $\mathcal{LET}'$. Since $\mathcal{LET}$ and $\mathcal{LET}'$ are deterministic, we have $s^0 = \mathcal{M}(\emptyset) = \mathcal{M}'(\emptyset)$.

First, let us define a relation $\mathcal{R}$ between $\mathsf{Min}_{\leq}(E)$ and $\mathsf{Min}_{\leq'}(E')$ as follow: $e \,\mathcal{R}\, e'$ if $\mathcal{L}(e) = \mathcal{L}'(e')$ and $\mathcal{M}(\{e\}) = \mathcal{M}'(\{e'\})$. Notice that $\mathsf{Min}_{\leq}(E)$ are the set of extensions of the empty configuration $\emptyset$ in $\mathcal{LET}$. Let e be any event in $\mathsf{Min}_{\leq}(E)$, and denote $a = \mathcal{L}(e)$, $s = \mathcal{M}(\{e\})$. By definition of induced labeled transition system (Definition 3.2.4), it follows from $\emptyset \vdash e$ and $\mathcal{M}(\{e\}) = s$ that $s^0 \xrightarrow{a} s$. Thanks to Lemma 3.2.12, in $\mathcal{LET}'$, there exist a configuration that is a singleton $\{e'\}$ such that $\mathcal{L}'(e') = a$ and $\mathcal{M}(e')$ is the reachable state s. Therefore, $e \,\mathcal{R}\, e'$ because $\geq'(e') = \{e'\}$ and thus $e' \in \mathsf{Min}_{\leq'}(E')$. $\mathcal{R}$ is thus total. Suppose that $\mathcal{R}$ is not single-valued, there exists two events $e', f' \in \mathsf{Min}_{\leq'}(E')$ such that $\{e', f'\} \subseteq \mathcal{R}(e)$ for some event $e \in \mathsf{Min}_{\leq}(E)$. We have then, $\mathcal{L}'(e') = \mathcal{L}'(f') = \mathcal{L}(e)$, $\mathcal{M}'(\{e'\}) = \mathcal{M}'(\{f'\}) = \mathcal{M}(\{e\})$, and in addition, $e' \#' f'$ due to Corollary 3.3.2. The labeled event tree $\mathcal{LET}'$ is thus redundant by Definition 3.2.14. It contradicts to Definition 3.3.3. Therefore, $\mathcal{R}$ is single-valued, and is a function from $\mathsf{Min}_{\leq}(E)$ to $\mathsf{Min}_{\leq'}(E')$, i.e. $\mathcal{R} : \mathsf{Min}_{\leq}(E) \rightarrow \mathsf{Min}_{\leq'}(E')$. By the same reasoning, we also obtain that $\mathcal{R}^{-1} : \mathsf{Min}_{\leq'}(E') \rightarrow \mathsf{Min}_{\leq}(E)$. Hence, we can conclude that $\mathcal{R}$ is a bijection between $\mathsf{Min}_{\leq}(E)$ and $\mathsf{Min}_{\leq'}(E')$.

Second, let e be any event in $\mathsf{Min}_{\leq}(E)$ and $e' = \mathcal{R}(e)$. Since $\mathcal{LET}, \mathcal{LET}'$ are deterministic labeled event trees, their corresponding suffixes, denoted by $\mathcal{LET}|_{E\setminus\{e\}\setminus\#(e)}$ and $\mathcal{LET}'|_{E'\setminus\{e'\}\setminus\#'(e')}$, are also deterministic labeled event trees. Moreover, because $\mathcal{M}(\{e\}) = \mathcal{M}'(\{e'\})$, it follows from Lemma 3.2.10 on page 30 that these suffixes induce the same induced labeled transition system. Therefore, as previously proved, there exists a bijection $\mathcal{R}'$ between the sets of minimal events in these suffixes $\mathcal{LET}|_{E\setminus\{e\}\setminus\#(e)}$ and $\mathcal{LET}'|_{E'\setminus\{e'\}\setminus\#'(e')}$, such that $\mathcal{L}(f) = \mathcal{L}(f')$ and $\mathcal{M}(\{e, f\}) = \mathcal{M}(\{e', f'\}$ if $\mathcal{R}'(f) = f'$.

It is straightforward that the domain and the co-domain of the bijection $\mathcal{R}'$ are the sets of direct successors of e and e' respectively, i.e. $\lessdot(e)$ and $\lessdot'(e')$.

We can thus defined a relation $\mathcal{B}$ between E and E' which is the union of all bijections $\mathcal{R}, \mathcal{R}', \ldots$ in a constructive way. Since $\mathcal{LET}, \mathcal{LET}'$ are event trees, these bijections' domains as well as its co-domains are pairwise disjoint. Moreover, notice that every configurations of an event tree is the local configuration of some event, i.e. $\mathcal{C}_{\mathcal{LET}} = \mathcal{C}^l_{\mathcal{LET}}$. $\mathcal{B}$ is thus a bijection that satisfies properties in Definition 3.2.7 on page 29. Therefore, $\mathcal{LET}$ and $\mathcal{LET}'$ are isomorphic. □

Proposition 3.3.4 gives rise to the notion of *unique labeled event tree*, up to isomorphism, for labeled transition system.

Definition 3.3.5. Given a labeled transition system $\mathcal{LTS}$, *the deterministic labeled event tree* of $\mathcal{LTS}$ is a deterministic labeled event tree $\mathcal{LET}$ whose induced labeled transition system is $\mathcal{LTS}$.

Although deterministic labeled event tree, a classical structure for system's behavior, is not compact and in general is not the minimal labeled event tree for some labeled transition system, it is simple to construct in practice (see Chapter 5). The reason is that every configuration represents only an execution of the underlying system. And the size of such a labeled event tree becomes huge easily due to interleaving of firable actions. Therefore, we only use labeled event trees for modeling component systems in which it is difficult to find or there exists no concurrency between actions/events (see Chapter 6), for instance, modeling systems' state without queues' content in *communicating finite state machines* [Boc78, BZ83, LI05].

3.3.2 Counters

A *counter* is a well-known datatype with values ranging over the set of natural numbers $\mathbb{N}$, equipped with two operations: '+' and '−' that respectively increases and decreases its value. A counter takes a natural number as its initial value, and may be viewed as a labeled transition system.

Definition 3.3.6 (Counter)**.** A *v-initialized counter*, where $v \in \mathbb{N}$, is a labeled transition system $v\text{-}\mathcal{CT} = (\mathbb{N}, \{+, -\}, \rightarrow_{\mathcal{CT}}, v)$ where the transition relation $\rightarrow_{\mathcal{CT}}$ is the set $(\{\langle n, +, n+1\rangle \,/\, n \in \mathbb{N}\} \cup \{\langle n+1, -, n\rangle \,/\, n \in \mathbb{N}\})$.

By definition, counters are thus deterministic. Example 2.4.2 on page 15 shows the semantics of the 1-initialized counters. In the following, we aim at defining concurrent labeled event structures dedicated to behaviors of such counters. We first restrict to the ones modeling the 0-initialized counter.

Definition 3.3.7 (k-causality event structures)**.** Let k be a natural number, a *k-causality event structure* is an event structure $\mathcal{E} = (E, \leq, \#)$ where $\# = \emptyset$ and $\leq$ satisfies:

1. for all $e \in \mathsf{Min}_{\leq}(E)$, $\lessdot(e) \neq \emptyset$;
2. for all $e \in E$, if $\lessdot(e) \neq \emptyset$ then $|\lessdot(e)| = k + 1$ and $|\{e' \in \lessdot(e) \,/\, \lessdot(e') = \emptyset\}| = 1$;
3. for all $e \in (E \setminus \mathsf{Min}_{\leq}(E))$, $\gtrdot(e)$ is a singleton; and
4. $|\mathsf{Min}_{\leq}(E)| = k$ if $k > 0$ and $|\mathsf{Min}_{\leq}(E)| = \infty$ if $k = 0$.

Recall that $\lessdot$ is the predecessor relation and is the minimal relation w.r.t. the inclusion order such that $\lessdot^* = \leq$. For an event e, $\lessdot(e) = \emptyset$ means that e has no (direct) successors, and is called a *leaf* (as in graph theory). A minimal event (w.r.t. causality)

is not a leaf due to the first item. While the second item intuitively means that if an event e is not a leaf, then it has exactly $k+1$ direct successors, formally defined by the set $\lessdot(e)$, and only one of them is a leaf. The first item says that every event e has at most one direct predecessor, this predecessor is $\gtrdot(e)$ if exists. As a consequence, one can find out that the restriction of $\leq$ onto the local configuration of e, i.e. $\leq|_{\geq(e)}$, is a total order. Hence, a k-causality event structure $\mathcal{E}$ is intuitively a set of disjoint event trees without conflict relation. The roots of such trees correspond 1-to-1 to the minimal events in $\mathsf{Min}_{\leq}(E)$. The last item distinguishes the particular causality event structure where k is zero, and will be explained lately in Section 3.3.2.

Moreover, given any event e which is not a leaf, let S denote the set of direct successors of e which are not leaves. Then the restriction of the k-causality event structure over the upward-closure set of S, w.r.t. the causality $\leq$, is isomorphic with $\mathcal{E}$ itself.

Lemma 3.3.8. *Given a k-causality event structure $\mathcal{E} = (E, \leq, \#)$ and an event $e \in E$ such that $\lessdot(e) \neq \emptyset$. Let e' be the unique direct successor of e which has no successor, i.e. $e \lessdot e'$ and $\lessdot(e') = \emptyset$. If $k > 0$ then $\mathcal{E}$ and $\mathcal{E}|_{>(e)\setminus\{e'\}}$ are isomorphic.*

Proof. Obvious by definition. □

As a consequence, the k-causality event structure are unique, w.r.t. isomorphism, for any given number k. Aiming at modeling the 0-initialized counter, a k-causality event structure is nothing but an underlying structure for a k-causality process defined below. One intuitively labels its leaf events by the decrement action '$-$' and its other events by the increment action '$+$'.

Definition 3.3.9 (k-causality process)**.** Let k be a natural number, the *k-causality process* is a labeled event structure $k\text{-}\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ where $\mathcal{E} = (E, \leq, \#)$ is the k-causality event structure, and

- labeling function $\mathcal{L} : E \to \{+, -\}$ defined as $\mathcal{L}(e) = -$, if e has no successor, i.e. $\lessdot(e) = \emptyset$, and $\mathcal{L}(e) = +$, otherwise;
- marking function $\mathcal{M} : \mathcal{C}_{\mathcal{E}} \to \mathbb{N}$ defined as $\mathcal{M}(C) = |\{e \in C \,/\, \mathcal{L}(e) = +\}| - |\{e \in C \,/\, \mathcal{L}(e) = -\}|$.

Figure 3.7 illustrates k-causality processes for different values of k. In causality processes, all events, which correspond to either increment action '$+$' or decrement action '$-$', are pairwise concurrent or in causality. There are two types of causality: causality between a decrement event and an increment event, or causality between two increment events. The first one naturally comes from the fact that a counter can not take a negative value, so that a decrement event must occur after some increment event. However the second type of causality is our own constraint to causality processes in order to guarantee the finite-branching property of k-causality processes.

Lemma 3.3.10. *The k-causality process $k\text{-}\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, for a given finite number $k > 0$, is a non-redundant and finitely-branching labeled event structure.*

Proof. Since there is no conflict in k-causality process, $k\text{-}\mathcal{CP}$ is thus non-redundant by Definition 3.2.14.

Let C be any configuration of $k\text{-}\mathcal{CP}$, and $X = \{e \in E \,/\, C \vdash e\}$ be the set of its extension events. Due to the downward-closure property of configurations, we have $e \in X$ only if either e is a minimal event w.r.t. causality, i.e. $e \in \mathsf{Min}_{\leq}(E)$, or e is a direct successor of some increment event e_+ in C, i.e. $\exists e_+ \in C : e_+ \lessdot e$. Therefore, the

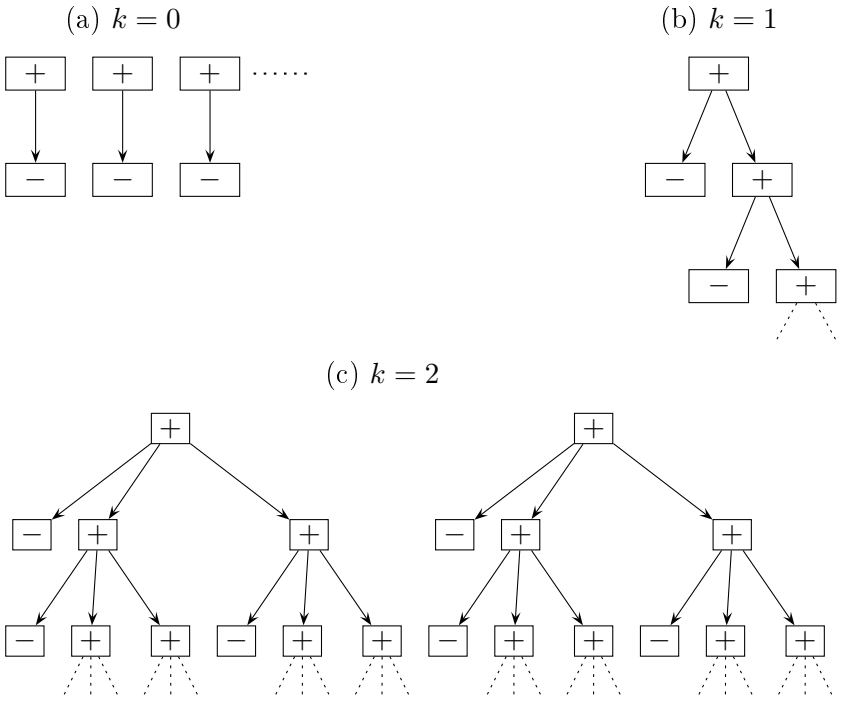


Figure 3.7: Examples of k-causality processes

cardinal of X can not exceed $|\mathsf{Min}_{\leq}(E)| + (k+1)|C| = k + (k+1)|C|$, because each increment event has exactly $(k+1)$ direct successors. It follows from the finiteness of k and of configuration C that the set X is finite. As a consequence, k-$\mathcal{CP}$ is finitely-branching by Definition 3.1.9. □

Lemma 3.3.11. *For any given number k, the k-causality process is coherent and its induced labeled transition system is the zero-initialized counter $\mathcal{CT}^0$.*

Proof. Let k-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be the k-causality process, and denote $E_+ = \{e \in E \,/\, \mathcal{L}(e) = +\}$ and $E_- = \{e \in E \,/\, \mathcal{L}(e) = -\}$. Let C be any configuration of k-$\mathcal{CP}$.

We will first prove that there exists an extension event e_+ of C such that $e_+ \in E_+$ (*). If $C \cap E_+ = \emptyset$, since $\mathsf{Min}_{\leq}(E) \neq \emptyset$ and $\mathsf{Min}_{\leq}(E) \subseteq E_+$ by definition, we can choose any event e_+ in $\mathsf{Min}(E)$ that satisfies (*). If $C \cap E_+ \neq \emptyset$, let f_+ be any maximal, w.r.t. $\leq$, increment event of C, i.e. $f_+ \in \mathsf{Max}_{\leq}(C \cap E_+)$, we have then two cases. In the first case, $k = 0$, by Definition 3.3.7, there are infinitely many minimal events. The set $(\mathsf{Min}_{\leq}(E) \setminus C)$ is not empty and contains only increment events. Any event in this set satisfies (*). In the second case, $k > 0$, let e_+ be any event in k increment direct successors of f_+. f_+ is not in C and is thus an extension event of C which satisfies.

Secondly, suppose that $\mathcal{M}(C) > 0$. We can deduce that $\mathsf{Max}_{\leq}(C) \cap E_+ \neq \emptyset$ because otherwise, for every increment event of C, its direct successor which is a decrement event is also in C. Hence $|C \cap E_+|$ can not exceed $|C \cap E_-|$, so that $\mathcal{M}(C) = 0$, contradict to the hypothesis. Let e_+ be any increment event in $\mathsf{Max}_{\leq}(C) \cap E_+$. By definition, e_+ has a direct successor $e_- \in E_-$. It is obvious that $C \vdash e_-$. The reverse, i.e. if C has an extension event $e_- \in E_-$ then $\mathcal{M}(C) > 0$, can be proved in the same manner.

Now, by definition of the marking function $\mathcal{M}$, if $C \vdash e$, we have $\mathcal{M}(C \cup \{e\}) = \mathcal{M}(C)+1$ if $e \in E_+$ and $\mathcal{M}(C \cup \{e\}) = \mathcal{M}(C)-1$ otherwise. Because for any configuration

C, it always has an extension event in E_+, and in addition, an extension event in E_- if $\mathcal{M}(C) > 0$, the k-causality process is thus coherent.

Therefore, in the induced labeled transition system, we have $\langle \mathcal{M}(C), +, \mathcal{M}(C)+1\rangle \in \rightarrow_{\mathcal{LTS}^{k\text{-}\mathcal{CP}}}$ for all configuration C of k-$\mathcal{CP}$, and $\langle \mathcal{M}(C), +, \mathcal{M}(C)+1\rangle \in \rightarrow_{\mathcal{LTS}^{k\text{-}\mathcal{CP}}}$ for all C whose marking is positive due to Definition 3.3.9 and Definition 3.2.4. Moreover, $\mathcal{M}(\emptyset) = |\emptyset \cap E_+| - |\emptyset \cap E_-| = 0$, the set of states in $\mathcal{LTS}^{k\text{-}\mathcal{CP}}$ is thus $\mathbb{N}$. Therefore, $\mathcal{LTS}^{k\text{-}\mathcal{CP}}$ is the zero-initialized counter defined in Definition 3.3.6. □

Parameter k in causality processes

The idea of our k-causality process is inspired by the unfolding technique on Petri nets [McM95a]. A counter could be intuitively considered as a place with tokens of a Petri net. The value of a counter corresponds thus to the number of tokens in this place. One can add a token to a place or remove some existing one from this place. These two actions are really independent.

The 0-causality process is a deterministic labeled event structure in which added tokens are distinguishably represented by a minimal increment event and its only direct successor. Each pair of such events with its causality can be seen as a labeled event structure for a token. Moreover, a place of a Petri net can be seen as a synchronized product of tokens without synchronization vector. As a consequence, the 0-causality process can be computed by a synchronized product of labeled event structures modeling tokens (see Section 3.3.4).

Since there is only causality between an increment event and a decrement event as naturally needed, 0-causality process is the most concurrent process. In another words, this causality process admits certain *ω-concurrency*. And by using it in synchronized products of labeled event structures, we can obtain the same structure as with labeled occurrence nets [McM95a, Haa99] or branching processes [ERV96, DJN04] on Petri nets.

However, there are three problems. First, 0-causality process may not be adapted to defining bounded counters or safe Petri nets' places (see Section 3.3.2). Second, 0-causality process' infinitely-branching property prevents itself from concurrent verification technique (see Section 4.2). Notice here that in other works, verification techniques for Petri nets is guaranteed by either the boundedness/safeness of places, which is supposed or is proved by other techniques. And third, like other k-branching processes where k is a great number, 0-branching process may give rise to enormous redundancy in a global synchronized product in which 0-branching process is used as a component. This redundancy is not easy to reduce (see Section 6.2.2 for more details).

By using a positive and finite number k in causality processes, the decidability of verification problems based on labeled event structures is guaranteed. The greater parameter k is, the less causality between increment events there is, and as a consequence, the more concurrent causality process we have. Changing k for component causality process is a heuristic way to equilibrate the concurrency of the global labeled event structure and its redundancy; so that one can obtain a more or less compact labeled event structure (see comparison results in Section 6.3.2). Intuitively, if counter's value never exceeds b, parameter k greater than b is not necessary.

The 1-causality process is a particular one in which there is a total causality (order) over increment events. Our process for bounded counters is based on it.

Bounded counters

A bounded counter differs from general counters (Definition 3.3.6) only on its set of reachable states. Its value never goes beyond some given number. Formally,

Definition 3.3.12 (b-bounded counter)**.** Let $b \in \mathbb{N}, b > 0$ and $v \in \{0, 1, \dots b\}$. A *v-initialized bounded counter* is the labeled transition system $b\text{-}\mathcal{BC}^v = (\{0, 1, \dots, b\}, \{+, -\}, \rightarrow_{\mathcal{BC}}, v)$ where the labeled transition relation $\rightarrow_{\mathcal{BC}}$ is the union set $\{\langle n, +, n+1\rangle \,/\, n \in \{0, 1, \dots, b-1\}\} \cup \{\langle n, -, n-1\rangle \,/\, n \in \{1, 2, \dots, b\}\}$.

Consider the 1-bounded counter $1\text{-}\mathcal{BC}^0$ whose initial value is zero. This labeled transition system has only two states 0 and 1 and two labeled actions '+' and '−' for switching its state. An event corresponding to the increment action gives rise to only one other event which corresponds to a decrement action, and inversely. There is no concurrency at all in the behaviors of this bounded counter. The minimal, w.r.t. isomorphism, finitely-branching and coherent labeled event structure for $1\text{-}\mathcal{BC}^0$ is thus its deterministic labeled event tree which is graphically represented in Figure 3.8.a.

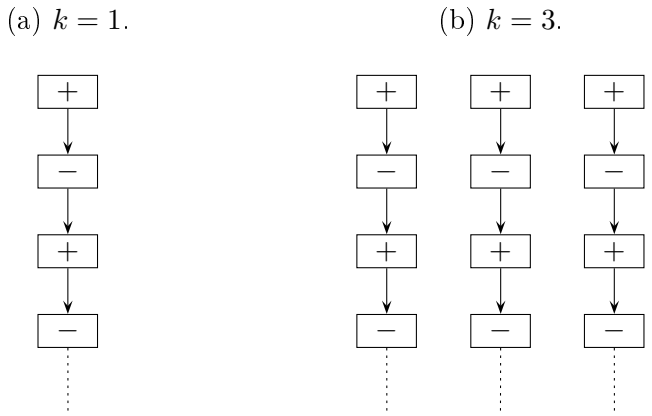


Figure 3.8: Graphical representation of k-bounded processes

As previously mentioned, a counter can be seen as a synchronized product of tokens on a Petri net's place in which every synchronization vector concerns only one component. In the case of a bounded counter, the boundedness may be certified by limiting the number of synchronized tokens to some number b. The b-bounded process defined below is somehow a synchronized product of labeled event structures which all aim at modeling behaviors of $1\text{-}\mathcal{BC}^0$. But with a slight modification on synchronized product's marking function $\mathcal{M}$ so that $\mathsf{Codom}(\mathcal{M})$ is $\mathbb{N}$, not $\mathbb{N}^b$ (see Section 3.3.4).

Definition 3.3.13 (b-bounded process)**.** Given a positive natural number b, the *b-bounded process* is the labeled event structure $b\text{-}\mathcal{BP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ where:

1. $\# = \emptyset$;
2. $|\mathsf{Min}_{\leq}(E)| = b$ and for all $e \in (E \setminus \mathsf{Min}_{\leq}(E)$, $|\gtrdot(e)| = 1$;
3. $\mathcal{L} : E \rightarrow \{+, -\}$ and $\mathcal{M} : \mathcal{C}_{b\text{-}\mathcal{BP}} \rightarrow \{0, 1, \dots, b\}$ such that $\mathcal{M}(C) = |\{e \in C \,/\, \mathcal{L}(e) = +\}| - |\{e \in C \,/\, \mathcal{L}(e) = -\}|$.
4. for every minimal event $e \in \mathsf{Min}_{\leq}(E)$, $b\text{-}\mathcal{BP}|_{(<(e))}$ is the deterministic labeled event tree of $1\text{-}\mathcal{BC}^0$; and

The first item obviously says that there is no conflict in the b-bounded process. Due to the second item, every event has one direct predecessor except b minimal events w.r.t. the causality $\leq$. As a consequence, events can be distributed in b disjoint sets, each set contains a minimal event $e_m \in \mathsf{Min}_{\leq}(E)$, and all events which are in causality with e_m. Two events from different sets are thus concurrent. The third item states that labeling functions of the b-bounded process are more or less similar to the ones for causality processes in Definition 3.3.9. The last one is the most interesting, it tells that each disjoint set of events above, determines a labeled event tree for 1-bounded counter 1-$\mathcal{BC}^0$ by means of restriction. And the number of component structures is exactly b due to the second item. As a consequence of Proposition 3.3.4, for any given b, b-bounded process is unique up to isomorphism.

Figure 3.8 illustrates two bounded processes. Since the causality in the labeled event tree of 1-$\mathcal{BC}^0$ is a total order, every event has a different label with its only direct successor. It is obvious that the marking of a configuration is determined by the label of its unique maximal event w.r.t. causality. Of the same manner,

Remark: We have $\mathcal{M}(C) = |\mathsf{Max}_{\leq}(C) \cap \{e \in E \,/\, \mathcal{L}(e) = +\}|$ for any b-bounded process.

Lemma 3.3.14. *Given a finite number b, the b-bounded process b-$\mathcal{BP}$ is deterministic and finitely-branching.*

Proof. Due to the definition of the marking function in Definition 3.3.13, b-$\mathcal{BP}$ is deterministic by Definition 3.2.2. Let $C \in \mathcal{C}_{b\text{-}\mathcal{BP}}$ be any configuration, C has exactly b extension events which are separately located in b different sub-structures of b-$\mathcal{BP}$. b-$\mathcal{BP}$ is thus finitely-branching due to Definition 3.1.9. □

Lemma 3.3.15. *Given a positive natural number b, the b-bounded process b-$\mathcal{BP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ is coherent and its induced labeled transition system $\mathcal{LTS}^{b\text{-}\mathcal{BP}}$ is the b-bounded counter b-$\mathcal{BC}^0$ whose initial state is 0.*

Proof. As mentioned, since b-$\mathcal{BP}$ is intuitively composed of b labeled event trees whose causality is total, for any configuration $C \in \mathcal{C}_{b\text{-}\mathcal{BP}}$, $\mathsf{Max}_{\leq}(C)$ is not greater than b. And moreover, any of its extension event is either a direct successor of some maximal event of C w.r.t. $\leq$, or a minimal event of E.

Let us denote $E_+ = \{e \in E \,/\, \mathcal{L}(e) = +\}$ and $E_- = \{e \in E \,/\, \mathcal{L}(e) = -\}$. Due to the remark above, if $\mathcal{M}(C) = k$ then $\mathsf{Max}_{\leq}(C) \cap E_- = \emptyset$, thus C can not have extension event labeled by '+'. If $\mathcal{M}(C) < k$, we have two cases. First, if $\mathsf{Min}_{\leq}(E) \not\subseteq C$, every event in the non-empty set $\mathsf{Min}_{\leq}(E) \setminus C$ is an extension event of C and is an increment event. Second, if $\mathsf{Min}_{\leq}(E) \subseteq C$, we have $|\mathsf{Max}_{\leq}(C)| = k$. Since $\mathcal{M}(C) < k$, C has at least one maximal event which is a decrement event, and its unique direct successor, which is an increment event, is thus an extension event of C.

Therefore, C has extension event which is an increment event if and only if $\mathcal{M}(C) < k$. In the same manner, we can prove that C has extension event labeled '$-$' if and only if $\mathcal{M}(C) > 0$ (1).

By definition of the marking function $\mathcal{M}$, when $C \vdash e$ we have $\mathcal{M}(C \cup \{e\}) = \mathcal{M}(C) + 1$ if $e \in E_+$, and $\mathcal{M}(C \cup \{e\}) = \mathcal{M}(C) - 1$ if $e \in E_-$ (2). The b-bounded process is coherent by Definition 3.2.11.

In addition, $\mathcal{M}(\emptyset) = |\emptyset \cap E_+| - |\emptyset \cap E_-| = 0$. From (1) and (2), we can conclude that the induced labeled transition system $\mathcal{LTS}^{b\text{-}\mathcal{BP}}$ is the b-$\mathcal{BC}^0$ defined in Definition 3.3.12. □

Now let us discuss the possibility of adapting a k-causality process k-$\mathcal{CP}$ to model behaviors of b-bounded counters for some given number b. Our idea is to add some causality between events so that the obtained labeled event structure, denoted by $\mathcal{E}$, disallows all configurations of marking b to have an extension event which is an increment event. Let $C \in \mathcal{C}_{k\text{-}\mathcal{CP}}$ be a configuration with $\mathcal{M}(C) = b$, X_+ and X_- are the sets of extension events of C which are labeled by '+' and '−' respectively. Since causality processes' induced labeled transition system $\mathcal{LTS}^{k\text{-}\mathcal{CP}}$ is zero-initialized counter, we have $|X_-| = b$ and $X_+ \neq \emptyset$.

Naturally, any event in X_- is still an extension event of C in $\mathcal{E}$ while an event in X_+ is not. Moreover, let e_+ be any event in X_+, e_+ could be an extension event of the configuration $(C \cup \{e_-\}) \in \mathcal{C}_{\mathcal{E}}$ for some (or all) event $e_- \in X_-$. Due to the absolute concurrency between decrement events in k-$\mathcal{CP}$, this idea is difficult to be implemented. We need somehow a total causality over X_- and to impose the causality between e_+ and the minimal event of X_-, w.r.t. $\leq$, afterward.

Due to the intuitive idea above, we can only conform 1-causality process to modeling bounded counters' behaviors. The goal labeled event structures are isomorphic with bounded M-causality processes for FIFO channels where M is a singleton (see Section 3.3.3).

Counters initialized by positive values

The v-initialized counter can be seen as a Petri net's place in which there are initially v tokens. Behaviors of such a place are the same as behaviors of an empty place, modeled by means of 0-initialized counter, combined with concurrent events which remove initial tokens. These v tokens with only removing operations can be modeled by a simple labeled transition system, called v-countdown counter.

Definition 3.3.16 (v-countdown counter)**.** Given a number v, the *v-countdown counter* is the labeled transition system $\mathcal{CD}^v = (\{0, 1, \ldots, v\}, \{-\}, \rightarrow, v)$ where the transition relation $\rightarrow$ is the set $\{\langle n, -, n-1\rangle \,/\, n \in \{1, \ldots, v\}\}$.

Since all events corresponding to the decrement action, labeled by '−', are pairwise concurrent and there are at most v events. We define a labeled event structure for countdown counters in which there is no strict causality and conflict. However, by omitting this total concurrency, one can also gives other labeled event structures for representing countdown counters„ for instance, labeled event trees.

Definition 3.3.17 (v-countdown process)**.** Given a number v, the *v-countdown process* is a labeled event structure v-$\mathcal{CD} = (E, \mathcal{I}_E, \emptyset, \mathcal{L}, \mathcal{M})$ where:

- there are exactly v events, i.e. $|E| = v$,
- labeling function $\mathcal{L} : E \times \{-\}$, and
- marking function $\mathcal{M} : \mathcal{C}_{v\text{-}\mathcal{CD}} \rightarrow \{0, 1, \ldots, v\}$ is defined as $\mathcal{M}(C) = v - |C|$.

Figure 3.9: The 4-countdown process

The following is straightforward.

Lemma 3.3.18. *Let $v \in \mathbb{N}$ be any number, the v-countdown process v-$\mathcal{CD}$ is a deterministic, finitely-branching and coherent labeled event structure for v-countdown counter $\mathcal{CD}^v$ defined in Definition 3.3.16.*

Our labeled event structure for v-initialized counters intuitively consists of a v-countdown process and a k-causality process, for a given number k.

Definition 3.3.19 ((k,v)-causality process)**.** Let k, v be two natural numbers, the (k,v)-*causality process* is a labeled event structure $(k,v)\text{-}\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ where E is the union set of two disjoint sets E_v and E_k such that:

1. $(k,v)\text{-}\mathcal{CP}|_{E_v}$ is $v\text{-}\mathcal{CD}$ w.r.t. isomorphism,
2. $(k,v)\text{-}\mathcal{CP}|_{E_k}$ is $k\text{-}\mathcal{CP}$ w.r.t. isomorphism,
3. $\leq = (\leq|_{E_v}) \cup (\leq|_{E_k})$ and $\# = \emptyset$,
4. $\mathsf{Dom}(\mathcal{L}) = E$ and $\mathsf{Codom}(\mathcal{L}) = \{+, -\}$, and
5. marking function $\mathcal{M} : \mathcal{C}_{(k,v)\text{-}\mathcal{CP}} \to \mathbb{N}$ is defined as $\mathcal{M}(C) = v + |\{e \in E \,/\, \mathcal{L}(e) = +\}| - |\{e \in E \,/\, \mathcal{L}(e) = -\}|$.

Although the labeling function $\mathcal{L}$ is not explicitly defined in Definition 3.3.19, thanks to the first and second items, one can see that $\mathcal{L}$ is the union of two disjoint functions $\mathcal{L}|_{E_v}$ and $\mathcal{L}|_{E_k}$ which are well defined. So that, for instance, events in E_v are all labeled by the decrement action '$-$' by Definition 3.3.17. Figure 3.10 shows causality processes with different parameters k and v. For instance, in the $(3,2)$-causality process (Figure 3.10.c), we have $E_v = \{e_v^1, e_v^2, e_v^3\}$ and $E_k = E \setminus E_v$. Notice that when $v = 0$, i.e. $E_v = \emptyset$, the $(k,0)$-causality process is the k-causality process in Definition 3.3.9.

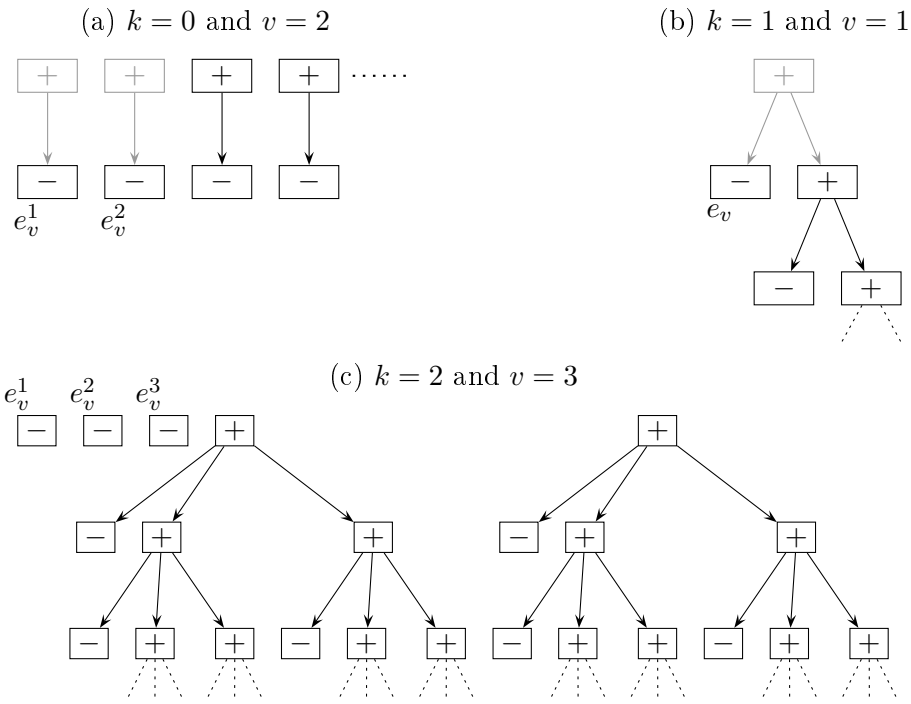


Figure 3.10: Example of (k,v)-causality processes

Figure 3.10.a and Figure 3.10.b show the (k,v)-causality processes together with its corresponding k-causality processes for two particular values of k: $k = 0$ and $k = 1$. In these two figures, events in k-causality process which are not in the (k,v)-causality process are colored in gray. One can find out that, (k,v)-causality process is intuitively a suffix of k-causality process for every value of v. However, it is true only when k is either

0 or 1. As a consequence, it follows from Lemma 3.2.10 that induced labeled transition systems of $(0, v)$-causality processes and $(1, v)$-causality processes are all v-initialized counters.

Lemma 3.3.20. *Let k, v be two numbers. (k, v)-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ is a finitely-branching if $k > 0$, deterministic, non-redundant and coherent labeled event structure whose induced labeled transition system $\mathcal{LTS}^{(k,v)\text{-}\mathcal{CP}}$ is the v-initialized counter v-$\mathcal{CT}$ in Definition 3.3.6.*

Proof. By definition of the marking function $\mathcal{M}$, (k, v)-$\mathcal{CP}$ is deterministic by Definition 3.2.2. It is also non-redundant by Definition 3.2.14 because there is no conflict, i.e. $\# = \emptyset$.

Let C be any configuration of (k, v)-$\mathcal{CP}$. By definition, (k, v)-$\mathcal{CP}$ is intuitively a combination of v-$\mathcal{CD}$, k-$\mathcal{CP}$ of which event sets are respectively E_v, E_k, and these sets are concurrent, i.e. $E_v \parallel^s E_k$. Hence, every extension event of C in (k, v)-$\mathcal{CP}$ is either an extension event of $C \cap E_v$ in v-$\mathcal{CD}$ (number of such events can not exceed v) or an extension event of $C \cap E_k$ in k-$\mathcal{CP}$ (number of such events is finite if $k > 0$ due to Lemma 3.3.10). Therefore, (k, v)-$\mathcal{CP}$ is finitely-branching.

Now, we are going to prove the coherence of (k, v)-$\mathcal{CP}$. Let X be the set of all extension events of C in (k, v)-$\mathcal{CP}$, and denote by $X_+ = X \cap E_+$, $X_- = X \cap E_-$ its two disjoint subsets of increment events, decrement events respectively. X_+ is not empty since C has at least one extension event $e_+ \in E_+$ in k-$\mathcal{CP}$ due to Lemma 3.3.11.

By definition of marking function $\mathcal{M}$, $\mathcal{M}(C) = 0$ (*) if and only if $|C \cap E_-| = v + |C \cap E_+|$. Since $E_v \subset E_-$, we can write $|C_v| + |C_k \cap E_-| = v + |C_k \cap E_+|$ where $C_v = C \cap E_v$ and $C_k = C \cap E_k$. Because C_k is a configuration of k-$\mathcal{CP}$, so the number of its decrement events can not be greater than the one of its increment events; and $|C_v|$ is less than or equal to the number of events in v-$\mathcal{CD}$, hence (*) if and only if $C_v = E_v$ and $\mathcal{M}_{k\text{-}\mathcal{CP}}(C_k) = 0$. In this case, $X_- \subseteq E_k$, and X_- is thus empty due to Lemma 3.3.11. Otherwise, i.e. either $C_v \subset E_v$ or $\mathcal{M}_{k\text{-}\mathcal{CP}}(C_k) \neq 0$, X_- contains at least one event which is in $E_v \setminus C_v$ or is an event extension $e_- \in E_k$ of C_k in k-$\mathcal{CP}$.

Therefore, for all configurations C of (k, v)-$\mathcal{CP}$, C always has an extension which is an increment event, and C has an extension which is an decrement event if and only if $\mathcal{M}(C) > 0$. Moreover, by definition, $\mathcal{M}(C \cup e) = \mathcal{M}(C) + 1$ if $e \in E_+$, and $\mathcal{M}(C \cup e) = \mathcal{M}(C) - 1$ otherwise. (k, v)-$\mathcal{CP}$ is thus coherent.

In addition, since $\mathcal{M}(\emptyset) = v$, the induced labeled transition $\mathcal{LTS}^{(k,v)\text{-}\mathcal{CP}}$ is the v-initialized counter v-$\mathcal{CT}$ in Definition 3.3.6. □

Remark: Combination of labeled event structures like in (k, v)-causality processes could be applied to put together, for instance, causality processes with different parameters k, and then with any coherent and deterministic labeled event structure for v-bounded counters. But notice that the number of these component processes should be finite in order to guarantee the finitely-branching property of the global one. In this way, we can obtain coherent labeled event structures for counters. The proof's idea is the same as the one of Lemma 3.3.20.

Labeled event structures for bounded counters which are initialized by a given number, are just suffixes of bounded processes. As a consequence, they inherit the coherence, finitely-branching, and non-redundancy of bounded processes. Moreover, there is only one labeled event structure, w.r.t. isomorphism, for each pair of parameters bound k and initial value v. The following is thus a direct sequence of Lemma 3.2.10.

Lemma 3.3.21. *Given natural numbers $k > 0$, and a natural number v which is not greater than k. Let C be any configuration in the k-bounded process $b\text{-}\mathcal{BP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ satisfying $\mathcal{M}(C) = v$. The C-suffix of $b\text{-}\mathcal{BP}$, i.e. $b\text{-}\mathcal{BP}|_{E\setminus C\setminus \#(C)}$, is a labeled event structure for the k-bounded counter which is initialized by v.*

3.3.3 FIFO channels

Nowadays, many works on verification aim at verifying communication protocols. The popular model *Communicating Finite State Machine* [BZ83] for specifying and verifying these protocols, can be considered as a synchronized product of some *FIFO (First-In-First-Out) channels* and some other finite-state labeled transition systems. FIFO channel is thus a standard model which allows to represent the exchange of messages in a communication protocol.

Intuitively, a FIFO channel is a variable holding a finite word over some alphabet M. This word determines its current state. At each time, the environment, e.g. the client of a server, can either remove the first letter of this word, by a so called *receiving* operation, or insert a new letter in M after the last letter of this word, by a so called *sending* operation.

Definition 3.3.22 (v-initialized FIFO channel over M)**.** Let M be a non-empty alphabet and v be any finite word over M. The *v-initialized FIFO channel over M* is the labeled transition system $(M, v)\text{-}\mathcal{FF} = (M^*, \Sigma, \rightarrow, v)$ where

- action set $\Sigma = \{!m \,/\, m \in M\} \cup \{?m \,/\, m \in M\}$, and
- transition relation $\rightarrow = \{\langle w, !m, w.m\rangle \,/\, m \in M, w \in M^*\} \cup \{\langle m.w, ?m, w\rangle \,/\, m \in M, w \in M^*\}$.

Notation 3.3.23. We denote by $!M$ the set $\{!m \,/\, m \in M\}$ and call it the *sending action set*; and respectively $?M = \{?m \,/\, m \in M\}$ the *receiving action set*.

Figure 2.2 on page 15 illustrates an example of $(M, v)\text{-}\mathcal{FF}$ where $M = \{a, b\}$ and $v = a$ (Example 2.4.3). Although a state of a FIFO channel is a finite word, there is no limit on its size. As a consequence, any sending action $!m$ where $m \in M$ is always enabled. However, a receiving action $?m$ is only enabled from a state $w \in M^*$ if m is a prefix of the word w. Reachable states of $(M, v)\text{-}\mathcal{FF}$ can be computed by means of *M-letter-morphisms* defined as follow:

Definition 3.3.24 (M-letter-morphisms)**.** The *M-letter-morphisms* $\Pi_{!M}$ and $\Pi_{?M}$ are two functions from $(!M \cup ?M)$ to $(M \cup \{\varepsilon\})$, where ε is the empty word, such that

- $\Pi_{!M}(!m) = \Pi_{?M}(?m) = m$ for all $m \in M$, and
- $\Pi_{!M}(?m) = \Pi_{?M}(!m) = \varepsilon$ for all $m \in M$.

Recall that the function $\Pi^{\mathcal{W}}_{!M}$ ($\Pi^{\mathcal{W}}_{?M}$) is based on $\Pi_{!M}$ ($\Pi_{?M}$, resp.) (see Section 2.2 on page 12). By definition, for a given word $w \in (!M \cup ?M)$, in order to obtain $\Pi^{\mathcal{W}}_{!M}(w)$ ($\Pi^{\mathcal{W}}_{?M}(w)$, resp.), one intuitively 'erases' all letters in $?M$ ($!M$, resp.) of w, then 'erases' all 'notes of exclamation' ('question marks', resp.).

Let $\sigma \in (!M \cup ?M)^*$ be any firing sequence of $(M, v)\text{-}\mathcal{FF}$, and w be its only reachable state (because $(M, v)\text{-}\mathcal{FF}$ is deterministic), i.e. $v \xrightarrow{\sigma} w$. We have that

$$w = (\Pi^{\mathcal{W}}_{?M}(\sigma))^{-1}(v.\Pi^{\mathcal{W}}_{!M}(\sigma))$$

Intuitively, one can first insert all messages according to sending actions in the firing sequence σ, to obtain the word $w' = (v.\Pi^{\mathcal{W}}_{!M}(\sigma))$. Then, by removing all messages according to receiving actions in σ from w', one finally gets the reachable state w. This inserting order as well as this removing order should respect to the order of actions in σ. Therefore, such removing messages form the prefix $\Pi^{\mathcal{W}}_{?M}(\sigma)$ of w' while w is a suffix of w'.

The key idea of modeling a FIFO channel by some labeled event structure $\mathcal{E}$ is that every event labeled by a sending action $!m \in !M$, shortly called a *sending event*, gives rise to another event labeled by $?m \in ?M$, called a *receiving event*, that should be a successor of the sending one. This relation between sending events and receiving ones is a bijection. Moreover, it follows from the total order of messages in FIFO channels that sending events should not be concurrent. In order to avoid redundancy, it is natural that a sending event $e_!$ has $|M|$ direct sending successors which correspond to different sending actions in $!M$. The receiving event $e_?$ associated to $e_!$ is hopefully a direct successor of $e_!$ and concurrent with other direct sending successors of $e_!$.

Aiming at labeled event structures for FIFO channels, and at first for the empty-initialized FIFO channel, one may think about using k-causality event structures defined in Definition 3.3.7. Recall that, given a k-causality event structure $\mathcal{E} = (E, \leq, \#)$, let $E_- = \{e \in E \,/\, \lessdot(e) = \emptyset\}$ and $E_+ = E \setminus E_-$, hence E_+ and E_- correspond respectively to the increment event set and the decrement event set in the k-causality process. Each increment event $e_+ \in E_+$ has exactly $k+1$ direct successors, and among them, there is only one decrement event. Moreover, $\mathcal{B} = \{\langle e_+, e_?\rangle \in (E_+ \times E_-) \,/\, e_+ \lessdot e_-\}$ is a bijection from E_+ to E_-. The bijection $\mathcal{B}$ could be obtain in another way that $\mathcal{B} = (E_+ \times E_-) \cap \lessdot$. In the following definition of M-causality event structure, for an alphabet M, we simply use the k-causality event structure where k is the cardinal of M, i.e. $k = |M|$.

Definition 3.3.25 (M-causality event structure)**.** The *M-causality event structure*, for a given non-empty alphabet M, is an event structure $\mathcal{E} = (E, \leq, \#)$, where E is union of two disjoint sets $E_!$ and $E_?$, such that:

1. let $\leq' = (\leq \setminus (E_? \times E_?)) \cup \mathcal{I}_{E_?}$, then $(E, \leq', \emptyset)$ is the $|M|$-causality event structures, and $E_? = \{e \in E \,/\, \lessdot'(e) = \emptyset\}$;
2. let $\mathcal{B}_!$ be the bijection defined by $\mathcal{B}_! = (E_! \times E_?) \cap \lessdot'$ and let $\mathcal{B}_? = \mathcal{B}_!^{-1}$, then for all $e_?, f_? \in E_?$, $e_? \leq f_?$ iff $\mathcal{B}_?(e_?) \leq' \mathcal{B}_?(f_?)$; and
3. $\{\langle e, f\rangle \in \# \,/\, >(e)\, \overline{\#^s}\, >(f)\} = \{\langle e_!, f_!\rangle \in (E_! \times E_!) \,/\, e_! \neq f_! \text{ and } >(e_!) = >(f_!)\}$. [1]

An M-causality event structure is simply a k-event structure, where $k = |M|$, with additional causality and conflict as stated in the first item. Events are separated into two sets $E_!$ and $E_?$ which respectively represent sending events and receiving events. Because $E_? = \{e \in E \,/\, \lessdot'(e) = \emptyset\}$, receiving events intuitively correspond to decrement events in $|M|$-causality process.

Suppose that the FIFO channel is initially empty, a message in the FIFO channel must be inserted by a sending event and could be removed by another receiving event. These two events are related by bijections $\mathcal{B}_!$ and $\mathcal{B}_?$ defined in the second item. It is obvious that the receiving event must occur after the sending one, thanks to the causality $\leq'$ in the k-causality event structure, so that $e_! \lessdot' \mathcal{B}_!(e_!)$ and equally, $e_! \lessdot \mathcal{B}_!(e_!)$ for all $e_+ \in E_!$. Moreover, the environment can only receive messages in the order that they were sent into the channel due to its First-In-First-Out property, this fact gives rise to

[1] $\overline{\#^s}$ is the complement of $\#^s$ (see Section 2.1 on page 11)

a causality on the set of receiving events which respects the causality on the set of its corresponding sending events. More precisely, two receiving events $e_? , f_? \in E_?$ are causal, for example $e_? < f_?$, if and only if in the channel, the corresponding message of $e_?$ is inserted before the one of $f_?$. The second condition is guaranteed when corresponding sending events are causal, i.e. $\mathcal{B}_?(e_?) < \mathcal{B}_?(f_?)$ which is equivalent to $\mathcal{B}_?(e_?) <' \mathcal{B}_?(f_?)$ by definition.

In the third item, the set $\{\langle e, f\rangle \in \# \,/\, > (e) \overline{\#^s} > (f)\}$ represents the relation of minimal conflict on events, denoted by $\#_m$. In words, $e \#_m f$ if events in the downward closure $\geq(\{e, f\})$ are pairwise either causal or concurrent, except the pair e and f. And when $e \# f$ and $e \overline{\#_m} f$, we say that e and f are in conflict due to conflict inheritance w.r.t. causality, that means there exists two predecessors of e and f which are in conflict. Hence, the third item states that minimal conflict $\#^m$ in M-causality event structures comes from the conflict between sending events which are extension events of a same configuration, which correspond to sending actions from a same state. Intuitively, given any word $w \in M^*$ which is some current state of a FIFO channel $\mathcal{FF}$, one can firstly send a message $a \in M$ and then another message $b \in M$, or conversely. However, since messages in $\mathcal{FF}$ are totally ordered, if a differs from b, one thus obtains different states by permuting this successive sending actions $!a$ and $!b$. Formally, because $w \xrightarrow{!a!b} w.a.b$ and $w \xrightarrow{!b!a} w.b.a$, so $a \neq b$ implies $w.a.b \neq w.b.a$. Sending actions are not independent.

For the goal of having a non-redundant labeled event structure, each sending event in M-causality event structure has exactly $|M|$ sending direct successors. These direct successors are pairwise in conflict, and moreover, it is the origin of the minimal conflict $\#^m$ from which the whole conflict relation $\#$ can be computed due to conflict inheritance. Therefore, all sending events are either in causality or in conflict. The concurrency in FIFO channel is formally represented by concurrency in the M-causality event structure.

Proposition 3.3.26. *Let $\mathcal{E} = (E, \leq, \#)$ be any M-causality event structure defined in Definition 3.3.25, for a given non-empty alphabet M.*

$$\| = \{\langle e_?, f_!\rangle, \langle f_!, e_?\rangle \,/\, e_? \in E_?, f_! \in E_! \text{ and } \mathcal{B}_?(e_?) < f_!\}$$

Proof. As previously mentioned, all sending events are pairwise either causal due to causality in its k-causality event structure, or in conflict by the third item of Definition 3.3.25. Hence there exists no concurrency between sending events (*). And so do for receiving events. Because, suppose the opposite, let $e_?, f_?$ be any receiving events which are concurrent. As a consequence of the second item in Definition 3.3.25, the sending events correspond to $e_?$ and $f_?$ w.r.t. $\mathcal{B}_?$ are also concurrent, i.e. $\mathcal{B}_?(e_?) \parallel \mathcal{B}_?(f_?)$. This contradicts (*). We can conclude that

$$\| \cap (E_! \times E_!) = \| \cap (E_? \times E_?) = \emptyset$$

Now, let $e_? \in E_?$ be any receiving event, and $f_! \in E_!$ be any sending one. If $\mathcal{B}_?(e_?)$ and $f_!$ are conflict, then since $e_?$ is a direct successor of $\mathcal{B}_?(e_?)$, it is also conflict with $f_!$ due to conflict inheritance. Otherwise, i.e. $\mathcal{B}_?(e_?) \overline{\#} f_!$, as explained above, $\mathcal{B}_?(e_?)$ and $f_!$ must be in causality. There are thus two cases. First, $f_! \leq \mathcal{B}_?(e_?)$, we have $f_! < e_?$ because $\mathcal{B}_?(e_?) \lessdot e_?$. Second, $\mathcal{B}_?(e_?) < f_!$, in the k-causality event structure, $e_?$ and $f_!$ are concurrent, hence $\langle e_?, f_!\rangle \notin (\leq' \cup \geq')$. We have thus $\langle e_?, f_!\rangle \notin (\leq \cup \geq)$ because $\leq \cap (E_? \times E_!) = \leq' \cup (E_? \times E_!) = \emptyset$. Moreover, we have $\geq(\mathcal{B}_?(e_?)) \subset \geq(e_?)$ and $\geq(f_!) = \geq(\mathcal{B}_?(e_?)) \cup F_!$ where $F_! \subset E_!$ is the set of all events in the path from $\mathcal{B}_?(e_?)$ to $f_!$ in the k-causality structure. Therefore, suppose that $\mathcal{B}_?(e_?) \# f_!$, this conflict must be inherited from some minimal conflict $\mathcal{B}_?(e_?) \#_m f'_!$ where $f'_! \in F_!$. This contradicts the

fact that $\#_m \subset (E_! \times E_!)$ stated in the third item of Definition 3.3.25. Hence, in this case, $e_? \parallel f_!$ if and only if $\mathcal{B}_?(e_?) < f_!$. Finally,

$$\begin{aligned} \parallel &= \parallel \cap ((E_! \times E_?) \cup (E_? \times E_!)) \\ &= \{\langle e_?, f_! \rangle, \langle f_!, e_? \rangle \,/\, e_? \in E_?, f_! \in E_! \text{ and } \mathcal{B}_?(e_?) < f_!\} \end{aligned}$$

□

Proposition 3.3.26 intuitively says that a receiving event $e_?$ is concurrent with all sending events which occur after the one corresponding to $e_?$, i.e. $\mathcal{B}_?(e_?)$. And all concurrency in M-causality event structures is of this type.

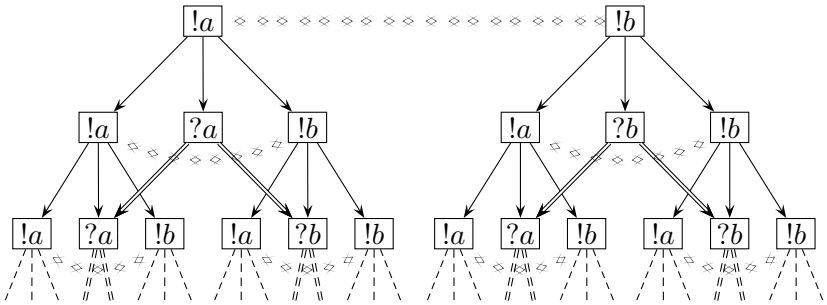


Figure 3.11: M-causality processes where $M = \{a, b\}$

Figure 3.11 gives an example of M-causality process, defined in the following, as well as its M-causality event structure. Since $M = \{a, b\}$, one can find out that it is similar to the 2-causality event structure in Figure 3.7.c. In addition to the 2-causality event structure, there are the minimal conflict $\#_m$ between sending events $E_!$ and the causality between receiving events $E_?$. This new causality is represented by double-line arrows. and are shown in red color. Intuitively, each receiving event has two direct successors that are also receiving events.

Remark: Thanks to Proposition 3.3.26, given a configuration C of a M-causality event structure, its sending events $(C \cap E_!)$ are totally ordered. And so do the receiving events $(C \cap E_?)$. As a consequence, for all configuration C, there is a unique linearisation of $(C \cap E_!)$, and a unique one of $(C \cap E_?)$, w.r.t. the causality.

Definition 3.3.27 (M-causality process)**.** Let M be a non-empty alphabet. The *M-causality process* is a labeled event structure $M\text{-}\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ where $((E_! \cup E_?), \leq, \#)$ is the M-causality event structure defined in Definition 3.3.25, and

- labeling function $\mathcal{L} : (E_! \cup E_?) \to (!M \cup ?M)$ such that
  1. $\mathsf{Codom}(\mathcal{L}|_{E_!}) = !M$ and $\mathsf{Codom}(\mathcal{L}|_{E_?}) = ?M$,
  2. for all $e_! \in E_!$, $\Pi_{!M}(\mathcal{L}(e_!)) = \Pi_{?M}(\mathcal{L}(\mathcal{B}_!(e_!)))$,
  3. for all $e_! \in E_!$, let $F_! = (\lessdot(e_!) \cap E_!)$, then $\mathcal{L}|_{F_!}$ is a bijection between $F_!$ and $!M$;
  4. $\mathcal{L}|_{\mathsf{Min}_\leq(E)}$ is a bijection between $\mathsf{Min}_\leq(E)$ and $!M$,
- marking function $\mathcal{M} : \mathcal{C}_{M\text{-}\mathcal{CP}} \to M^*$ defined by $\mathcal{M}(C) = (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_!)))$ where $\sigma_!$ and $\sigma_?$ are respectively the linearisations, w.r.t. the causality $\leq$, of $(C \cap E_!)$ and $(C \cap E_?)$.

In words, the labeling function $\mathcal{L}$ says that: first, sending events $E_!$ are labeled by sending actions $!M$ while receiving events $E_?$ are labeled by receiving actions $?M$; second, by means of M-letter morphisms $\Pi_{!M}$ and $\Pi_{?M}$, sending events and receiving events which are related by the bijection $\mathcal{B}_!$, as well as by $\mathcal{B}_?$, must concern a same message; and third, events in the sending direct successor set of any sending event $e_! \in E_!$, denoted by $F_!$, must be pairwise distinguishably labeled. Since $F_! = |M|$ by Definition 3.3.7, we have thus $\mathcal{L}(F_!) = \{\mathcal{L}(e) \,/\, e \in F_!\} = !M$. The fourth property of labeling function $\mathcal{L}$ is like the third one but the set of sending events here is the set of minimal events w.r.t. causality.

Notice that $\sigma_!$ and $\sigma_?$ in the definition of a marking function are linearisations which are considered as words over alphabets $(C \cap E_!)$ and $(C \cap E_?)$ respectively. Therefore, $\mathcal{L}^{\mathcal{W}}(\sigma_!)$ as well as $\mathcal{L}^{\mathcal{W}}(\sigma_!.\sigma_?)$ may be firing sequences of some (M, v)-$\mathcal{FF}$. The definition of marking function $\mathcal{M}$ in Definition 3.3.27 respects to the way of computing reachable states in the empty-initialized FIFO channel (M, ε)-$\mathcal{FF}$ (see Definition 3.3.22 on page 43).

Lemma 3.3.28. *Let M-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be the M-causality process for a non-empty alphabet M. M-$\mathcal{CP}$ is a coherent labeled event structure for (M, ε)-$\mathcal{FF}$.*

Proof. Let $C \in \mathcal{C}_{M\text{-}\mathcal{CP}}$ be any configuration of the M-causality process, let us denote $C_! = C \cap E_!$ and $C_? = C \cap E_?$. We first shows how the marking of C is computed from $C_!$ and $C_?$. Since $\mathcal{B}_?(e_?) \lessdot e_?$ for all receiving events $e_?$, we have thus $\mathcal{B}_?(C_?) \subseteq (\geq(C_?))$. It follows from the downward-closure of C w.r.t. the causality $\leq$ that $\mathcal{B}_?(C_?)$ must be a subset of sending event set $C_!$. Moreover, let $C'_! = C_! \setminus \mathcal{B}_?(C_?)$, we must have that, for all $\langle e_!, e'_!\rangle \in (\mathcal{B}_?(C_?) \times C'_!)$, $e_! < e'_!$ (1). Because, as mentioned above, $(C_!, \leq|_{C_!})$ is a totally-ordered set, if $e'_! < e_!$ then $\mathcal{B}_!(e'_!) < \mathcal{B}_!(e_!)$ by the second item of Definition 3.3.25. Hence the receiving event $\mathcal{B}_!(e'_!)$ must be in $C_?$, and as a consequence, its corresponding sending event $e'_!$ is in $\mathcal{B}_?(C_?)$. This contradicts to the fact that $C'_! \cap \mathcal{B}_?(C_?) = \emptyset$. Therefore, from (1), the unique linearisation $\sigma_!$ of $C_!$, w.r.t. causality $\leq$, must be $\sigma_{\mathcal{B}}.\sigma'_!$ where $\sigma_{\mathcal{B}}$ is the linearisation of $\mathcal{B}_?(C_?)$ w.r.t. $\leq$. By definition of the labeling function $\mathcal{L}$, we have $\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)) = \Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_{\mathcal{B}}))$, where $\sigma_?$ is the linearisation of $C_?$ w.r.t. $\leq$. The marking $\mathcal{M}(C)$ in Definition 3.3.27 can be computed as follows:

$$\begin{aligned}\mathcal{M}(C) &= (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_{\mathcal{B}}.\sigma'_!)))\\ &= (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_{\mathcal{B}}))).(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma'_!)))\\ &= (\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma'_!))) \qquad (2)\end{aligned}$$

Now, let $e'_!, f'_!$ be respectively the minimal and maximal events, if they exist, of $C'_!$ w.r.t. $\leq$. Let $x_? = \mathcal{B}_!(e'_!)$ be the direct receiving successor of $e'_!$ and $X_! = \{f_! \in E_! \,/\, f'_! \lessdot f_!\}$ be the set of direct sending successors of $f'_!$. By definition, $(\{x_?\} \cup X_!) \parallel^s C$. Since for all $x_! \in X_!, \geq(x_!) = \geq(f'_!) \cup \{x_!\} = C_! \cup \{x_!\}$, and $\geq(x_?) = \geq(e'_!) \cup \mathcal{B}_!(\geq(e'_!) \setminus \{e'_!\}) = (\mathcal{B}_?(C_?) \cup \{e'_!\}) \cup C_?$, then $x \cup C$ are thus downward closed, and moreover conflict-free (by Definition 3.3.25). Hence $X = X_! \cup \{x_?\}$ is the set of extension events of C, because all other events $e \in (E \setminus (C \cup X))$ which is not conflict with events in C, must be either a successor of some some event $x_! \in X_!$ if $e \in E_!$, or a successor of $x_?$ if $e \in E_?$.

Notice that $|X_!|$ is the number of $f_!$'s direct successors, $|X_!|$ must be $|M|$. It follows from the bijection $\mathcal{L}|_{X_!}$ between $X_!$ and $!M$ by Definition 3.3.27 that, for every $m \in M$, there exists a sending event $x_! \in X_!$ satisfying $\mathcal{L}(x_!) = !M$. In other words, C has an extension event that corresponds to any sending action in $!M$ (3). And if $\mathcal{M}(C) \neq \varepsilon$, let m be the first letter (or message) in the word $\mathcal{M}(C) = \Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma'_!))$. Since $e'_!$ is the minimal event, w.r.t. $\leq$, and $C'_!$ is totally ordered by $\leq$, the first letter of $\mathcal{L}^{\mathcal{W}}(\sigma')$ must be $\mathcal{L}(e'_!)$. We have thus $\Pi_{!M}(e'_!) = m$, as a consequence, $\mathcal{L}(e'_!) = !m$ and $\mathcal{L}(x_?) = ?m$.

Therefore, C has one and only one extension event labeled by $?m$ if and only if m is a prefix of $\mathcal{M}(C)$ (4).

Let $x_!$ be any sending extension event of a given label $!m$, i.e. $\mathcal{L}(x_!) = !m$. Because, for all $e \in C'_!, e < x_!$, from (2) we have

$$\begin{aligned}\mathcal{M}(C \cup x_!) &= (\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma'_!.x_!))) \\ &= (\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma'_!))).\Pi_{!M}(\mathcal{L}(x_!)) \\ &= \mathcal{M}(C).m\end{aligned}$$

And if C has a receiving extension event $x_? \in E_?$ with label $?m$, i.e. $C \vdash x_?$ and $\mathcal{L}(x_?) = ?m$, once again, due to (2),

$$\begin{aligned}\mathcal{M}(C \cup x_?) &= (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?.x_?)))^{-1}(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_{\mathcal{B}}.\sigma'_!))) \\ &= (\Pi_{!M}(\mathcal{L}(x_!)))^{-1}.(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma'_!))) \\ &= m^{-1}.\mathcal{M}(C)\end{aligned}$$

In coordination with (3) and (4), we can conclude that M-$\mathcal{CP}$ is coherent, and that the marking of $C \cup x$ corresponding to a state which is reachable from the marking of C by firing the action $\mathcal{L}(x)$. Moreover, $\mathcal{M}(\emptyset) = \varepsilon$ by definition, M-$\mathcal{CP}$ is a thus labeled event structure for (M, ε)-$\mathcal{FF}$. □

Lemma 3.3.29. *Let M-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be the M-causality process for a non-empty alphabet M. M-$\mathcal{CP}$ is finitely-branching and non-redundant.*

Proof. Thanks to the proof of Lemma 3.3.28, any configuration $C \in \mathcal{C}_{M\text{-}\mathcal{CP}}$ has $|M|$ sending extension events and at most one receiving extension event. M-$\mathcal{CP}$ is thus finitely-branching by Definition 3.1.9.

Although sending extension events of C, denoted by the set $X_!$ are pairwise in conflict, its labels and the receiving extension event's label, if exists, are pairwise different because $\mathcal{L}|_{X_!}$ is a bijection between $X_!$ and $!M$ by Definition 3.3.27. As a consequence, M-$\mathcal{CP}$ is non-redundant by Definition 3.2.14. □

FIFO channels initialized with non-empty word

Consider now a FIFO channel (M, v)-$\mathcal{FF}$ where $v \neq \varepsilon$. Intuitively, each letter m of v gives rise to only one event which corresponds to the receiving action $?m$. Due to the first-in-first-out property, such events, depending on letter m, are totally ordered. Without looking at other events, these $|v|$ events and their causality form a simple labeled event structure called a (M, v)-flushing process.

Definition 3.3.30 (v-flushing process)**.** Let $v \in M^*$ be a finite word for some given alphabet M. The (M, v)-*flushing process*, denoted by (M, v)-$\mathcal{CP}^?$, is the deterministic labeled event tree for the labeled transition system $(M^*, ?M, \rightarrow, v)$ where $\rightarrow$ is the restriction of the transition relation $\rightarrow_{\mathcal{FF}}$ in (M, v)-$\mathcal{FF}$ onto $(M^* \times ?M \times M^*)$.

Figure 3.12.a shows an example of (M, v)-flushing processes. It follows from the Definition 3.3.3 that there is no concurrency in (M, v)-$\mathcal{CP}^?$. However, one can find out that there is no conflict, so that all events are pairwise in causal.

Now, by the same manner as in causality processes for counters (see Section 3.3.2), we introduce a labeled event structure for a given (M, v)-$\mathcal{FF}$ that intuitively consists of a v-flushing process, a M-causality process, and some causality in addition.

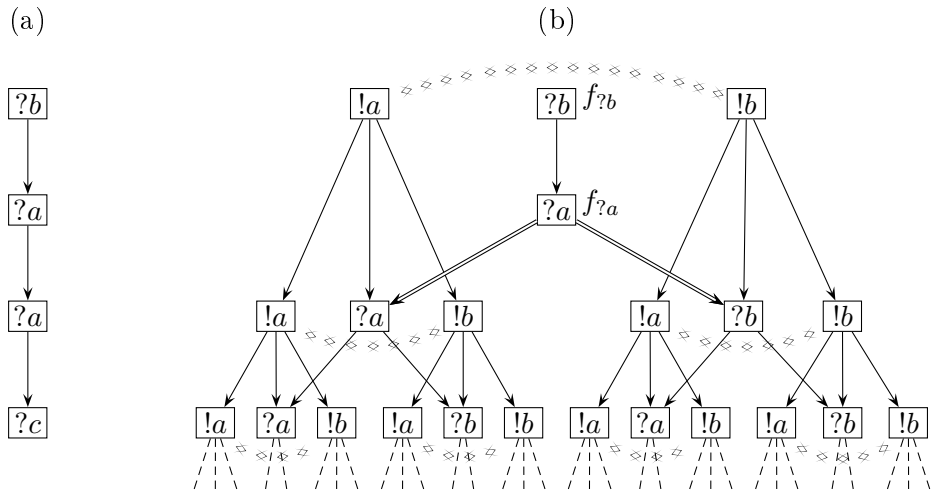


Figure 3.12: (a) (M, v)-flushing process where $M = \{a, b, c\}$ and $v = baac$; (b) (M, v)-causality process where $M = \{a, b\}$ and $v = ba$.

Definition 3.3.31. Let M be a non-empty alphabet and v be any finite word over M. The (M, v)-causality process is a labeled event structure (M, v)-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ where E is the union set of two disjoint sets E^f and E^m such that

1. (M, v)-$\mathcal{CP}|_{E^f}$ is the (M, v)-flushing process w.r.t. isomorphism,
2. (M, v)-$\mathcal{CP}|_{E^m}$ is the M-causality process w.r.t. isomorphism,
3. $\# = \#|_{E^m}$,
4. $\leq = (\leq|_{E^f}) \cup (\leq|_{E^m}) \cup (E^f \times E^m_?)$ where $E^m_?$ is the set of receiving events in (M, v)-$\mathcal{CP}|_{E^m}$.
5. the labeling function $\mathcal{L} : E \to (!M \cup ?M)$ is defined as $\mathcal{L}(e) = \mathcal{L}_{\leq|_{E^f}}(e)$ if $e \in E^f$, and $\mathcal{L}(e) = \mathcal{L}_{\leq|_{E^m}}(e)$ otherwise, and
6. the marking function $\mathcal{M} : \mathcal{C}_{M\text{-}\mathcal{CP}} \to M^*$ is defined as $\mathcal{M}(C) = (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(v.\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_!)))$ where $\sigma_!$ and $\sigma_?$ are respectively the linearisations, w.r.t. the causality $\leq$, of $C_! = \{e \in C \,/\, \mathcal{L}(e) \in !M\}$ and $C_? = \{e \in C \,/\, \mathcal{L}(e) \in ?M\}$.

Figure 3.12.b illustrates $(\{a, b\}, ba)$-causality process. Its $(\{a, b\}, ba)$-flushing process contains only two events $f_{?b}$ and $f_{?a}$ that are in the middle of the figure. In this example, $E^f = \{f_{?b}, f_{?a}\}$ and $E^m = E \setminus E^f$. As stated in the third item of Definition 3.3.31, conflict in (M, v)-$\mathcal{CP}$ is the conflict in its M-causality process. The additional causality between events in these two disjoint sets, i.e. $\leq \cap (E^f \times E^m)$, comes from the fact minimal receiving events of M-causality processes, w.r.t. $\leq$, must be direct successors of the maximal event of (M, v)-flushing process in order to respect the first-in-first-out property. Hence, this predecessor relation $\lessdot \cap (E^f \times E^m) = \mathsf{Max}_{\leq}(E^f) \times \mathsf{Min}_{\leq}(E^m_?)$ is represented by double arrows in the figure. It respects well to the third item. The marking function is similar to the one of M-causality process in Definition 3.3.27 with attention at initial word v.

Lemma 3.3.32. *Let M be a non-empty alphabet and v be any finite word over M. The (M, v)-$\mathcal{CP}$ is a coherent, non-redundant and finitely-branching labeled event structure for the (M, v)-$\mathcal{FF}$.*

Proof. Let C be any configuration of (M,v)-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$. C has exactly M extension events which are sending extension events of $C \cap E^m$ in M-$\mathcal{CP}$, and they are formally represented by the direct successor set of the only sending event $e^m_!$ where $\{e^m_!\} = \mathsf{Max}_{\leq}(C \cap E_!)$. These extension events are distinguishably labeled by sending actions in $!M$ (1) because $\mathcal{L}_{M\text{-}\mathcal{CP}}|_{\lessdot(e_{!m})}$, as a consequence $\mathcal{L}|_{\lessdot(e_{!m})}$, is a bijection by Definition 3.3.27.

First, if $E^f \subseteq C$, the E^f-suffix of (M,v)-$\mathcal{CP}$ is the M-$\mathcal{FF}$ w.r.t. isomorphism because for all configurations $X \subseteq E^m$ in this suffix, denoted by $\mathcal{E}^x = (M,v)\text{-}\mathcal{CP}|_{E \setminus E^f \setminus \#(E^f)} = (M,v)\text{-}\mathcal{CP}|_{E^m}$, one has that

$$\begin{aligned}
\mathcal{M}_{\mathcal{E}^x}(X) &= \mathcal{M}(E^f \cup X) \\
&= (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(v.\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_!))) \\
&= (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma^f_?.\sigma^x_?)))^{-1}(v.\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_!))) \\
&= ((\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma^f_?))).(\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma^x_?))))^{-1}(v.\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma^x_!))) \\
&= (v.\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma^x_?)))^{-1}(v.\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma^x_!))) \\
&= (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma^x_?)))^{-1}(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma^x_!)))
\end{aligned}$$

is the same formula as in definition of M-causality process, where $\sigma_?, \sigma_! = \sigma^x_!, \sigma^f_?$, and $\sigma^x_?$ are respectively linearisations of $(C \cap E_?), (C \cap E_!) = (X \cap E_!), E^f$, and $X \cap E_?$. Thanks to Lemma 3.3.28, C has receiving extension labeled by $?m \in !M$ if and only if m is the first letter of $\mathcal{M}(C) = \mathcal{M}_{\mathcal{E}^x}(C \setminus E^f)$, that means $X = (C \setminus E^f)$ has a receiving extension event labeled by $?m$ in the E^f-suffix $\mathcal{E}^x$.

Second, if $E^f \not\subseteq C$, let $e^f_?$ be the maximal event w.r.t. $\leq$ of E^f, and $x^f_?$ is the unique direct successor of $e^f_?$. Since all receiving events are either causal or in conflict, we have $C \cap E_? = {\geq}(e^f_?), x^f_? \in E^f \setminus C$, and $x^f_? \in E_?$. Receiving event $x^f_?$ is thus the unique receiving extension event of C; and marking $\mathcal{M}(C) = (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(v.\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_!)))$ defined in Definition 3.3.31 has a prefix $(\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(v)$ where $\sigma_?$ is the linearisation of ${\geq}(e^f_?)$ w.r.t. $\leq$. By Definition 3.3.30, the first letter of this prefix, and of $\mathcal{M}(C)$ as a consequence, is $\Pi_{?M}(x^f_?)$ (2).

Now let e be any extension event of C, i.e. $C \vdash e$. Since there is no concurrency between sending events, as well as receiving events, if $\sigma_!, \sigma_?, \sigma'_!$ and $\sigma'_?$ are respectively linearisations of $(C \cap E_!), (C \cap E_?), ((C \cup \{e\}) \cap E_!)$ and $((C \cup \{e\}) \cap E_!)$, then we have that $\sigma'_! = \sigma_!.e$ and $\sigma'_? = \sigma_?$ if $e \in E_!$, or $\sigma'_! = \sigma_!$ and $\sigma'_? = \sigma_?.e$ otherwise, i.e. $e \in E_?$. Therefore, we can deduce from the definition of the marking function that

$$\mathcal{M}(C \cup \{e\}) = \begin{cases} \mathcal{M}(C).\Pi_{!M}(\mathcal{L}(e)) & \text{if } e \in E_! \\ (\Pi_{!M}(\mathcal{L}(e)))^{-1}\mathcal{M}(C) & \text{if } e \in E_? \end{cases} \quad (3)$$

Hence (M,v)-$\mathcal{CP}$ is coherent due to (1), (2), and (3). Moreover, $\mathcal{M}(\emptyset) = v$ by definition, the induced labeled transition system $\mathcal{LTS}^{(M,v)\text{-}\mathcal{CP}}$ is the (M,v)-$\mathcal{FF}$ defined in Definition 3.3.22.

Finally, since the M-causality process is the E^f-prefix of the (M,v)-causality process. It follows from the non-redundancy and finitely-branching properties of M-causality process which are proved in Lemma 3.3.29, that (M,v)-causality process is non-redundant and finitely-branching too thanks to Corollary 3.1.17. □

Bounded FIFO channels

In practice, FIFO channels are usually finite-state systems due to the fact that channels cannot contain more than b messages for some given number b. Intuitively, when the

channel is full, i.e. contains b messages, all sending action is enabled only after some receiving one. Bounded FIFO channels are formally defined as follows:

Definition 3.3.33. Let b be any positive number, M be a non-empty alphabet, and v be a word over M whose size is not greater than b. The *FIFO channel over M which is initialized by v and is bounded by b*, denoted by (M, v, b)-$\mathcal{BF}$, is the restriction of the (M, v)-$\mathcal{FF}$ to the state set $M^{[0,b]}$.

The action set of (M, v, b)-$\mathcal{BF}$ is still $(!M \cup ?M)$ like the one of (M, v)-$\mathcal{FF}$, however, its semantics are slightly different. Since (M, v, b)-$\mathcal{BF}$ is a restriction of (M, v)-$\mathcal{FF}$ on its states, every firing sequence σ of (M, v, b)-$\mathcal{BF}$ is also a firing sequence of (M, v)-$\mathcal{FF}$. So that, if one can model the (M, v, b)-$\mathcal{BF}$ by a called (M, v, b)-*causality process* (M, v, b)-$\mathcal{CP}$ which is based on (M, v)-causality process (M, v)-$\mathcal{CP}$, then a configuration in (M, v)-$\mathcal{CP}$ must correspond to a configuration in (M, v, b)-$\mathcal{CP}$. In fact, the key idea is intuitively that one needs to add some causality to (M, v)-$\mathcal{CP}$ in order to avoid all configurations whose marking is not in the range $M^{[0,b]}$. Let us consider an example where $M = \{a, b, c\}, v = ba$ and the bound-parameter $b = 3$. Figure 3.13 illustrates a prefix of the corresponding (M, v)-$\mathcal{CP}$ in which there is no conflict. Recall that, since there is no conflict, this prefix is similar to causality processes for FIFO channels where the message set contains only one message.

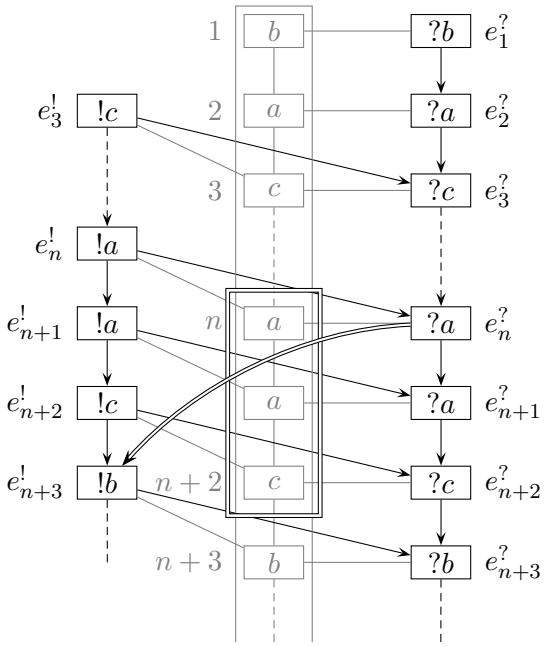


Figure 3.13: An example illustrates a (M, v) FIFO channel's content together with corresponding events in the (M, v)-$\mathcal{CP}$ where $M = \{a, b, c\}$ and $v = 3$. The double arrow is an additional causality that comes from a bounded constraint: $b = 3$.

One can find in this example many configurations C, and as a consequence, many firing sequences σ of both (M, v)-$\mathcal{FF}$ and (M, v, b)-$\mathcal{BF}$. As usual, such a firing sequence σ comes from a linearisation of some configuration C. The graphical representation of the channel's whole content, i.e. without removing message due to receiving events, i.e. receiving actions in σ, is drawn in the middle of Figure 3.13 (with gray color). All messages inserted due to the execution σ are represented in top-down order to respect

the causality of sending messages. Two messages b and a are found at the top because of the initial value $v = ab$, any other message m is the origin of a sending event $e_!$ and a receiving one $e_?$ satisfying $m = \Pi_{!M}(e_!) = \Pi_{?M}(e_?)$. For instance, $e_!^3$ and $e_?^3$ concern the first message c in the channel. Such events $e_!$ and $e_?$ are related the one to the other by the bijections $\mathcal{B}_!$ and $\mathcal{B}_?$, defined in Definition 3.3.25, of (M, v)-$\mathcal{CP}$. Moreover, consider the channel' content as a word in which messages has distinguishable indices in $\mathbb{N}$. Event $e_!$ and $e_?$ are then associated to the index of the message m in this word. This index could be defined in another way as below.

Definition 3.3.34. Given a (M, v)-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, the *depth function* $\mathcal{D} : E \to \mathbb{N}$ is defined by:

$$\mathcal{D}(e) = \begin{cases} |v| + |{\geq}(e) \cap E_!| & \text{if } e \in E_! \\ |{\geq}(e) \cap E_?| & \text{if } e \in E_? \end{cases}$$

As shown in Figure 3.13, for all $k > 0$, $\mathcal{D}(e_k^!) = \mathcal{D}(e_k^?) = k$. The $\mathcal{D}$ function is computed from the causality as stated in Definition 3.3.34. Moreover, one may use $\mathcal{D}$ together with the conflict relation to determine the bijections $\mathcal{B}_!$ and $\mathcal{B}_?$ on (M, v)-causality processes.

Lemma 3.3.35. *Given a (M, v)-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, two events $e_! \in E_!$ and $e_? \in (E_? \setminus E^f)$ are related by the bijection $\mathcal{B}_!$, as well as $\mathcal{B}_?$, if and only if $e_! \overline{\#} e_?$ and $\mathcal{D}(e_!) = \mathcal{D}(e_?)$.*

Proof. By the same manner as in the proof of Lemma 3.3.32, for the left-to-right implication, if $e_? = \mathcal{B}_!(e_!)$ then ${\geq}(e_?) = {\geq}(e_!) \cup \mathcal{B}_!({\geq}(e_!)) \cup E^f$. Hence, ${\geq}(e_?) \cap E_? = \mathcal{B}_!({\geq}(e_!)) \cup E^f$, and consequently, $\mathcal{D}(e_?) = |\mathcal{B}_!({\geq}(e_!)) \cup E^f| = |{\geq}(e_!)| + |E^f| = |{\geq}(e_!) \cap E_!| + v = \mathcal{D}(e_!)$.

For the right-to-left implication, thanks to Proposition 3.3.26, $e_! \overline{\#} e_?$ implies that the sending direct predecessor $\mathcal{B}_?(e_?)$ of $e_?$ is in causal with $e_!$. Moreover, for all sending successors $f_! \in E_!$ of $e_!$, i.e. $e_! < f_!$, it follows from $({\geq}(f_!)) \supseteq ({\geq}(e_!) \cup \{f_!\}) \supset ({\geq}(e_!))$ that $\mathcal{D}(f_!) > \mathcal{D}(e_!)$ by definition of the function $\mathcal{D}$. In a same manner for predecessors $f_! \in E_!$ of $e_!$, one obtains $\mathcal{D}(f_!) < \mathcal{D}(e_!)$. Therefore, $\mathcal{D}(e_?) = \mathcal{D}(e_!)$ implies that $\mathcal{B}_?(e_?)$ is $e_!$, i.e. $\mathcal{B}_?(e_?) = e_!$, and of course, $\mathcal{B}_!(e_!) = e_?$. □

Once again, look at Figure 3.13, the marking of a configuration C intuitively corresponds to a window on the channel's content. Such a window is limited by the indices of maximal events, w.r.t. the causality, in C. For example, if $C = \{e_3^!, \ldots, e_{n+2}^!\} \cup \{e_1^?, \ldots e_{n-1}^?\}$, its maximal events are $e_{n+2}^! \in E_!$ and $e_{n-1}^? \in E_?$, the marking of C thus consists of messages with indexes from n to $n+2$, that is graphically grouped by the double-frame in Figure 3.13, i.e. $\mathcal{M}(C) = \Pi_{!M}^{\mathcal{W}}(e_n^!.e_{n+1}^!.e_{n+2}^!) = aac$.

Aiming at defining labeled event structures for bounded FIFO channels based on (M, v)-$\mathcal{CP}$, for instance $b = 3$, we need somehow a constraint in order to disable the extension event $e_{n+3}^!$ of configuration C because $|\mathcal{M}(C)| = b$. However, $e_{n+3}^!$ is hopefully an extension event of the configuration $(C \cup \{e_n^?\})$ for respecting the fact that the bounded channel can accept a new sending action just after some receiving one. In order to do so, we add a new causality from $e_n^?$ to $e_{n+3}^!$, and by generally applying this to all pairs of a sending event $e^! \in E_!$ and a receiving $e^? \in E_?$ where $\mathcal{D}(e^!) = \mathcal{D}(e^?) + b$, one can obtain labeled event structures for (M, v, b)-bounded FIFO channels, based on (M, v)-causality processes.

Definition 3.3.36 ((M, v, b)-causality process)**.** Let b be any positive number, M be a non-empty alphabet, and v be a word over M whose size is not greater than b. Let

(M,v)-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be the (M,v)-causality process. The (M,v,b)-*causality process* is the tuple (M,v,b)-$\mathcal{CP} = (E, (\leq \cup \leq_b), \#, \mathcal{L}, \mathcal{M})$ where

$$\leq_b = \{\langle e, f\rangle \in ((E \times E) \setminus \#) \,/\, \mathcal{D}(f) \geq \mathcal{D}(e) + b\}$$

Lemma 3.3.37. *(M,v,b)-$\mathcal{CP}$ is a coherent labeled event structure for (M,v,b)-$\mathcal{BF}$.*

Proof. Let us denote $\leq' = \leq \cup \leq_b$, we first prove that $\leq'$ is a partial-order. By definition of function $\mathcal{D}$, and as a consequence of Proposition 3.3.26, we have $e \leq f$ implies that $\mathcal{D}(f)$ is greater than or equal to $\mathcal{D}(e)$. Let e, f be any two events in E. If $e \leq f$ then, for all $f' \in E$, $f \leq f'$ implies $e \leq f'$ because $(E, \leq)$ is a poset; and $f \leq_b f'$ implies $e \leq_b f'$ because $\mathcal{D}(f') \geq \mathcal{D}(f) + b \geq \mathcal{D}(e) + b$ (1). If $e \leq_b f$, i.e. $\mathcal{D}(f) \geq \mathcal{D}(e) + b$, we also have that, for all $f' \in E$, $f \leq f'$ implies $e \leq_b f'$; and $f \leq_b f'$ implies $e \leq_b f'$ (2) because $\mathcal{D}(f') \geq \mathcal{D}(f)$. From (1) and (2), $e \leq' f$ and $f \leq' f'$ implies $e \leq' f'$. Hence, $\leq'$ is transitive. Moreover, since for all $e, f \in E$, $\mathcal{D}(e) < \mathcal{D}(f)$ implies $e \neq f$, so if $e \leq' f$ and $f \leq' e$ then it is due to $e \leq f$ and $f \leq e$. As a consequence, $e = f$ because $\leq$ is a partial order. Therefore, $\leq'$ is antisymmetric, and consequently, is a partial order on E.

By the third item of Definition 3.3.25, for all $e_! \in E_!$, since $e_! \lessdot \mathcal{B}_!(e!)$, two sets $\#(e_!)$ and $\#(\mathcal{B}!(e_!))$ coincide (3). When two events $e, f \in E$ are in conflict, we have either $e \# \mathcal{B}_!(f)$ if $f \in E_!$, or $e \# \mathcal{B}_?(f)$ if $f \in E_?$. Therefore, for every event $f' \in E$ such that $f \leq_b f'$, f' must be in causality with either f or $\mathcal{B}_!(f)$ or $\mathcal{B}_?(f)$ if exists. Due to the conflict-inheritance of (M,v)-$\mathcal{CP}$, we have $e \# f'$. The conflict inheritance thus is preserved in (M,v,b)-$\mathcal{CP}$ (4).

From (3), (4), and notice that the conflict relation is the same in (M,v)-$\mathcal{CP}$ and (M,v,b)-$\mathcal{CP}$, (M,v,b)-$\mathcal{CP}$ is a labeled event structure. And moreover, $\mathcal{C}_{(M,v,b)\text{-}\mathcal{CP}} = \mathcal{C}_{(M,v)\text{-}\mathcal{CP}} \setminus \mathcal{C}'$ where $\mathcal{C}'$ is the set of configurations $C \in \mathcal{C}_{(M,v)\text{-}\mathcal{CP}}$ which are not downward-closed w.r.t. $\leq'$. Since sending events are pairwise concurrent or in conflict, and so do receiving events, we have $(\leq' \setminus \leq) = \{\langle e_!, e_?\rangle \in ((E \times E) \setminus \#) \,/\, \mathcal{L}(e_!) \in !M, \mathcal{L}(e_?) \in ?M$ and $\mathcal{D}(e_!) \geq \mathcal{D}(e_?) + b\}$, as intuitively mentioned above.

Therefore, for any $C \in \mathcal{C}_{(M,v)\text{-}\mathcal{CP}}$, C is not downward-closed w.r.t. $\leq'$ if and only if, denoted by $e_!$ the maximal event w.r.t. $\leq$ of $C \cap E_!$, $\{e_? \in (C \cap E_?) \,/\, \mathcal{D}(e_?) + b = \mathcal{D}(e_!)\} = \emptyset$; this is equivalent to $\mathcal{D}(e_!) > \mathcal{D}(f_?) + b$, where $f_?$ is the maximal event w.r.t. $\leq$ of $C \cap E_?$. Thanks to Lemma 3.3.35, we have $|\mathcal{M}(C)| > b$ by definition of a marking function. Hence, in words, $\mathcal{C}_{(M,v,b)\text{-}\mathcal{CP}}$ contains all configuration C in $\mathcal{C}_{(M,v)\text{-}\mathcal{CP}}$ whose size is less than or equal to b, i.e. $b \geq |\mathcal{M}(C)|$ or simply $\mathcal{M}(C) \in M^{[0,b]}$.

(M,v,b)-$\mathcal{CP}$ is thus coherent and is a labeled event structure of the bounded FIFO channel (M,v,b)-$\mathcal{BF}$. □

Lemma 3.3.38. *(M,v,b)-$\mathcal{CP}$ is a deterministic, finitely-branching and non-redundant labeled event structure.*

Proof. By Definition 3.3.36, (M,v,b)-$\mathcal{CP}$ differs from (M,v)-$\mathcal{CP}$ only on additional causality $\leq_b$. Hence, (M,v,b)-$\mathcal{CP}$ inherits all deterministic, finitely-branching and non-redundant properties of (M,v)-$\mathcal{CP}$ which are proved by Lemma 3.3.32. □

Figure 3.14 illustrates a causality process for bounded FIFO channel where M is a singleton. In this case, there is no difference between messages which implies the conflict relation over events. (M,v,b)-bounded FIFO channel is bisimilar with b-bounded counter which is initialized by $|v|$. The simulation relation $(\mathcal{R}_S, \mathcal{R}_\Sigma)$ could be formally defined as follows:

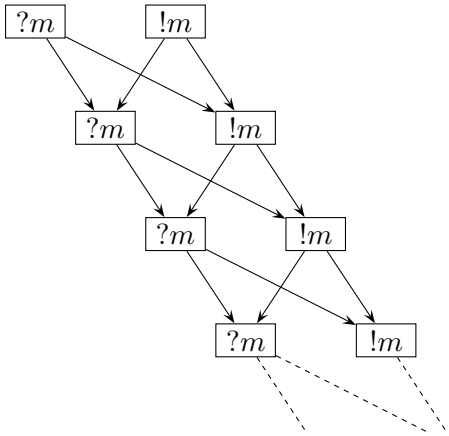


Figure 3.14: The (M, v, b)-causality process where $M = \{m\}$, $v = m$, and $b = 2$.

- $\mathcal{R}_S : M^* \to \mathbb{N}$ such that $\forall w \in M^*, \mathcal{R}_S(w) = |w|$, and
- $\mathcal{R}_\Sigma = \{\langle !m, + \rangle, \langle ?m, - \rangle\}$.

Therefore, the $(\{m\}, v, b)$-causality process, for some given word $v \in \{m\}^*$ such that $|v| \leq b$, is a deterministic, non-redundant, finitely-branching, and coherent for b-bounded counter initialized by $|v|$ (see Definition 3.3.12). Notice that in this labeled event structure, there is no conflict but all increment events are pairwise in causality, and so do decrement events.

3.3.4 Synchronized Products of Labeled Event Structures

Most of systems can be considered as concurrent systems which are composed of different components. These component systems can act in parallel and interact with each other. Interaction between components as well as simple component's actions are thus represented by a synchronization of the global system which, for example, could be modeled by synchronization vectors as explained in Section 2.4.2.

The unfolding technique [McM95a] was firstly applied to one-safe Petri nets, and then to synchronized products of transition systems [ER99], communicating finite-state machines [LI05], or high-level Petri nets [KK03]. The goal is to find compact structures modeling concurrent behaviors of such systems (systems' model). However, these complex models may be seen as synchronized product of standard systems, for example, places of Petri nets, FIFO channels, counters. Therefore, we hopefully aim at giving a general unfolding technique which computes concurrent structures for synchronized products from the ones of their components based on labeled event structures.

Definition 3.3.39 (Product of event structures)**.** Let $\mathcal{E}_i = (E_1, \leq_1, \#_1), \ldots, \mathcal{E}_n = (E_n, \leq_n, \#_n)$, be n event structures, for some given number $n \in \mathbb{N}$. A *product* of $\mathcal{E}_1, \ldots, \mathcal{E}_n$ is any quadruple $(E, \leq, \#, \mathcal{V})$, where:

1. $\leq$ is a partial order on E,
2. $\mathcal{V}$ is a function from E to $\otimes_\varepsilon(E_1, E_2, \ldots, E_n) \setminus \{\langle \varepsilon, \varepsilon, \ldots, \varepsilon \rangle\}$,
3. for all $e, e' \in E$, $e \lessdot e'$ implies that there exists $i \in \{1, 2, \ldots, n\}$ such that $\mathcal{V}(e)\downarrow_i \lessdot_i \mathcal{V}(e')\downarrow_i$,
4. for all $e, e' \in E$, $e \# e'$ iff there exists $f \leq e, f' \leq e'$ such that $f \neq f'$ and for some $i \in \{1, 2, \ldots, n\}$, we have either $\mathcal{V}(f)\downarrow_i \#_i \mathcal{V}(f')\downarrow_i$ or $\mathcal{V}(f)\downarrow_i = \mathcal{V}(f')\downarrow_i \neq \varepsilon$,

5. *not self-conflict*: for all $e \in E, e \overline{\#} e$, and
6. *componentially downward-closed*: for all $e \in E$ and for all $i \in \{1, \ldots, n\}$, $\mathcal{V}(\geq(e))\downarrow_i \setminus \{\varepsilon\}$ is a downward-closed set w.r.t. $(E_i, \leq_i)$.

Intuitively, the function $\mathcal{V}$ tells us how component events are synchronized together to obtain a global event. Although the causal relation $\leq$ and this function $\mathcal{V}$ are independently defined, the constraint stated in the third item of Definition 3.3.39 simply means that causality $\leq$ must be a consequence of some causalities in its component event structures. However, there is no way to define $\leq$ from $\mathcal{V}$ and component causalities $\leq_i$, $i \in \{1, 2, \ldots, n\}$. Because, for instance, there may exist two events that are related to a same vector v as seen in the example in the next sub-section.

But, conflict relation in a component, for instance $\#_i$, gives rise to the conflict relation $\#$ in the product. Moreover, since all configurations in an event structure are set of events, two different events which correspond to a same event in a certain component event structure must be in conflict, or in other words, they cannot both occur. As a consequence, a global event cannot be in conflict with itself due to the not-self conflict property.

The componentially downward-closed property in Definition 3.3.39 may be the most interesting one, and is the key idea for constructing synchronized product of labeled event structures (see Section 5.3). Intuitively, every global event e corresponds to n component event sets $\mathcal{V}(\geq(e))\downarrow_i$, $i \in \{1, 2, \ldots, n\}$. Although of the downward-closure of e w.r.t. $\leq$, its event sets $\mathcal{V}(\geq(e))\downarrow_i$ must be downward-closed too, w.r.t. $\leq_i$. As a consequence, for every configuration C in the product, its restriction on any component i, i.e. $\mathcal{V}(C)\downarrow_i$, is a configuration in $\mathcal{E}_i$.

Lemma 3.3.40. *Let $(E, \leq, \#, \mathcal{V})$ be a product of n event structures $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$. $\mathcal{E} = (E, \leq, \#)$ is an event structure.*

Proof. $\#$ is irreflexive due to the non-self conflict property of Definition 3.3.39, and moreover, the conflict-inheritance is guaranteed by definition of conflict relation $\#$.

Moreover, for all events $e, e' \in E$ satisfying $e < e'$, if $\varepsilon \notin \{\mathcal{V}(e)\downarrow_i, \mathcal{V}(e')\downarrow_i\}$ for some $i \in \{1, 2, \ldots, n\}$, then $\mathcal{V}(e)\downarrow_i \not\geq_i \mathcal{V}(e')\downarrow_i$ (1). Now, recall that it follows from the finitary property of component event structures $\mathcal{E}_i$ (see Definition 3.1.1) that every component event in E_i has finitely many predecessors. As a consequence, a global event $e \in E$ has also a finite number of direct predecessors. Because otherwise, i.e. $\succ(e)$ is infinite, since the number of components n is finite, there exists two direct predecessors $f, f' \in \succ(e)$ and an index $i \in \{1, 2, \ldots, n\}$ such that $\mathcal{V}(f)\downarrow_i = \mathcal{V}(f')\downarrow_i$. Hence $f \# f'$ due to the forth item in Definition 3.3.39, and it contradicts the fifth item that means e is not in conflict with itself. Therefore, $\succ(e)$ is finite for all $e \in E$ (2).

Now, suppose that there exists a global event $e \in E$ of which local configuration $\geq(e)$ is infinite. It follows from (2) that there is an infinite sequence $e = e_1 \succ e_2 \succ \ldots$ where $e_k \in \geq(e)$ for all $k \in \mathbb{N}$. Since n is finite, thanks to (1) and the third item of Definition 3.3.39, this infinite sequence contains another infinite sequence $e_{k_1}, e_{k_2}, \ldots$, where $\mathcal{V}(e_{k_1})\downarrow_i >_i \mathcal{V}(e_{k_2})\downarrow_i >_i \ldots$, for some index i. This contradicts the finitary property of the event structure $\mathcal{E}_i$ that requires the finiteness of $\geq_i(e_{k_1})$.

Therefore, $(E, \leq, \#)$ satisfies the finitary property, i.e. $\geq(e)$ is finite for all $e \in E$, and is thus an event structure. □

Graphical representation of a product of event structures

Example 3.3.41. Let $\mathcal{E}_e = (E_e, \leq_e, \#_e)$ and $\mathcal{E}_f = (E_f, \leq_f, \#_f)$ be two event structures obtained from k-bounded processes for 2-$\mathcal{BC}^0$ and 1-$\mathcal{BC}^0$ (see Definition 3.3.13 and Lemma 3.3.21). Let e_1 be any event in $\mathsf{Min}_{\leq_e}(E_e)$ and f_1 be any event in $\mathsf{Min}_{\leq_f}(E_f)$; and e_2, f_2 are respectively its direct successors. Let us define:

- $P = \{p_1, p_2, p_3, p_4, p_5\}$;
- $\leq = \{\langle p_1, p_2\rangle, \langle p_3, p_5\rangle, \langle p_4, p_5\rangle\} \cup \mathcal{I}_P$;
- $\mathcal{V} : P \to (\otimes_\varepsilon(E_e, E_f) \setminus \{\langle \varepsilon, \varepsilon\rangle\})$ where
  - $\mathcal{V}(p_1) = \langle e_1, f_1\rangle$,
  - $\mathcal{V}(p_2) = \langle e_2, f_2\rangle$,
  - $\mathcal{V}(p_3) = \langle e_1, \varepsilon\rangle$,
  - $\mathcal{V}(p_4) = \langle \varepsilon, f_1\rangle$,
  - $\mathcal{V}(p_5) = \langle e_2, f_2\rangle$; and
- $\# = (\{p_1, p_2\} \times \{p_3, p_4, p_5\}) \cup (\{p_3, p_4, p_5\} \times \{p_1, p_2\})$.

The quadruple $(P, \leq, \#, \mathcal{V})$ is thus a product of $\mathcal{E}_e$ and $\mathcal{E}_f$ by Definition 3.3.39.

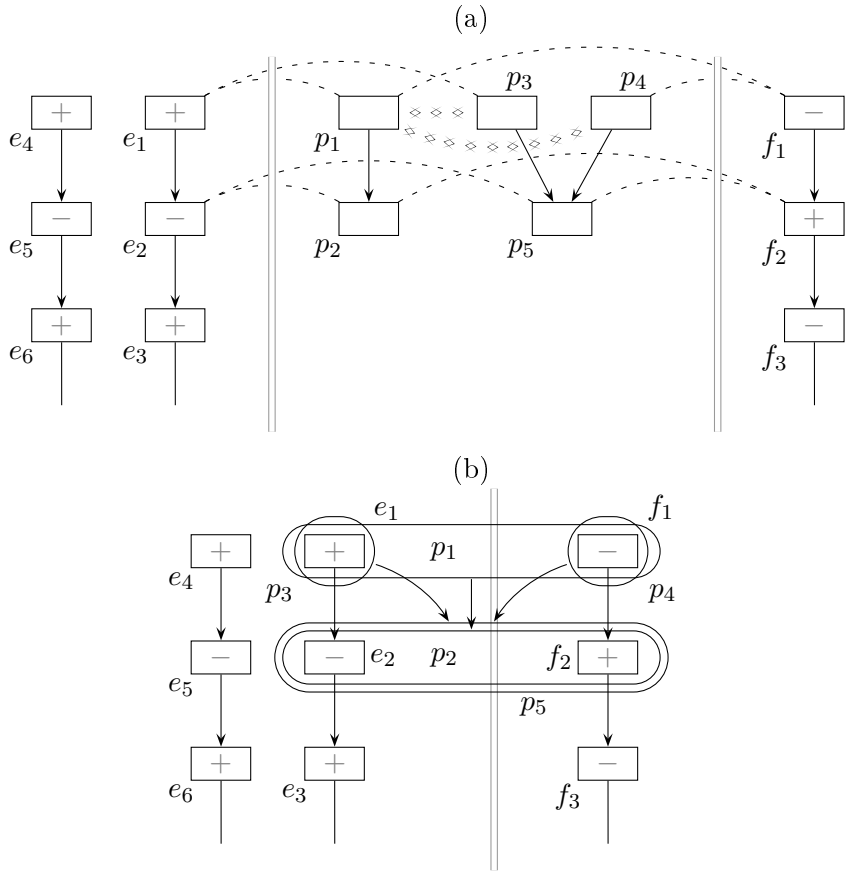


Figure 3.15: Two graphical representations of a product of event structures

Figure 3.15 illustrates the product $(P, \leq, \#, \mathcal{V})$ in Example 3.3.41, as well as its component event structures $\mathcal{E}_e, \mathcal{E}_f$ by two manners. With the first one, Figure 3.15.a, the event structure $(P, \leq, \#)$ is represented in the middle while the function $\mathcal{V}$ is illustrated by dashed arcs which tell us the relation between a global event and its related component

events. However, in this work, we prefer the second one particularly for product of more than two components, Figure 3.15.b, where global events are represented by boxes which group its corresponding component events.

It is worth noticing that the function $\mathcal{V}$ is not injective, for instance $\mathcal{V}(p_2) = \mathcal{V}(p_5)$, so that the causality $\leq$ can not be defined solely based on $\mathcal{V}$. And there is a partial-order on products of the same component event structures, like the prefix-order on event structures (see Definition 3.1.12 on page 25). Such a product $(E, <, \#, \mathcal{V})$ is a prefix of another one $(E', <', \#', \mathcal{V}')$ if $(E, \leq, \#)$ is a prefix of $(E', \leq', \#')$ w.r.t. some bijection $\mathcal{B}$ and for all $e \in E$, $\mathcal{V}(e) = \mathcal{V}'(\mathcal{B})$.

Lemma 3.3.42. *Let $\mathcal{E} = (E, \leq, \#)$ be the event structure of a product $(E, \leq, \#, \mathcal{V})$ of n given event structures $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$. For all configurations C of $\mathcal{E}$ and for all $i \in \{1, 2, \ldots, n\}$, we have $\mathcal{V}(C)\downarrow_i \setminus \{\varepsilon\}$ is a configuration of $\mathcal{E}_i$, i.e. $(\mathcal{V}(C)\downarrow_i \setminus \{\varepsilon\}) \in \mathcal{C}_{\mathcal{E}_i}$.*

Proof. Recall that $\mathcal{V}(C)\downarrow_i = \bigcup_{e \in C} (\mathcal{V}(\geq(e))\downarrow_i)$.

Since, for all $e \in E$, $(\mathcal{V}(\geq(e))\downarrow_i) \setminus \{\varepsilon\}$ is downward-closed w.r.t. $(E_i, \leq_i)$ by Definition 3.3.39, $(\mathcal{V}(C)\downarrow_i) \setminus \{\varepsilon\}$ is thus downward-closed w.r.t. $(E_i, \leq_i)$. Suppose that $\mathcal{V}(C)\downarrow_i$ is not conflict-free w.r.t. $\#_i$ for some index i. Hence, there exists two component events $e_i, f_i \in \mathcal{V}(C)\downarrow_i$ such that $e_i \#_i f_i$, and consequently, there are two corresponding event $e, f \in C$ satisfying $\mathcal{V}(e)\downarrow_i = e_i$ and $\mathcal{V}(f)\downarrow_i = f_i$. It follows from the irreflexivity of $\#_i$ that $e_i \neq f_i$. Hence e should not be equal to f. The fact that $e \# f$ and $e, f \in C$, contradicts the conflict-freeness of the configuration C. Therefore, we can conclude that $\mathcal{V}(C)\downarrow_i$ is conflict-free w.r.t. $\#_i$, and is thus a configuration in $\mathcal{E}_i$ for every index $i \in \{1, 2, \ldots, n\}$. □

Remark: For all different events e, f in a configuration C, for all index $i \in \{1, 2, \ldots, n\}$, one has either $\varepsilon \in \{\mathcal{V}(e)\downarrow_i, \mathcal{V}(f)\downarrow_i\}$ or $\mathcal{V}(e)\downarrow_i \neq \mathcal{V}(f)\downarrow_i$. Although this is not stated in Lemma 3.3.42, its proof is similar to the one of Lemma 3.3.42.

Notation 3.3.43. Given n sets $E_1, E_2, \ldots, E_n$. Let $\mathcal{L}_i, \mathcal{M}_i$, $i \in \{1, 2, \ldots, n\}$ be $2n$ functions satisfying $\mathsf{Dom}(\mathcal{L}_i) = E_i$ and $\mathsf{Dom}(\mathcal{M}_i) = \mathcal{P}(E_i)$, for all $i \in \{1, 2, \ldots, n\}$. Let $\mathcal{V}$ be any function whose co-domain is $\otimes_\varepsilon(E_1, E_2, \ldots, E_n)$. We denote:

- $\mathcal{L}_\mathcal{V}$ the function from $\mathsf{Dom}(\mathcal{V})$ to $\otimes_\varepsilon(\mathsf{Codom}(\mathcal{L}_1), \mathsf{Codom}(\mathcal{L}_2), \ldots, \mathsf{Codom}(\mathcal{L}_n))$ such that, for all $e \in \mathsf{Dom}(\mathcal{V})$ and $i \in \{1, 2, \ldots, n\}$, $\mathcal{L}_\mathcal{V}(e)\downarrow_i = \varepsilon$ if $\mathcal{V}(e)\downarrow_i = \varepsilon$, and $\mathcal{L}_\mathcal{V}(e)\downarrow_i = \mathcal{L}_i(\mathcal{V}(e)\downarrow_i)$ otherwise,
- $\mathcal{M}_\mathcal{V}$ the function from $\mathcal{P}(\mathsf{Dom}(\mathcal{V}))$ to $\otimes(\mathsf{Codom}(\mathcal{M}_1), \mathsf{Codom}(\mathcal{M}_2), \ldots, \mathsf{Codom}(\mathcal{M}_n))$ such that, for all $C \in \mathcal{P}(\mathsf{Dom}(\mathcal{V}))$, $\mathcal{M}_\mathcal{V}(C) = \mathcal{M}_1(\mathcal{V}(C)\downarrow_i \setminus \{\varepsilon\}) \times \mathcal{M}_2(\mathcal{V}(C)\downarrow_2 \setminus \{\varepsilon\}) \times \ldots \times \mathcal{M}_n(\mathcal{V}(C)\downarrow_n \setminus \{\varepsilon\})$.

Definition 3.3.44 (Synchronized product of labeled event structures)**.** Given n labeled event structures $\mathcal{E}_1 = (E_1, \leq_1, \#_1, \mathcal{L}_1, \mathcal{M}_1)$, $\mathcal{E}_2 = (E_2, \leq_2, \#_2, \mathcal{L}_2, \mathcal{M}_2), \ldots, \mathcal{E}_n = (E_n, \leq_n, \#_n, \mathcal{L}_n, \mathcal{M}_n)$. Let Σ be any subset of $\otimes_\varepsilon(\mathsf{Codom}(\mathcal{L}_1), \mathsf{Codom}(\mathcal{L}_2), \ldots, \mathsf{Codom}(\mathcal{L}_n))$. Let $(E, \leq, \#, \mathcal{V})$ be the maximal product, w.r.t. isomorphism, of n event structures $(E_1, \leq_1, \#_1)$, $(E_2, \leq_2, \#_2), \ldots, (E_n, \leq_n, \#_n)$ such that

- *synchronization*: for all $e \in E, \mathcal{L}_\mathcal{V}(e) \in \Sigma$, and
- *no-duplication*: for all $e, f \in E$, if $(>(e)) = (>(f))$ and $\mathcal{V}(e) = \mathcal{V}(f)$ then $e = f$.

The *synchronized product of labeled event structures* $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ w.r.t. Σ is the tuple $\mathcal{SP} = (E, \leq, \#, \mathcal{L}_\mathcal{V}, \mathcal{M}_\mathcal{V})$.

The event structure of $\mathcal{SP}$ contains only events satisfying the synchronization constraint. Suppose that the maximal product without duplication of $(E_1, \leq_1, \#_1), (E_2, \leq_2, \#_2), \ldots, (E_n, \leq_n, \#_n)$ is $(E', \leq', \#', \mathcal{V}')$, the event structure $\mathcal{E}_{\mathcal{SP}} = (E, \leq, \#)$ is thus the maximal prefix of $(E', \leq', \#')$ satisfying this constraint. Formally, $\mathcal{E}_{\mathcal{SP}} = (E', \leq', \#')|_E$ where $E = E' \setminus \leq(\{e \in E' / \mathcal{L}_{\mathcal{V}}(e) \notin \Sigma\})$.

However, the no-duplication property does not imply the no-redundancy in the synchronized product $\mathcal{SP}$. For instance, in the Example 3.3.41 above, due to the maximality of $\mathcal{SP}$, E must contain not only p_1 but also another event p_1' where $(>(p_1')) = (>(p_1)) = \emptyset$ and $\mathcal{V}(p_1') = \langle e_4, f_1 \rangle$ while $\mathcal{V}(p_1) = \langle e_1, f_1 \rangle$. It is obvious that $p_1 \# p_1'$ by Definition 3.3.39, but in the 2-bounded process for 2-$\mathcal{BC}^0$, $\mathcal{L}_e(e_1) = \mathcal{L}_e(e_4) = +$ and $\mathcal{M}_e(\{e_1\}) = \mathcal{M}_e(\{e_2\}) = 1$ so that $\mathcal{L}_{\mathcal{V}}(p_1) = \mathcal{L}_{\mathcal{V}}(p_1')$ and $\mathcal{M}_{\mathcal{V}}(\{p_1\}) = \mathcal{M}_{\mathcal{V}}(\{p_1'\})$. This satisfies the redundancy property stated in Definition 3.2.14. Section 6.2.2 will give more details on ways to reduce this redundancy, called *auto-redundancy* [KK03], which comes from synchronized products of concurrent labeled event structures.

Lemma 3.3.45. *The synchronized product of labeled event structures is a labeled event structure and it is finitely-branching if its component labeled event structures are all finitely-branching.*

Proof. Let $\mathcal{SP} = (E, \leq, \#, \mathcal{L}_{\mathcal{V}}, \mathcal{M}_{\mathcal{V}})$ be the synchronized product of n labeled event structures $\mathcal{E}_1 = (E_1, \leq_1, \#_1, \mathcal{L}_1, \mathcal{M}_1), \mathcal{E}_2 = (E_2, \leq_2, \#_2, \mathcal{L}_2, \mathcal{M}_2), \ldots, \mathcal{E}_n = (E_n, \leq_n, \#_n, \mathcal{L}_n, \mathcal{M}_n)$. Since $\mathcal{E}_{\mathcal{SP}} = (E, \leq, \#)$ is an event structure, $\mathcal{SP}$ is obviously a labeled event structure by definition.

Let C be any configuration in $\mathcal{E}_{\mathcal{SP}}$ and X be its set of extension events, i.e. $X = \{e \in E / C \vdash e\}$. For all $x \in X$, because $C \cup \{x\}$ is downward-closed w.r.t. $(E, \leq)$, x must be a direct successor of some event $e \in C$ or $x \in \mathsf{Min}_{\leq}(E)$. In the first case, due to the third property of Definition 3.3.39, e has at most $\prod_{i:\mathcal{V}(e)\downarrow_i \neq \varepsilon} |\lessdot_i(\mathcal{V}(e)\downarrow_i)|$ direct successors which is finite because component event structures are finitely-branching and there is no duplication in $\mathcal{SP}$. In the second case, we have that for all $i \in \{1, 2, \ldots, n\}$, $\mathcal{V}(x)\downarrow_i \in \mathsf{Min}_{\leq_i}(E_i)$, once again, the number of extension events x is finite. Therefore X is finite, and consequently, $\mathcal{SP}$ is finitely-branching. □

Theorem 3.3.46. *Given a number $n \in \mathbb{N}$ and n coherent labeled event structures $\mathcal{E}_1 = (E_1, \leq_1, \#_1, \mathcal{L}_1, \mathcal{M}_1), \mathcal{E}_2 = (E_2, \leq_2, \#_2, \mathcal{L}_2, \mathcal{M}_2), \ldots, \mathcal{E}_n = (E_n, \leq_n, \#_n, \mathcal{L}_n, \mathcal{M}_n)$. Let Σ be any subset of $\otimes_\varepsilon(\mathsf{Codom}(\mathcal{L}_1), \mathsf{Codom}(\mathcal{L}_1), \ldots, \mathsf{Codom}(\mathcal{L}_1))$, and $\mathcal{SP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be the synchronized product of $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ w.r.t. the synchronization Σ. We have that:*

- *$\mathcal{SP}$ is coherent, and*
- *if $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ are respectively labeled event structures for some n labeled transition systems $\mathcal{LTS}_1, \mathcal{LTS}_2, \ldots, \mathcal{LTS}_n$, then $\mathcal{SP}$ is the labeled event structure for the synchronized product of these labeled transition systems w.r.t. Σ.*

Proof. We will prove the two items of this theorem the order that they are stated.

- For the coherence of $\mathcal{SP}$, let C, C' be any two configurations in $\mathcal{SP}$. Let C_i and C_i' respectively denote the sets $\mathcal{V}(C)\downarrow_i \setminus \{\varepsilon\}$ and $\mathcal{V}(C)\downarrow_i \setminus \{\varepsilon\}$, for $i \in \{1, 2, \ldots, n\}$. Thanks to Lemma 3.3.42, C_i and C_i' are both configurations in $\mathcal{E}_i$.

  First, if $\mathcal{M}(C) = \mathcal{M}(C')$ then $\mathcal{M}_i(C_i) = \mathcal{M}_i(C_i')$ by Notation 3.3.43 and by Definition 3.3.44 which means that $\mathcal{M} = \mathcal{M}_{\mathcal{V}}$. Let e be any extension event of C, i.e. $C \vdash e$, and consequently, for every index i, $C_i \vdash_i \mathcal{V}(e)\downarrow_i$. It is also a

consequence of Lemma 3.3.42. Notice that $\mathcal{V}(e)\downarrow_i$ may be equal to ε. It follows from the coherence of component $\mathcal{E}_i$, more precisely the second item in Definition 3.2.11, that if $e_i \neq \varepsilon$ then there exists an extension event e'_i of C'_i satisfying $\mathcal{M}(C'_i \cup \{e'_i\}) = \mathcal{M}(C \cup \mathcal{V}(e)\downarrow_i)$ and $\mathcal{L}_i(e'_i) = \mathcal{L}_i(\mathcal{V}(e)\downarrow_i)$. Therefore, due to the maximality of the synchronized product $\mathcal{SP}$, there must exists an extension event $e' \in E$ of C', i.e. $C' \vdash e'$ such that $\mathcal{V}(e') = \langle e'_1, e'_2, \ldots, e'_n \rangle$. We have that $\mathcal{M}(C' \cup e')\downarrow_i = \mathcal{M}_i(C' \cup \{e'_i\}) = \mathcal{M}_i(C \cup \{e_i\}) = \mathcal{M}(C \cup e)\downarrow_i$ if $\mathcal{V}(e)\downarrow_i \neq \varepsilon$, and $\mathcal{M}(C' \cup e')\downarrow_i = \mathcal{M}_i(C') = \mathcal{M}_i(C) = \mathcal{M}(C \cup e)\downarrow_i$ otherwise. Hence, $\mathcal{M}(C' \cup \{e'\}) = \mathcal{M}(C \cup \{e\})$ in both case. Moreover, $\mathcal{L}(e)$ is obviously equal to $\mathcal{L}(e')$. The coherence of $\mathcal{SP}$, more precisely the second item in Definition 3.2.11, is thus proved when $\mathcal{M}(C) = \mathcal{M}(C')$.

Second, if $\mathcal{M}(C) \cap \mathcal{M}(C') \neq \emptyset$. By the same reasoning as above, for each label $a \in \Sigma$, the set of extension events e of C satisfying $\mathcal{L}(e) = a$ gives rise to the set of extension events e' of C' satisfying $\mathcal{L}(e') = a$. Due to the coherence of component labeled event structures and the maximality of synchronized product $\mathcal{SP}$, by taking the union of all marking sets, one obtains that

$$\bigcup_{e:C\vdash e,\mathcal{L}(e)=a} \mathcal{M}(C \cup e) \subseteq \bigcup_{e':C'\vdash e',\mathcal{L}(e')=a} \mathcal{M}(C' \cup e')$$

and reversely,

$$\bigcup_{e:C\vdash e,\mathcal{L}(e)=a} \mathcal{M}(C \cup e) \supseteq \bigcup_{e':C'\vdash e',\mathcal{L}(e')=a} \mathcal{M}(C' \cup e')$$

These two marking set are thus equal, so that $\mathcal{SP}$ is coherent by the first item in Definition 3.2.11.

- Let us denote $s_1, s_2, \ldots, s_n$ respectively the initial states of $\mathcal{LTS}_1, \mathcal{LTS}_2, \ldots, \mathcal{LTS}_n$, and $\mathcal{LTS}_\Sigma$ the synchronized product of these labeled event structures w.r.t. Σ. By Definition 3.3.44, we have that for all $e \in E, \mathcal{L}(e) \in \Sigma$. Moreover, it follows from the definition of function marking $\mathcal{M}$ that $\langle s_1, s_2, \ldots, s_n \rangle \in \mathcal{M}(\emptyset)$. In other words, $\mathcal{M}(\emptyset)$ contains the initial state of $\mathcal{LTS}_\Sigma$. Hence, in order to prove the second property of this theorem, one only needs to show that the extension relation in $\mathcal{E}$ corresponds to the transition relation $\rightarrow$ of $\mathcal{LTS}_\Sigma$. That means $s \xrightarrow{a} s'$ is a transition in $\mathcal{LTS}_\Sigma$ iff there exist a configuration $C \in \mathcal{C}_\mathcal{E}$ and an extension e of C such that $s \in \mathcal{M}(C), s' \in \mathcal{M}(C \cup \{e\})$ and $\mathcal{L}(e) = a$ (1). It follows from the coherency of $\mathcal{E}$ as well as of its components that $C \vdash e$ for some event e where $\mathcal{L}(e) = a$, then for all i, let $C_i = \mathcal{V}(C)\downarrow_i \setminus \{\varepsilon\}$, we have that:

$$\begin{aligned}
\bigcup_{e':C\vdash e',\mathcal{L}(e')=a} \mathcal{M}(C \cup e') &= \bigotimes_i \left(\left(\bigcup_{e':C\vdash e',\mathcal{L}(e')=a} \mathcal{M}(C \cup e') \right)\downarrow_i \right) \\
&= \bigotimes_i \left(\bigcup_{e'_i:C_i\vdash e'_i,\mathcal{L}_i(e'_i)=a\downarrow_i} \mathcal{M}_i(C_i \cup \{e'_i\} \setminus \{\varepsilon\}) \right) \\
&= \bigotimes_i \left(\bigcup_{s_i:s_i\in\mathcal{M}(C_i)} \left(\bigcup_{s'_i:\langle s_i,a\downarrow_i,s'_i\rangle\in\rightarrow_i} s'_i \right) \right) \\
&= \bigcup_{s\in\mathcal{M}(C)} \left(\bigcup_{s':\langle s,a,s'\rangle\in\rightarrow} s' \right)
\end{aligned}$$

Therefore, the right-to-left implication of (1) is obvious. By definition, transition relation in a synchronized product of labeled transition systems is based on the synchronisation Σ and the transition relations of its components (see Definition 2.4.12). It follows from the maximality of $\mathcal{E}$ (Definition 3.3.44) that the left-to-right implication of (1) is also true. Hence, $\mathcal{E}$ is a labeled event structure for $\mathfrak{LTS}_\Sigma$, or in other words, $\mathfrak{LTS}_\Sigma$ is an induced labeled transition systems of $\mathcal{E}$.

□

Chapter 4

Truncation for well-preordered labeled event structures

Contents

A labeled event structure is infinite as soon as the underlying system has an infinite execution. Thus, we need property-preserving truncation techniques in order to decide verification problems using only a finite prefix of an event structure. *Well-structured transition systems* were introduced in [Fin91, AJ93, AJ94, AČJ00] as an abstract generalization of Petri nets satisfying the same *monotocity* property, and hence enjoying nice decidability properties. It turns out that many classes of infinite-state systems are well-structured [FS01]. The application to labeled event structures of this result is detailed in Section 4.1.

In Section 4.2, we will show that the well-known truncation techniques [McM95a, ERV96, CGP01, DJN04] for safe Petri nets are also successful for well-preordered labeled event structures. Hence, one can verify different problems on infinite systems as explained in Section 4.3.

4.1 Well-preordered systems

A *preordered system* intuitively consists of a (infinite) system as well as a preorder $\preccurlyeq$ on the system's state space and a *compatibility* property on the system's transition relation. Its formal definition could be found in [FS01]. In fact, this definition is the same as our Definition 4.1.5 when one does not deal with actions/labels as remarked in Section 2.4. Figure 4.1 illustrates a compatibility which tells that if v is a reachable from a state s, i.e. $s \longrightarrow\!\!\!\!\!\rightarrow v$, then from any state s' satisfying $s \preccurlyeq s'$, one can obtain a state v' (may be possibly s'), i.e. $s' \longrightarrow\!\!\!\!\!\rightarrow v'$, such that $v \preccurlyeq v'$. We can say that the preorder $\preccurlyeq$ is preserved by the transition relation.

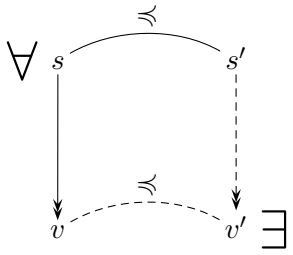


Figure 4.1: Compatibility

Remark: One can see in other works the word *quasi-order* that is also common for preorders. In this work, we use the same terminology as in [HST07] so that we prefer the word preorder instead of quasi-order, and as a consequence, prefer *well-preordered* transition systems, and further *well-preordered labeled event structures* (*cf.* Definitions Definition 4.1.5 and Definition 4.2.1) to *well-structured* ones which are more standard and firstly given in [Fin87]. The reason is that we would like to avoid the confusion between the "well-structured" property over states by means of compatibility and the structure over events determined by causality and conflict relations in labeled event structures (see Section 4.2.1).

Example 4.1.1. Since "less than or equal to" $\leq$ is a preorder over natural numbers $\mathbb{N}$, counters (see Section 3.3.2) are preordered systems in which there are only increment and decrement actions.

4.1.1 Adapting preordered compatibility to labeled transitions

Our presentation of preordered systems differs from the standard (non-labeled) one as we need to take care of labels. However, our definition is sufficiently general so that all results from standard preordered systems may be found in a same way with a little tuning. Before giving this formal definition in Section 4.1.2, let us take an example to clarify its intuitive idea.

An example: Lossy FIFO channels

Nowadays, *lossy FIFO channels* [AJ94] are widely used for modeling communicating systems and verifying communicating protocols. They differs from the FIFO channels (see Definition 3.3.22 on page 43) on the possibility of loosing messages: channel's content which is a finite word may loose some letter at any moment and become a *subword* (see Definition 2.2.1 on page 12) of the old one.

Definition 4.1.2 (v-initialized lossy FIFO channel over M)**.** Let M be a non-empty alphabet and v be any finite word over M. The *v-initialized lossy FIFO channel over M* is the labeled transition system (M, v)-$\mathcal{LF} = (M^*, \Sigma, \rightarrow, v)$ where

- the action set Σ is $\{\tau\} \cup !M \cup ?M$, and
- the transition relation

$$\begin{aligned} \rightarrow = & \{\langle w, \tau, w' \rangle / m \in M, w, w' \in M^* \text{ and } w' \preccurlyeq w\} \\ & \cup \{\langle w, !m, w.m \rangle / m \in M, w \in M^*\} \\ & \cup \{\langle m.w, ?m, w \rangle / m \in M, w \in M^*\} \end{aligned}$$

  where $\preccurlyeq$ is the subword order over M^*.

Definition 4.1.2 is the same as Definition 3.3.22 except for action τ, called a *lossy action*, as well as its related transitions $\rightarrow \cap (M^* \times \{\tau\} \times M^*)$. Let $s, s' \in M^*$ be any two states of (M, v)-$\mathcal{LF}$ satisfying $s \preccurlyeq s'$, we have $s' \xrightarrow{\tau}{}^{*} s$. Hence, all reachable states from s are also reachable states from s'. Lossy FIFO channels form obviously a preordered system.

Internal actions Σ^τ

One can say that the lossy action τ gives lossy FIFO channels its preordered compatibility, and moreover this fact can be found in many other preordered systems. In the following definition of preordered labeled transition systems till the end, we assume that each set of actions Σ is partitioned into a set Σ^τ of *internal actions* and a set Σ^γ of normal actions. This introduction of internal actions Σ^τ, on the one hand, is to tackle the problem when generalizing preordered compatibility for transitions with label by means of actions, and on the other hand, allows to clearly describe the characteristics of preordered labeled transition systems in the point of view of preordered properties.

Moreover, it is worth noticing that one can somehow eliminate internal actions Σ^τ while modeling preordered systems by labeled transition systems. For example, by considering each sending action $!m$ (each receiving action $?m$) of a lossy FIFO channel as a composed action in which the channel firstly loses some messages, then executes the sending action $!m$ (the receiving action $?m$ respectively), and finally loses some other messages; one can obtain another model for lossy FIFO channel without loss of generality, as the following:

Definition 4.1.3 (v-initialized lossy FIFO channel over M without Σ^τ)**.** Let M be a non-empty alphabet and v be any finite word over M. The *v-initialized lossy FIFO channel over M without Σ^τ* is a labeled transition system (M, v)-$\mathcal{LF} = (M^*, \Sigma, \rightarrow, v)$ where

- the action set Σ is $\{!m / m \in M\} \cup \{?m / m \in M\}$, and
- the transition relation $\rightarrow$ is $\{\langle w, !m, w' \rangle \in M^* \times !M \times M^* / \exists v \in M^* : v \preccurlyeq w, w' \preccurlyeq v.m\} \cup \{\langle w, ?m, w' \rangle \in M^* \times !M \times M^* / \exists v \in M^* : m.v \preccurlyeq w, w' \preccurlyeq v\}$ where $\preccurlyeq$ is the subword order over M^*.

4.1.2 Well-preordered labeled transition systems

Recall that ε is the "do nothing" action that is particularly used in synchronized products of labeled transition systems. ε is also the empty firing sequence for all labeled transition systems, and, in which $s \xrightarrow{\varepsilon} s$ for every state s. In the following definition and afterwards, we assume that the internal action set Σ^τ does not contain ε, i.e. $\varepsilon \notin \Sigma^\tau$.

Definition 4.1.4 (Compatibilities)**.** Let $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ be a labeled transition system and $\preccurlyeq$ be a preorder on S. We say that $\preccurlyeq$ is *compatible* (resp. *transitively compatible, reflexively compatible*) with the transition relation $\rightarrow$ if for every transition $s \xrightarrow{a} v$ and $s \preccurlyeq s'$ there exists $v \preccurlyeq v'$ such that $s' \xrightarrow{\sigma} v'$ for some $\sigma \in \Sigma^*$ satisfying:

- *compatibility*: $\sigma \in (\Sigma^\tau)^*$ if $a \in \Sigma^\tau$ and $\sigma \in (\Sigma^\tau)^*.a.(\Sigma^\tau)^*$ otherwise, or
- *transitive compatibility*: $\sigma \in (\Sigma^\tau)^+$ if $a \in \Sigma^\tau$ and $\sigma \in (\Sigma^\tau)^*.a.(\Sigma^\tau)^*$ otherwise, or
- *reflexive compatibility*: $\sigma \in (\{\varepsilon\} \cup \Sigma^\tau)$ if $a \in \Sigma^\tau$ and $\sigma = a$ otherwise.

One can say that $\preccurlyeq$ is also compatible with the transitive closure $\longrightarrow\!\!\!\!\!\rightarrow$ of the transition relation $\rightarrow$ as a consequence of Definition 4.1.4. That means if $s \xrightarrow{\sigma}\!\!\!\!\!\rightarrow v$ and $s \preccurlyeq s'$ then there exists $v' \in S$ and $\sigma' \in \Sigma^*$ such that $s' \xrightarrow{\sigma'}\!\!\!\!\!\rightarrow v'$ (proof by induction in the length of σ). And σ' must not be shorter than σ only in the case of transitive compatibility. Moreover, the reflexive compatibility induces that the longest subwords of σ and σ' which contain only normal actions Σ^γ are the same.

A preorder $\preccurlyeq$ is *strictly compatible* with $\rightarrow$ if both $\preccurlyeq$ and $\prec$ are compatible with $\rightarrow$ (recall that $s \prec s'$ is defined by $s \preccurlyeq s' \not\preccurlyeq s$). Of course, this strictness notion may be combined with transitive and reflexive compatibilities.

Remark: Definition 4.1.4 as well as the definition of strict compatibility coincide with the definitions for systems without labeled actions of Finkel *et al.* given in [FS01] when $\Sigma = \Sigma^\tau$ is a singleton. Moreover, their only definition for labeled transition system corresponds to our reflexive compatibility when there is no internal action, i.e. $\Sigma^\tau = \emptyset$. In this case, we say that $\preccurlyeq$ is *strongly compatible* with transition relation $\rightarrow$. Lossy FIFO channels in Definition 4.1.3 are examples of this compatibility.

A class of infinite systems with decidability results

Although there is compatibility between preorder $\preccurlyeq$ on states and transition relation $\rightarrow$, decidability results for such infinite systems must rely on the existence of a well-preorder property of $\preccurlyeq$ (see Definition 2.3.1 on page 13).

Definition 4.1.5 (Well-preordered labeled transition systems)**.** A *well-preordered labeled transition system* $(\mathcal{LTS}, \preccurlyeq)$ consists of a labeled transition system $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ and a preorder $\preccurlyeq$ on S satisfying:

- *well-preorder*: $\preccurlyeq$ is well-preorder or converse well-preorder on S, and
- *compatibility*: $\preccurlyeq$ is compatible with $\rightarrow$.

Example 4.1.6. Since the "less than or equal to" order $\leq$ is well-founded on $\mathbb{N}$, and is thus well-preordered, as a consequence of Example 4.1.1, counters are well-preordered labeled transition systems. One can also say that lossy FIFO channels (Definition 4.1.2) are well-preordered labeled transition systems. Because, the subword order over M is well-founded and well-preordered for all finite alphabet M (*cf.* Higman's lemma).

In [FS01], Finkel *et al.* have given a classification of well-known systems into family of well-structured transition systems as well as its decidable problems which depends on the type of compatibility. Notice that their downward well-structure transition systems correspond to our well-preordered labeled transition systems in which the preorder on states is converse well-preorder (see Definition 2.3.1).

Our definition of well-preorder labeled transition systems is enough general so that all decidability results in [FS01] are still valid. The goal of the next sections is not to prove these results again but to essentially show how to efficiently verify decidable problems on a partial-order structure, more precisely, on labeled event structures.

Synchronized products of well-preordered labeled transition systems

Definition 4.1.7 (Product preorder)**.** Let $\preccurlyeq_1, \preccurlyeq_2, \ldots, \preccurlyeq_n$ be n preorders on n sets $X_1, X_2, \ldots, X_n$ respectively. The *product preorder* of these n preorders is a binary relation, denoted by $\preccurlyeq_\otimes$, on the n-dimension space $\otimes(X_1, X_2, \ldots, X_n)$, and is defined by: for all $x, x' \in \otimes(X_1, X_2, \ldots, X_n)$, $x \preccurlyeq_\otimes x'$ iff $x_i \preccurlyeq_i x'_i$ for every $1 \leq i \leq n$.

Recall that $x_i = x{\downarrow_i}$ is the component restriction onto i of x. We also write $\preccurlyeq_\otimes = \langle \preccurlyeq_1, \preccurlyeq_2, \ldots, \preccurlyeq_n \rangle$ and naturally mean that $\preccurlyeq_i$ is the component restriction onto i of $\preccurlyeq_\otimes$. The product preorder is also a preorder like its name, and moreover, the well-preordered property of its component, if exists, is preserved.

Lemma 4.1.8. *The product preorder $\preccurlyeq_\otimes$ of n preorders $\preccurlyeq_1, \preccurlyeq_2, \ldots, \preccurlyeq_n$, is a preorder and is well-preordered (converse well-preordered) if $\preccurlyeq_i$ is well-preordered (converse well-preordered resp.) for all $1 \leq i \leq n$.*

Proof. By Definition 4.1.7, $\preccurlyeq_\otimes$ is reflexive and transitive binary relation, and is thus a preorder. We will prove that $\preccurlyeq_\otimes$ is well-preordered on X by induction on n where $X = \otimes(X_1, X2, \ldots, X_n)$. When $n = 1$, it is straightforward. Suppose that it is true for some given k, i.e. the product preorder $\preccurlyeq'_k$ of k well-preorders $\preccurlyeq_1, \preccurlyeq_2, \ldots, \preccurlyeq_k$ is a well-preorder on X. Let $x_0, x_1, \ldots$, be any infinite sequence. Thanks to Erdös and Rado's lemma, it says that this sequence contains an infinite increasing subsequence $x_{i_0} \preccurlyeq'_k x_{i_1} \preccurlyeq'_k \ldots$, due to the well-preorder $\preccurlyeq'_k$. Further, it follows from the well-preorder $\preccurlyeq_{k+1}$ and this second infinite sequence that there exist two indices $l < m$ satisfying $x_{i_l} \preccurlyeq_{k+1} x_{i_m}$. Hence, one obtains both $x_{i_l} \preccurlyeq'_k x_{i_m}$ and $x_{i_l} \preccurlyeq_{k+1} x_{i_m}$, and consequently, $x_{i_l} \preccurlyeq'_{k+1} x_{i_m}$ where $\preccurlyeq'_{k+1}$ is the product preorder of $k+1$ well-preorders $\preccurlyeq_1, \preccurlyeq_2, \ldots, \preccurlyeq_{k+1}$. In other words, the induction hypothesis is also true for $k+1$. One can conclude that $\preccurlyeq_\otimes$ is a well-preorder for any finite number n. □

The set of internal actions Σ^τ previously introduced, not only gives the compatibility of preordered systems but also separates internal transitions and synchronized ones of synchronized products of labeled transition systems. We assume (1) that every synchronization constraint $\Sigma_\otimes$ implicitly contains the set $\Sigma^\tau_\otimes$ of synchronization vectors, defined as follows:

$$\begin{aligned}\Sigma^\tau_\otimes = &\{\langle \tau_1, \varepsilon, \ldots, \varepsilon \rangle / \tau_1 \in \Sigma^\tau_1\} \cup \ldots \\ &\cup \{\langle \varepsilon, \ldots, \varepsilon, \tau_i, \varepsilon, \ldots, \varepsilon \rangle / \tau_i \in \Sigma^\tau_i\} \cup \ldots \\ &\cup \{\langle \varepsilon, \ldots, \varepsilon, \tau_n \rangle / \tau_n \in \Sigma^\tau_n\}\end{aligned}$$

and (2) that no internal action $\tau_i \in \Sigma^\tau_i$ may appear in a synchronization vector of $\Sigma_\otimes \backslash \Sigma^\tau_\otimes$, i.e. for all $i \in \{1, 2, \ldots, n\}$: $(\Sigma_\otimes \setminus \Sigma^\tau_\otimes){\downarrow_i} \cap \Sigma^\tau_i = \emptyset$. Naturally, $\Sigma^\tau_\otimes$ is a subset of local actions of any synchronized product w.r.t. $\Sigma_\otimes$.

Definition 4.1.9. The *synchronized product of n preordered labeled transitions* $(\mathcal{LTS}_1, \preccurlyeq_1), (\mathcal{LTS}_2, \preccurlyeq_2), \ldots, (\mathcal{LTS}_n, \preccurlyeq_n)$ w.r.t. some synchronization constraint $\Sigma_\otimes \in \otimes_\varepsilon(\Sigma_1, \Sigma_2, \ldots \Sigma_n)$ is the synchronized product $\mathcal{LTS}_\otimes$ of $\mathcal{LTS}_1, \mathcal{LTS}_2, \ldots, \mathcal{LTS}_n$ w.r.t. $\Sigma_\otimes$ equipped with the product preorder $\preccurlyeq_\otimes = \otimes(\preccurlyeq_1, \preccurlyeq_2, \ldots, \preccurlyeq_n)$; and is denoted by $(\mathcal{LTS}_\otimes, \preccurlyeq_\otimes)$.

The following lemma shows that all compatibility notions defined above for preordered labeled transition systems are preserved under synchronized product.

Lemma 4.1.10. *Let* Cond *denote any compatibility condition among {(non-strict), strict} × {(standard), transitive, reflexive}. Any synchronized product of preordered labeled transition systems with compatibility* Cond *also has compatibility* Cond.

Proof. Consider n preordered labeled transition systems $(\mathcal{LTS}_1, \preccurlyeq_1), (\mathcal{LTS}_2, \preccurlyeq_2), \ldots, (\mathcal{LTS}_n, \preccurlyeq_n)$ where $\mathcal{LTS}_i = (S_i, \Sigma_i, s_i^0, \rightarrow_i)$; and assume that each $\mathcal{LTS}_i$ has compatibility Cond. We show that $(\mathcal{LTS}_\otimes, \preccurlyeq_\otimes)$ has compatibility Cond, where $\mathcal{LTS}_\otimes$ is the synchronized product of $\mathcal{LTS}_1, \mathcal{LTS}_2, \ldots, \mathcal{LTS}_n$ w.r.t. a given synchronization constraint $\Sigma_\otimes$. Let $s \xrightarrow{v}_\otimes s'$ be any transition in $\mathcal{LTS}_\otimes$, let t be any state such that $s \preccurlyeq_\otimes t$. There are two cases, depending on whether the action $v \in \Sigma_\otimes$ contains an internal action:

1. $v \in \Sigma_\otimes^\tau$, that means $v_j \in \Sigma_j^\tau$ for some $1 \leq j \leq n$ and $v_i = \varepsilon$ for all $i \neq j$. It follows from compatibility of $(\mathcal{LTS}_j, \preccurlyeq_j)$ that there exists a path $\pi_j = t_j \xrightarrow{v^1}_j u^2 \xrightarrow{v^2}_j u^3 \ldots \xrightarrow{v^k}_j u^{k+1}$ where $u^{k+1} \succcurlyeq_j s'_j$ and for all $i \in \{1, 2, \ldots, k\}, v^i \in \Sigma_j^\tau$ ([1]). This path may be extended to the synchronized product $\mathcal{LTS}_\otimes$ as $\pi = t \xrightarrow{v^1}_\otimes t^2 \xrightarrow{v^2}_\otimes t^3 \ldots \xrightarrow{v^k}_\otimes t^{k+1}$ where $v^h = \langle \varepsilon, \ldots, \varepsilon, v^i, \varepsilon, \ldots, \varepsilon \rangle, t_i^h = u^h$ and $t_i^h = t_i$ for all $i \neq j$. Observe that $s' \preccurlyeq t^{k+1}$ because $s'_i = s_i \preccurlyeq t_i$ for all $i \neq j$.

2. $v \in \Sigma_\otimes^\gamma$, that means $v_i \in \{\varepsilon\} \cup \Sigma_i^\gamma$ for all $1 \leq i \leq n$. We may assume without loss of generality that there exists $1 \leq m \leq n$ such that $v_i \in \Sigma_i^\gamma$ for all $i \leq m$ and $v_i = \varepsilon$ for all $m < i$. From compatibility of $(\mathcal{LTS}_i, \preccurlyeq_i)$ with $i \leq m$, we obtain that there exists m paths $\pi_i = t_i \xrightarrow{\sigma^i}\!\!\!\!\rightarrow_i u^i \xrightarrow{v_i}_i u'^i \xrightarrow{\sigma'^i}\!\!\!\!\rightarrow_i t'_i$ with $t'_i \succcurlyeq_i s'_i$ and $\sigma^i, \sigma'^i \in (\Sigma_i^\tau)^*$. Remark that $u \xrightarrow{v}_\otimes u'$ where $u_i = u^i$ and $u'_i = u'^i$ for all $i \leq m$, and $u_i = u'_i = t_i$ otherwise. As in the previous case, we may extend each sub-path $t_i \xrightarrow{\sigma^i}_i u_i$ and $u'_i \xrightarrow{\sigma'^i}_i t'_i$ to the synchronized product, and their concatenations yields a path $\pi = t \xrightarrow{\sigma_\otimes}_\otimes u \xrightarrow{v}_\otimes u' \xrightarrow{\sigma'_\otimes}_\otimes t'$ where $\sigma_\otimes, \sigma'_\otimes \in (\Sigma_\otimes^\tau)^*$, $t'_i = t'^i$ for all $1 \leq i \leq m$ and $t'_i = t_i$ otherwise. Observe that $t' \succcurlyeq_\otimes s'$ since $s'_i = s_i \preccurlyeq_\otimes t_i$ for all $i > m$.

Thus we obtain that there exists $t' \succcurlyeq s'$ and $\sigma \in (\Sigma_\otimes)^*$ such that $t \xrightarrow{\sigma}\!\!\!\!\rightarrow_\otimes t'$. Moreover, a routine check shows that in both cases, the constructed path $t \xrightarrow{\sigma}\!\!\!\!\rightarrow_\otimes t'$ satisfies Cond's requirements (for strict compatibility, the component path(s) should be carefully chosen so as to ensure strictness). □

A direct consequence of Lemma 4.1.8 and Lemma 4.1.10 is the following:

Lemma 4.1.11. *Synchronized product $(\mathcal{LTS}_\otimes, \preccurlyeq_\otimes)$ of n well-preordered labeled transition systems $(\mathcal{LTS}_1, \preccurlyeq_1), (\mathcal{LTS}_2, \preccurlyeq_2), \ldots, (\mathcal{LTS}_n, \preccurlyeq_n)$ is a well-preordered labeled transition system if $\preccurlyeq_1, \preccurlyeq_2, \ldots, \preccurlyeq_n$ are either all well-preorders or all converse well-preorders.*

4.1.3 From forward analysis to backward analysis in well-preordered transition systems

In this section, we show how to embed, in our forward partial-order analysis approach lately detailed (see Section 4.3), standard backward analysis techniques (called *set saturation methods* in [FS01]) for well-preordered transition systems. This idea is based on duality in the category of (labeled) transition systems.

Definition 4.1.12. The *dual* of a given labeled transition system $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow_{\mathcal{LTS}})$ is the labeled transition system $\mathcal{DTS} = (S, \Sigma, s^0, \rightarrow_{\mathcal{DTS}})$ such that $\langle s, a, s' \rangle \in \rightarrow_{\mathcal{DTS}}$ iff $\langle s', a, s \rangle \in \rightarrow_{\mathcal{LTS}}$.

[1] We use superscript indexing in addition to avoid confusing it with the component projection ($x_i = x\downarrow_i$).

The dual of a lossy FIFO channel $\mathcal{LTS}$ in Definition 4.1.2 is a labeled transition system $\mathcal{DTS}$ modeling a well-known *insertion-error FIFO channel* in which actions are all renaming such that $!m$ becomes $?m$ and conversely. The internal action τ allows to insert message into FIFO's content at any moment. More interestingly, since $(\mathcal{LTS}, \preccurlyeq)$ is a well-preordered labeled transition system, $(\mathcal{DTS}, \succcurlyeq)$ is too. However, $\preccurlyeq$ is well-preordered while $\succcurlyeq$ is converse well-preordered, and vice versa.

Notice that initial states of dual systems are not important for the reachability/covering problem in which one only needs to know if from a state s, some state v is reachable or not. By duality, $s \twoheadrightarrow_{\mathcal{LTS}} v$ if and only if $v \twoheadrightarrow_{\mathcal{DTS}} s$. Intuitively, with backward analysis technique, one firstly computes the set of states, denoted by $\mathsf{pre}^*(v)$ from which we can reach v, and then test if $\mathsf{pre}^*(v)$ contains s. With forward analysis, one computes the set $\mathsf{post}^*(s)$ of reachable states from s and then verifies if it contains v. Figure 4.2 illustrates these two approaches.

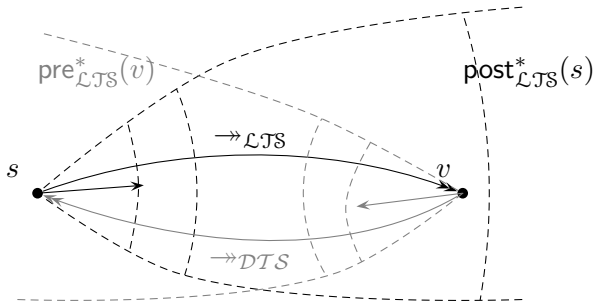


Figure 4.2: Forward and backward analysis for reachability

We now need a few additional notations in this section. Consider any labeled transition system $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow_{\mathcal{LTS}})$. The *one-step reachability relation* is the binary relation $\mathcal{R}_{\mathcal{LTS}}$ on S defined by $s \, \mathcal{R}_{\mathcal{LTS}} \, s'$ iff $s \xrightarrow{a}_{\mathcal{LTS}} s'$ for some action $a \in \Sigma$. By Definition 4.1.12, we have thus $\mathcal{R}_{\mathcal{DTS}} = \mathfrak{R}_{\mathcal{LTS}}$ where $\mathcal{R}_{\mathcal{DTS}}$ is the one-step reachability relation of the dual labeled transition system $\mathcal{DTS}$ of $\mathcal{LTS}$.

We will use the following backward/forward (reachability) set transformers: for any subset $X \subseteq S$, we define $\mathsf{post}_{\mathcal{LTS}}(X) = \mathcal{R}_{\mathcal{LTS}}(X)$, $\mathsf{pre}_{\mathcal{LTS}}(X) = \mathfrak{R}_{\mathcal{LTS}}(X)$, $\mathsf{post}^*_{\mathcal{LTS}}(X) = \mathcal{R}^*_{\mathcal{LTS}}(X)$, and $\mathsf{pre}^*_{\mathcal{LTS}}(X) = \mathfrak{R}^*_{\mathcal{LTS}}(X)$. Observe that the reachability set $\mathsf{post}^*_{\mathcal{LTS}}$ of $\mathcal{LTS}$ defined in Section 2.4.1 is equal to $\mathsf{post}^*_{\mathcal{LTS}}(s^0)$. For any label $a \in \Sigma$ and subset $X \subseteq S$, we define $\mathsf{pre}_{\mathcal{LTS}}(a, X) = \{s \in S \,/\, \exists s' \in X, s \xrightarrow{a}_{\mathcal{LTS}} s'\}$. Remark that we have, for every $X \subseteq S$, $\mathsf{pre}_{\mathcal{LTS}}(X) = \bigcup_{a \in \Sigma} \mathsf{pre}_{\mathcal{LTS}}(a, X)$.

Backward analysis for well-preordered systems is performed by a classical fix-point computation of $\mathsf{pre}^*(\preccurlyeq(X))$ in which one applies an upward closure w.r.t. $\preccurlyeq$ at each step. It requires a so called finite pred-basis condition [ACJT00, FS01].

Definition 4.1.13 (Finite pred-basis)**.** Given a preordered labeled transition system $(\mathcal{LTS}, \preccurlyeq)$ with $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow_{\mathcal{LTS}})$. A *finite pred-basis* for $(\mathcal{LTS}, \preccurlyeq)$ is any function pb from $\Sigma \times S$ to $\mathcal{P}_f(S)$ satisfying:

- $\bigcup_{a \in \Sigma} \mathsf{pb}(a, s)$ is finite for all $s \in S$, and
- $\preccurlyeq(\mathsf{pb}(a, s)) = \preccurlyeq(\mathsf{pre}_{\mathcal{LTS}}(a, \preccurlyeq(\{s\}))$ for all $a \in \Sigma$ and $s \in S$.

Given any finite pred-basis pb for $(\mathcal{LTS}, \preccurlyeq)$, we define the *pb-reverse* of $(\mathcal{LTS}, \preccurlyeq, \mathsf{pb})$ as the labeled transition system $\mathcal{R}(\mathcal{LTS}, \preccurlyeq, \mathsf{pb}) = (S, \Sigma, s^0, \rightarrow_{\mathcal{R}})$ where transition relation $\rightarrow_{\mathcal{R}}$ is defined by $s \rightarrow_{\mathcal{R}} s'$ iff $s' \in \mathsf{pb}(a, s)$.

Lemma 4.1.14. *For any preordered labeled transition system* $(\mathcal{LTS}, \preccurlyeq)$ *with and finite pred-basis* pb*,* $\mathcal{R} = \mathcal{R}(\mathcal{LTS}, \preccurlyeq, \mathsf{pb})$ *is finitely-branching,* $(\mathcal{R}, \succcurlyeq)$ *has reflexive compatibility, and* $\mathcal{R}$ *satisfies*

$$\mathsf{pre}^*_{\mathcal{LTS}}(\preccurlyeq(\{s\})) = \preccurlyeq(\mathsf{post}^*_{\mathcal{R}}(\{s\}))$$

for every state s*.*

Proof. Let $\mathcal{R}$ shortly denote $\mathcal{R}(\mathcal{LTS}, \preccurlyeq, \mathsf{pb})$. Recall that $\mathcal{R}$ is finitely-branching since $\bigcup_{a\in\Sigma} \mathsf{pb}(a, s)$ is finite for every state s. Consider any transition $s \xrightarrow{a}_{\mathcal{R}} s'$ and let $t \preccurlyeq s$. One obtains that

$$\begin{aligned}\preccurlyeq(\mathsf{pb}(a, s)) &= \preccurlyeq(\mathsf{pre}_{\mathcal{LTS}}(a, \preccurlyeq(\{s\}))) \\ &\subseteq \preccurlyeq(\mathsf{pre}_{\mathcal{LTS}}(a, \preccurlyeq(\{t\}))) \\ &= \preccurlyeq(\mathsf{pb}(a, t))\end{aligned}$$

Since $s' \in \mathsf{pb}(a, s)$, we get that $s' \succcurlyeq t'$ for some $t' \in \mathsf{pb}(a, t)$. Observe that $t \xrightarrow{a}_{\mathcal{R}} t'$ and hence $(\mathcal{R}, \succcurlyeq)$ has reflexive compatibility.

Consider any state s, let us prove that $\mathsf{pre}^*_{\mathcal{LTS}}(\preccurlyeq(\{s\})) = \preccurlyeq(\mathsf{post}^*_{\mathcal{R}}(\{s\}))$. We first prove inclusion $\subseteq$ and assume that $\pi = s \xrightarrow{a_1}_{\mathcal{LTS}} s_2 \dots s_k \xrightarrow{a_k}_{\mathcal{LTS}} s_{k+1}$ is a path in $\mathcal{LTS}$ such that $s_{k+1} \succcurlyeq s$. Let $t_{k+1} = s$. Observe that $s_k \in \mathsf{pre}_{\mathcal{LTS}}(a_k, \preccurlyeq(\{t_{k+1}\})) \subseteq \preccurlyeq(\mathsf{pb}(a_k, t_{k+1}))$. Sin pb is a finite pred-basis for $(\mathcal{LTS}, \preccurlyeq)$, we get that $t_k \preccurlyeq s_k$ for some $t_k \in \mathsf{pb}(a_k, t_{k+1})$ and hence $t_{k+1} \xrightarrow{a_k}_{\mathcal{R}} t_k$. By iterating this construction along the path π (from s_{k+1} to s_1) we get that there exists a path $s = t_{k+1} \xrightarrow{a_k}_{\mathcal{R}} t_k \xrightarrow{a_{k-1}}_{\mathcal{R}} t_{k-1} \dots t_2 \xrightarrow{a_1}_{\mathcal{R}} t_1$ in $\mathcal{R}$ with $t_i \preccurlyeq s_i$, and in particular with $t_1 \preccurlyeq s_1$. Therefore $s_1 \in \preccurlyeq(\mathsf{post}^*_{\mathcal{R}}(\{s\}))$ which concludes the proof of this inclusion.

Let us now prove inclusion $\supseteq$ and assume that $s = t_{k+1} \xrightarrow{a_k}_{\mathcal{R}} t_k \xrightarrow{a_{k-1}}_{\mathcal{R}} t_{k-1} \dots t_2 \xrightarrow{a_1}_{\mathcal{R}} t_1$ is a path in $\mathcal{R}$ and let $s_1 \preccurlyeq t_1$. For every $1 \le i \le k$, we get that $t_i \in \mathsf{pb}(a_i, t_{i+1}) \subseteq \preccurlyeq(\mathsf{pre}_{\mathcal{LTS}}(a_i, \preccurlyeq(\{t_{i+1}\})))$ and hence there exist u_i, u'_i such that $t_i \succcurlyeq u_i \xrightarrow{a_i}_{\mathcal{LTS}} u'_i \succcurlyeq t_{i+1}$. We obtain from the compatibility of $(\mathcal{LTS}, \preccurlyeq)$ that there exists $s_2 \succcurlyeq t_2, \dots, s_{k+1} \succcurlyeq t_{k+1}$ such that $s_1 \xrightarrow{*}\!\!\!\!\rightarrow_{\mathcal{LTS}} s_2 \xrightarrow{*}_{\mathcal{LTS}} s_3 \dots s_k \xrightarrow{*}\!\!\!\!\rightarrow_{\mathcal{LTS}} s_{k+1}$. Therefore $s_1 \in \mathsf{pre}^*_{\mathcal{LTS}}(\preccurlyeq(\{s\}))$ since $s_{k+1} \succcurlyeq t_{k+1} = s$. □

We implicitly assumed that the partition of Σ into internal actions Σ^τ and normal actions Σ^γ is the same in $\mathcal{R}(\mathcal{LTS}, \preccurlyeq, \mathsf{pb})$ as in $\mathcal{LTS}$. It is clear from the proof of Lemma 4.1.14 that reflexive compatibility of $(\mathcal{R}(\mathcal{LTS}, \preccurlyeq, \mathsf{pb}), \succcurlyeq)$ does not depend on this partition (e.g. we may as well choose that there is no internal action in $\mathcal{R}$).

4.2 Truncation of well-preordered labeled event structures

The intuition behind well-structure/well-preorder is that any state may be weakly simulated by any greater state, and thus we may forget about smaller states when performing reachability analysis. The well-preordering condition between states guarantees termination of the analysis [FS01]. We show in this section how to extend these ideas to the verification of well-preordered labeled transition systems.

Notice that Finkel *et al.* call *tree-saturation methods* for well-preordered system, the verification methods representing all possible executions inside a finite tree-like structure (particularly the finite reachability tree). Since we model well-preordered systems by labeled event structures, it is hoped that these methods benefit of the partial-order advantage of event structures.

4.2.1 Well-preordered labeled event structures

We lift the well-preorder notions defined in the previous section from labeled transition systems to labeled event structures. Given a labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, $\mathcal{E}$ gives rise to a labeled transition system $\mathcal{LTS} = (\mathcal{C}_\mathcal{E}, \mathsf{Codom}(\mathcal{L}), \emptyset, \vdash_\mathcal{L})$ where for all $C, C' \in \mathcal{C}_\mathcal{E}, a \in \mathsf{Codom}(\mathcal{L})$, $C \xrightarrow{a}_{\mathcal{LTS}} C'$ iff there exists $e \in E$ so that $C \vdash e, C' = C \cup \{e\}$ and $\mathcal{L}(e) = a$. The transitive closure $\Vdash_\mathcal{L} = \twoheadrightarrow_{\mathcal{LTS}}$ of transition relation $\vdash_\mathcal{L}$ may be also defined from the extension set relation $\Vdash$ of $\mathcal{E}$. It is worth noticing that the marking function $\mathcal{M}$ is not taken into account in $\mathcal{LTS}$.

Let $\preccurlyeq^\mathcal{C}$ be any preorder on $\mathcal{C}_\mathcal{E}$ that is compatible with the labeled event structure $\mathcal{E}$ as defined in Definition 4.1.4. Intuitively, $C \vdash e$ where $C \in \mathcal{C}_\mathcal{E}, e \in E$ implies that every configuration C' satisfying $C \preccurlyeq^\mathcal{C} C'$ could be extended by an event set $X \subseteq E$, and the preorder $\preccurlyeq^\mathcal{C}$ is preserved, i.e. $(C \cup \{e\}) \preccurlyeq^\mathcal{C} (C \cup X)$. Moreover, with an additional condition based on the labeling function $\mathcal{L}$, the definition of well-preordered labeled event structures coincides with the one for labeled transition systems (Definition 4.1.5).

Definition 4.2.1. Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a labeled event structure and $\preccurlyeq^\mathcal{C}$ be any preorder on $\mathcal{C}_\mathcal{E}$. We say that $(\mathcal{E}, \preccurlyeq^\mathcal{C})$ is *(well-)preordered labeled event structure* iff $((\mathcal{C}_\mathcal{E}, \mathsf{Codom}(\mathcal{L}), \emptyset, \vdash_\mathcal{L}), \preccurlyeq^\mathcal{C})$ is a (well-)preordered labeled transition system.

Notice that the codomain of labeling function $\mathcal{L}$ may contains also some subset of internal actions Σ^τ. So that $(\mathcal{E}, \preccurlyeq^\mathcal{C})$ can have different compatibilities $\mathsf{Cond} \in \{$(non-strict), strict$\} \times \{$(standard), transitive, reflexive, strong$\}$ based on this Σ^τ as in Definition 4.1.5.

Preordered labeled transition systems vs preordered labeled event structure

Let us recall the idea of using labeled event structures for modeling concurrent systems. One just gives a partial-order structure, here is a labeled event structure, that possibly represents all behaviors of some system. The system could be modeled in another way by a well-known labeled transition system. These two models are related the one to the other by the means of induced labeled transitions systems of labeled event structures in which one associates configurations to systems' states. Therefore, while working with preordered labeled event structures, among many binary relations on configuration set, we focus only on the one which is deduced from a given preorder on the system's state space. In the following, for every labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, we denote S the base set of the codomain of marking function $\mathcal{M}$, i.e. $S = \bigcup_{C \in \mathcal{C}_\mathcal{E}} \mathcal{M}(C)$, like in Definition 3.2.1 on page 27.

Definition 4.2.2 (Marking preorder)**.** Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a labeled event structure and $\preccurlyeq$ be a preorder on S. The *marking preorder* of $\mathcal{E}$ w.r.t. $\preccurlyeq$, denoted by $\preccurlyeq^\mathcal{M}$, is a binary relation on $\mathcal{C}_\mathcal{E}$ defined by: for all $C, C' \in \mathcal{C}_\mathcal{E}$, $C \preccurlyeq^\mathcal{M} C'$ iff for all $s \in \mathcal{M}(C)$, there exists $s' \in \mathcal{M}(C')$ such that $s \preccurlyeq s'$.

In a particular case where $\mathcal{E}$ is deterministic, that means $\mathsf{Codom}(\mathcal{M})$ contains only singletons of $\mathcal{P}(S)$, one can simply use $\preccurlyeq$ in the place of $\preccurlyeq^\mathcal{M}$ without risk of confusion. Notice that, C is strictly less than C' w.r.t. $\preccurlyeq^\mathcal{M}$ iff $C \preccurlyeq^\mathcal{M} C'$ and their markings contain at least two elements which are strictly ordered by $\preccurlyeq$. Formally, $C \prec^\mathcal{M} C'$ iff $C \preccurlyeq^\mathcal{M} C'$ and $\exists s \in \mathcal{M}(C), s' \in \mathcal{M}(C') : s \prec s'$.

Lemma 4.2.3. *The marking preorder $\preccurlyeq^\mathcal{M}$ is a preorder on $\mathcal{C}_\mathcal{E}$ if $\preccurlyeq$ is a preorder on S.*

Proof. The reflexivity and transitivity of $\preccurlyeq^\mathcal{M}$ is obvious from Definition 4.2.2. □

Lemma 4.2.4. *Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a labeled event structure and $\preccurlyeq$ be a preorder on S. If*

1. *for all $s, s' \in S$, $s \preccurlyeq s'$ implies $\exists C, C \in \mathcal{C}_\mathcal{E} : C \preccurlyeq^\mathcal{M} C'$ and $\langle s, s' \rangle \in (\mathcal{M}(C) \times \mathcal{M}(C'))$,*
2. *for all $C, C' \in \mathcal{C}_\mathcal{E}$, $(\mathcal{M}(C) \times \mathcal{M}(C')) \cap \preccurlyeq \neq \emptyset$ implies $C \preccurlyeq^\mathcal{M} C'$, and*
3. *$\mathcal{E}$ is coherent;*

then $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ is a preordered labeled event structure iff $(\mathcal{LTS}^\mathcal{E}, \preccurlyeq)$ is a preordered labeled transition system, and they have a same type of compatibility.

Proof. Recall that Σ is the codomain of labeling function $\mathcal{L}$. One can write a induced labeled transition system of $\mathcal{E}$ as $\mathcal{LTS}^\mathcal{E} = (S, \Sigma, s^0, \rightarrow_{\mathcal{LTS}^\mathcal{E}})$ for some $s^0 \in \mathcal{M}(\emptyset)$, and the labeled transition system based on the extension relation previously defined as $\mathcal{LTS}_\vdash = (\mathcal{C}_\mathcal{E}, \Sigma, \emptyset, \vdash_\mathcal{L})$. We will prove that the right-to-left implication, that means $(\mathcal{LTS}^\mathcal{E}, \preccurlyeq)$ is preordered if $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ is preordered, is a consequence of the first item. And reversely, the left-to-right implication is a consequence of the second and third items.

($\Rightarrow$): $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ is a preorder labeled event structure. Let $s \xrightarrow{a}_{\mathcal{LTS}^\mathcal{E}} v$ be any transition in $\mathcal{LTS}^\mathcal{E}$ and assume that $s \preccurlyeq s'$ for some given $s' \in S$. Due to the definition of induced labeled transition system (Definition 3.2.4), there must exist configurations $C \in \mathcal{C}_\mathcal{E}$ and one of its extension event $e \in E$, i.e. $C \vdash e$ such that $\mathcal{L}(e) = a, s \in \mathcal{M}(C)$ and $v \in \mathcal{M}(C \cup \{e\})$. The first condition yields the existence of a configuration $C' \in \mathcal{C}_\mathcal{E}$ satisfying $C \preccurlyeq^\mathcal{M} C'$. Because $\preccurlyeq^\mathcal{M}$ is compatible with $\vdash_\mathcal{L}$, we have $C' \xrightarrow{\sigma}\!\!\!\!\rightarrow_{\mathcal{LTS}_\vdash} (C' \cup X)$ for some given extension set $X \subset E$ of C', and $\sigma = \mathcal{L}^\mathcal{W}(l_X) \in \Sigma^*$ where $l_X \in E^*$ is a linearisation of X w.r.t. $\leq$. And $(C \cup \{e\}) \preccurlyeq^\mathcal{M} (C' \cup X)$ is also a consequence of the compatibility of $\preccurlyeq^\mathcal{M}$. By definition of $\preccurlyeq^\mathcal{M}$ (Definition 4.2.2), since $v \in \mathcal{M}(C \cup \{e\})$, there exists $v' \in \mathcal{M}(C' \cup X)$ satisfying $v \preccurlyeq v'$ and $s' \xrightarrow{\sigma}\!\!\!\!\rightarrow_{\mathcal{LTS}^\mathcal{E}} v'$ due to Lemma 3.2.12. Therefore $\preccurlyeq$ is compatible with $\rightarrow_{\mathcal{LTS}^\mathcal{E}}$.

($\Leftarrow$): $(\mathcal{LTS}^\mathcal{E}, \preccurlyeq)$ is a labeled transition system. Suppose that $C \xrightarrow{\mathcal{L}(e)}_{\mathcal{LTS}_\vdash} (C \cup \{e\})$ and $C \preccurlyeq^\mathcal{M} C'$ for some given configuration $C, C' \in \mathcal{C}_\mathcal{E}$ and extension event $e \in E$ of C. By definition of $\preccurlyeq^\mathcal{M}$ (Definition 4.2.2), one can choose any two states $s, s' \in S$ which are respectively included in C and C' so that $s \preccurlyeq s'$. Let v be any state in $\mathcal{M}(C \cup \{e\})$, it follows from the definition of an induced labeled transition system (Definition 3.2.4) that $s \xrightarrow{\mathcal{L}(e)}_{\mathcal{LTS}^\mathcal{E}} v$. Moreover, due to the compatibility of $\preccurlyeq$ with $\rightarrow_{\mathcal{LTS}^\mathcal{E}}$, there exists an execution $s' \xrightarrow{\sigma}\!\!\!\!\rightarrow_{\mathcal{LTS}^\mathcal{E}} v'$ for some given $v' \prec v$, and more precisely a path $\pi = s' \xrightarrow{b_1}_{\mathcal{LTS}^\mathcal{E}} s_1 \xrightarrow{b_2}_{\mathcal{LTS}^\mathcal{E}} s_2 \ldots s_{n-1} \xrightarrow{b_n}_{\mathcal{LTS}^\mathcal{E}} s_n = v'$ where $\sigma = b_1.b_2 \ldots b_n \in M_\mathcal{E}^*$ and $v \preccurlyeq v'$. Since $\mathcal{E}$ is coherent, $s' \xrightarrow{b_1}_{\mathcal{LTS}^\mathcal{E}} s_1$ and $s' \in \mathcal{M}(C')$ implies that there exists configuration $C_1 = C' \cup e_1$ for a given extension event $e_1 \in E$ of C' such that $\mathcal{L}(e_1) = b_1$ and $s_1 \in C_1$. By iterating this construction along the path π we get that $C' \vdash (C' \cup \{e_1\}) \vdash (C' \cup \{e_1, e_2\}) \vdash \ldots (C' \cup X)$ where $X = \{e_1, e_2, \ldots, e_n\}$, $\sigma = \mathcal{L}^\mathcal{W}(e_1.e_2 \ldots e_n)$, and $s_n = v' \in \mathcal{M}(C \cup X)$. One thus can write it as $C' \xrightarrow{\sigma}\!\!\!\!\rightarrow_{\mathcal{LTS}_\vdash} (C' \cup X)$ and $\langle s', v' \rangle \in (\mathcal{M}(C') \times \mathcal{M}(C' \cup X)$. Thanks to the third condition, because $\langle v, v' \rangle \in (\mathcal{M}(C \cup \{e\}) \times \mathcal{M}(C \cup X))$ and $v \preccurlyeq v'$, we have $(C \cup \{e\}) \preccurlyeq^\mathcal{M} (C \cup X)$. Preorder $\preccurlyeq^\mathcal{M}$ is thus compatible with $\vdash_\mathcal{L}$.

Notice that conclusions above depend on neither the type of compatibility of $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ nor the one of $(\mathcal{LTS}^\mathcal{E}, \preccurlyeq)$, this lemma's whole state is thus obvious. □

It is worth considering here counter-examples of some implication between compatibilities of a preordered labeled event structure and of its induced labeled transition system, when we have not the three additional conditions stated in Lemma 4.2.4. Figure 4.3 illustrates two simple preordered labeled event structures where configurations

are structured as a DAG w.r.t. the extension relation. Each configuration C is represented by the couple $C : \mathcal{M}(C)$. And there is no concurrency in both labeled event structures, i.e. $\| = \emptyset$. For simplicity, we suppose that there is no internal action, i.e. $\Sigma^{\tau}_{\mathcal{E}} = \emptyset$, and we look only at standard compatibility.

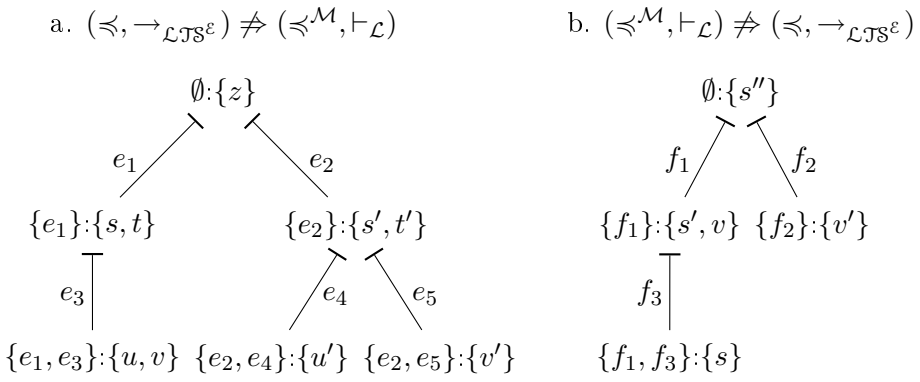


Figure 4.3: Counter examples of implication between compatibilities

For the first labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ represented in Figure 4.3-a where $E = \{e_1, e_2, e_3, e_4, e_5\}$ and $\mathcal{C}_{\mathcal{E}} = \{\emptyset, \{e_1\}, \{e_2\}, \{e_1, e_3\}, \{e_2, e_4\}, \{e_2, e_5\}\}$, we take the preorder $\preccurlyeq = \mathcal{I}_S \cup \{\langle s, s'\rangle, \langle t, t'\rangle, \langle u, u'\rangle, \langle v, v'\rangle\}$. It is obvious that the unique induced labeled transition system $\mathcal{LTS}^{\mathcal{E}}$ is a preordered one because the compatibility of $\preccurlyeq$ w.r.t. $\rightarrow_{\mathcal{LTS}^{\mathcal{E}}}$ is guaranteed mostly by e_4, e_5 and e_3. Consider configurations $C = \{e_1\}$ and $C' = \{e_2\}$ that could be ordered by $\preccurlyeq^{\mathcal{M}}$, we have $\mathcal{M}(C) = \{s, v\} \preccurlyeq^{\mathcal{M}} \{s', u'\} = \mathcal{M}(C')$. However, the marking of configuration $C \cup \{e_3\}$ extended from C, i.e. $\{u, v\}$, is incomparable with markings of configurations that may be obtained from C' by the extension relation. More precisely, there are three configurations $C \cup \{e_4\} = \{e_2, e_4\}$, $C' \cup \{e_5\} = \{e_2, e_5\}$ and $C' = \{e_2\}$ itself of which markings are respectively $\{u'\}, \{v'\}$, and $\{s', t'\}$. This example shows the need of the second condition in Lemma 4.2.4 that avoids the case when, for instance, $u \preccurlyeq u'$ but there exists incomparable markings $\{u, v\}$ and $\{u'\}$. In other words, the constraint given by this condition comes from the fact that one can not say anything on reachable states in $\mathsf{post}^*_{\mathcal{LTS}^{\mathcal{E}}}(\{u, v\})$ and $\mathsf{post}^*_{\mathcal{LTS}^{\mathcal{E}}}(\{u'\})$.

For the second example illustrated in Figure 4.3-b, where $E = \{f_1, f_2, f_3\}$ and $\mathcal{C}_{\mathcal{E}} = \{\emptyset, \{f_1\}, \{f_2\}, \{f_1, f_3\}\}$, let us assume that the preorder $\preccurlyeq$ satisfies $\preccurlyeq \cap (\{s, s', s''\} \times \{v, v'\}) = \emptyset$, $s \prec s' \prec s''$ and $v \prec v'$. By definition, $\preccurlyeq^{\mathcal{M}} = \mathcal{I}_{\mathcal{C}_{\mathcal{E}}} \cup \{\langle\{s\}, \{s', v\}\rangle\} \cup \{\langle\{s\}, \{s''\}\rangle\}$. It is obvious that $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ is preordered labeled event structure since the unique configuration $\{f_1, f_3\}$ of which marking is $\{s\}$, has no extension. However, $\preccurlyeq$ is not compatible with $\rightarrow_{\mathcal{LTS}^{\mathcal{E}}}$ due to the fact that $v \xrightarrow{\mathcal{L}(f_3)}_{\mathcal{LTS}^{\mathcal{E}}} s$ while $v \in \mathcal{M}(\{f_2\})$ is not enabled by any action in $\Sigma = \mathsf{Codom}(\mathcal{L})$. The first condition in Lemma 4.2.4 may eliminate all such cases where, for instance here, $v \preccurlyeq v'$ but there is no comparable configurations w.r.t. $\preccurlyeq^{\mathcal{M}}$ that contain respectively v and v'. With this condition, the compatibility of $\preccurlyeq^{\mathcal{M}}$ in $\mathcal{E}$ implies the one of $\preccurlyeq$ in $\mathcal{LTS}^{\mathcal{E}}$ as it is proved previously. By the way, we also point out that when working only on preorder labeled event structures, one does not really need the compatibility in its induced labeled transition system.

Remark: In deterministic labeled event structures, since every marking is singleton of S, the first and second conditions in Lemma 4.2.4 are both guaranteed.

The following lemma is a direct consequence of Lemma 4.2.4.

Lemma 4.2.5. *Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a deterministic and coherent labeled event structure and $\preccurlyeq$ be a preorder on S. $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ is a (well-)preordered labeled event structure iff $(\mathcal{LTS}^{\mathcal{E}}, \preccurlyeq)$ is a (well-)preordered labeled transition system, and they have a same type of compatibility* $\mathsf{Cond} \in$ *{(non-strict), strict} $\times$ {(standard), transitive, reflexive, strong}.*

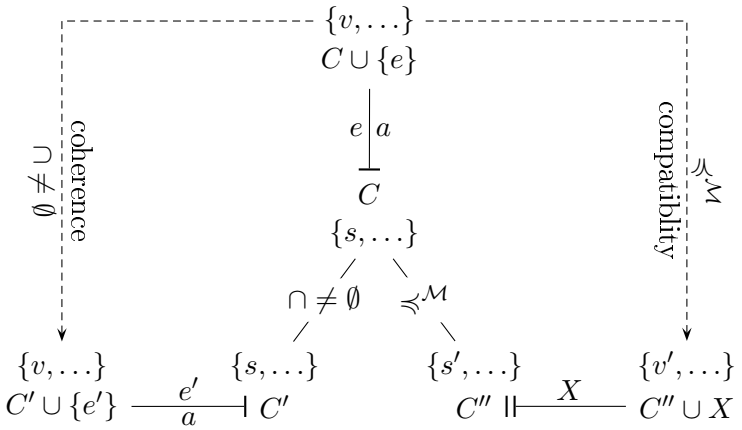


Figure 4.4: Coherence vs compatibility

Coherence is a required condition for the two-way implication stated in Lemma 4.2.5 and it somehow looks like compatibility. However, coherence and compatibility are quite different, as illustrated in Figure 4.4.

Lemma 4.2.6. *Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a coherent and deterministic labeled event structures. $(\mathcal{E}, \mathcal{I}^{\mathcal{M}})$ is a preordered labeled event structure with strong compatibility where $\mathcal{I}^{\mathcal{M}} = \mathcal{I}_S$ is the identity relation over S.*

Proof. The coherence of $\mathcal{E}$ by Definition 3.2.11 is exactly the strong compatibility of identity order $\mathcal{I}_S$ with $\vdash_{\mathcal{L}}$. □

In Section 4.3, one will see that compatibility of preordered labeled event structures, which may be nondeterministic, is enough for truncating, and consequently, doing certain verifications on their finite prefixes obtained. One does not need to see whether there exists a compatibility in its induced labeled transition systems.

Products of preordered labeled event structures

Definition 4.2.7. Given a number $n \in \mathbb{N}$. A synchronized product of n preordered labeled event structures $(\mathcal{E}_1, \preccurlyeq_1^{\mathcal{C}}), (\mathcal{E}_2, \preccurlyeq_2^{\mathcal{C}}), \ldots (\mathcal{E}_n, \preccurlyeq_n^{\mathcal{C}})$ is any tuple $(\mathcal{E}_\otimes, \preccurlyeq_\otimes^{\mathcal{C}})$ where

- $\mathcal{E}_\otimes$ is a synchronized product of $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ w.r.t. some synchronization constraint $\Sigma_\otimes$, and
- $\preccurlyeq_\otimes^{\mathcal{C}}$ is a binary order on $\mathcal{C}_{\mathcal{E}_\otimes}$ defined by: $C \preccurlyeq_\otimes^{\mathcal{C}} C'$ iff for all $1 \leq i \leq n$, $(\mathcal{V}(C))\!\downarrow_i \preccurlyeq_i^{\mathcal{C}} (\mathcal{V}(C'))\!\downarrow_i$ where $\mathcal{V}$ is the synchronization function of $\mathcal{E}_\otimes$.

Recall that the synchronization function $\mathcal{V}$ could map each configuration C in the synchronized product $\mathcal{E}_\otimes$ into a tuple $\langle C_1, C_2, \ldots, C_n \rangle$, where $C_i \in \mathcal{C}_{\mathcal{E}_i}$ (see Lemma 3.3.42 on page 57), and $\mathcal{V}$ is not injective. The preorder property of $\preccurlyeq_\otimes^{\mathcal{C}}$ is trivial and similar to the one of the product preorder $\otimes(\preccurlyeq_1^{\mathcal{C}}, \preccurlyeq_2^{\mathcal{C}}, \ldots, \preccurlyeq_n^{\mathcal{C}})$ on $\otimes_\varepsilon(\mathcal{C}_{\mathcal{E}_1}, \mathcal{C}_{\mathcal{E}_2}, \ldots, \mathcal{C}_{\mathcal{E}_n})$.

Lemma 4.2.8. *Let* Cond *denote any compatibility condition among {(non-strict), strict} × {(standard), transitive, reflexive}. Any synchronized product of preordered labeled event structures with compatibility* Cond *also has compatibility* Cond.

Proof. Let $(\mathcal{E}_\otimes, \preccurlyeq^{\mathcal{C}}_\otimes)$ denote some synchronized product of n preordered labeled event structures $(\mathcal{E}_1, \preccurlyeq^{\mathcal{C}}_1), (\mathcal{E}_2, \preccurlyeq^{\mathcal{C}}_2), \ldots, (\mathcal{E}_n, \preccurlyeq^{\mathcal{C}}_n)$. Suppose that $\mathcal{LTS}_{\vdash_\otimes}$ $= (\mathcal{C}_{\mathcal{E}_\otimes}, \Sigma_{\mathcal{E}_\otimes}, s^0_\otimes, \vdash_\otimes)$ and $\mathcal{LTS}_{\vdash_1}, \mathcal{LTS}_{\vdash_2}, \ldots, \mathcal{LTS}_{\vdash_n}$ are respectively their labeled transition systems based on the extension relation. We define another labeled transition system $\mathcal{LTS}_\otimes = (S_\otimes, \Sigma_{\mathcal{E}_\otimes}, s^0_{\mathcal{E}_\otimes}, \rightarrow_\otimes)$ as the synchronized product of n labeled transition system $\mathcal{LTS}_{\vdash_1}, \mathcal{LTS}_{\vdash_2}, \ldots, \mathcal{LTS}_{\vdash_n}$, its synchronized constraint is $\Sigma_{\mathcal{E}_\otimes}$. We have then:

1. by Definition 4.2.7, for all $C, C' \in \mathcal{C}_{\mathcal{E}_\otimes}$, $C \preccurlyeq^{\mathcal{C}}_\otimes C'$ iff $\langle \mathcal{R}_S(C), \mathcal{R}_S(C')\rangle \in \otimes(\preccurlyeq^{\mathcal{C}}_1, \preccurlyeq^{\mathcal{C}}_2, \ldots, \preccurlyeq^{\mathcal{C}}_n)$ where $\mathcal{R}_S(C) = \langle \mathcal{V}(C)\downarrow_1, \mathcal{V}(C)\downarrow_2, \ldots, \mathcal{V}(C)\downarrow_n\rangle$ for all $C \in \mathcal{C}_{\mathcal{E}_\otimes}$.
2. $(\mathcal{LTS}_\otimes, \otimes(\preccurlyeq^{\mathcal{C}}_1, \preccurlyeq^{\mathcal{C}}_2, \ldots, \preccurlyeq^{\mathcal{C}}_n))$ is a preordered labeled transition system with compatibility Cond due to Lemma 4.1.10.
3. $\mathcal{LTS}_{\vdash_\otimes}$ and $\mathcal{LTS}_\otimes$ are bisimilar w.r.t. $(\mathcal{R}_S, \mathcal{I}_{\Sigma_{\mathcal{E}_\otimes}})$ (*cf.* Definition 2.4.17) because of the maximization in $\mathcal{E}_\otimes$ by Definition 3.3.44.

Hence, $(\mathcal{LTS}_{\vdash_\otimes}, \preccurlyeq^{\mathcal{C}}_\otimes)$ as well as $(\mathcal{E}_\otimes, \preccurlyeq^{\mathcal{C}}_\otimes)$ is preordered with compatibility Cond. □

4.2.2 Truncation techniques

Although labeled event structures preserve all system's behaviors, they unfortunately may be infinite in general, as it may be "too deep" and/or "too wide". A well-preordering condition avoids the first possibility, and a branching finiteness assumption eliminates the seconds. Hence, it is hoped that one can obtain some finite parts of a labeled event structure to decide several verification problems.

Our technique is not far from the *tree-saturation methods* given in [Fin87, Fin91, FS01] where all system's possible executions are represented in some way inside a finite tree-like structure. In this section, we give the general idea of a truncation technique, and the more convenient one when partial-order is taken into account will be described in Section 4.3.

Cutting context

Definition 4.2.9. A *cutting context* of a given labeled event structure $\mathcal{E}$ is any tuple $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$ where:

1. $(\mathcal{E}, \preccurlyeq^{\mathcal{C}})$ is a preordered labeled event structure,
2. for all $C, C' \in \mathcal{C}_{\mathcal{E}}$, $C \subset C'$ implies $C \not\preccurlyeq^{\mathcal{C}} C'$,
3. $\mathcal{C} \subseteq \mathcal{C}_{\mathcal{E}}$ is a set of configurations.

We call the second property in Definition 4.2.9 the *inclusion respect condition*. Intuitively, by using a preorder $\preccurlyeq^{\mathcal{C}}$, for instance $\preccurlyeq^{\mathcal{C}}$=$\preccurlyeq^{\mathcal{M}}$, one may be interested in only configurations which are maximal w.r.t. $\preccurlyeq^{\mathcal{C}}$ when doing some analysis such as computing reachable states of possible induced labeled transition systems. However, notice that in practice, a labeled event structure $\mathcal{E}$ is constructed step by step by means of prefixes so that its configurations as well as its events are added to $\mathcal{E}$ w.r.t. the inclusion order $\subset$ (see Chapter 5). Every configuration comes from its sub-configurations w.r.t. the inclusion order. Suppose the opposite, $\preccurlyeq^{\mathcal{C}}$ does not stick to the truncation idea described later. For example, some configuration C' may be used for cutting another one C because $C \preccurlyeq^{\mathcal{C}} C'$ while C is a subset of C'. Therefore, the inclusion respect condition is naturally required.

We introduce the set of configurations $\mathcal{C}$ in Definition 4.2.9 in order to make this definition general. In this work, only two cases of $\mathcal{C}$ where $\mathcal{C}$ is either $\mathcal{C}_\mathcal{E}$ or $\mathcal{C}^l_\mathcal{E}$ are discussed (see Section 4.3). In our first approach to truncation technique, let us temporarily ignore the importance of configuration set $\mathcal{C}$ and assume that it is the configuration set $\mathcal{C}_\mathcal{E}$. This assumption leads us to the standard truncation technique on the coverability tree of an infinite system [Fin91].

Definition 4.2.10 (Cutoff configuration)**.** Let $(\preccurlyeq^\mathcal{C}, \mathcal{C})$ be a cutting context for a labeled event structure $\mathcal{E}$. A *cutoff configuration* is any configuration $C_{cut} \in \mathcal{C}$ such that there exists another configuration $C \in \mathcal{C}$ satisfying $C_{cut} \prec^\mathcal{C} C$.

We call a configuration $C_{out} \in \mathcal{C}_\mathcal{E}$ an *outer configuration* if C_{out} is greater than some cutoff configuration C_{cut} w.r.t. the inclusion order $\subseteq$. Such an outer configuration C_{out} may be obtained by the extension relation from a cutoff configuration C_{cut}, i.e. $C_{cut} \Vdash C_{out}$. Hence, C_{out} is intuitively useless, from the point of view of verification, due to the compatibility of preordered labeled event structures $(\mathcal{E}, \preccurlyeq^\mathcal{C})$. More precisely, it follows from Definition 4.2.10 that C_{cut} is cutoff due to the existence of some configuration C such that $C_{cut} \prec^\mathcal{C} C$, and as a consequence, there exists another configuration C' obtained from C, i.e. $C \Vdash C'$, satisfying $C_{out} \prec^\mathcal{C} C'$.

Remark: If $\mathcal{C} = \mathcal{C}_\mathcal{E}$ then every outer configuration is also a cutoff configuration. However, it does not hold in general when $\mathcal{C} \subset \mathcal{C}_\mathcal{E}$. Because, for example, either C_{out} or the corresponding configuration C' in the above reasoning may not be in $\mathcal{C}$.

It is worth noticing here that in other works on truncation techniques, one can find only notions of cutoff configuration/event ([McM95a]) or of subsume node ([Fin87]) which all coincide with our cutoff notion. Our notion of outer configuration on the one hand clarifies the definition of truncation below, and on the other hand, distinguishes outer ("useless") configurations with cutoff ones of which some are required for verifying certain problem, for instance, system's boundedness.

Notation 4.2.11. For a given labeled event structure $\mathcal{E}$ and any cutting context $(\preccurlyeq^\mathcal{C}, \mathcal{C})$, we denote

- $\mathcal{C}^o_\mathcal{E}$ the family of outer configurations,
- $\mathcal{C}^c_\mathcal{E}$ the family of cutoff configurations, and
- $\mathcal{C}^n_\mathcal{E}$ the family of configurations which are neither cutoff nor outer ones, i.e. $\mathcal{C}^n_\mathcal{E} = \mathcal{C}_\mathcal{E} \setminus (\mathcal{C}^o_\mathcal{E} \cup \mathcal{C}^c_\mathcal{E})$.

Definition 4.2.12 (Truncation)**.** Let $(\preccurlyeq^\mathcal{C}, \mathcal{C})$ be a cutting context of a labeled event structure $\mathcal{E}$. The *truncation* of $\mathcal{E}$ w.r.t. $(\preccurlyeq^\mathcal{C}, \mathcal{C})$, denoted by $\mathcal{T}(\mathcal{E}, \preccurlyeq^\mathcal{C}, \mathcal{C})$, is the union of all non-outer configurations, i.e. :

$$\mathcal{T}(\mathcal{E}, \preccurlyeq^\mathcal{C}, \mathcal{C}) = \bigcup_{C \in (\mathcal{C}_\mathcal{E} \setminus \mathcal{C}^o_\mathcal{E})} C$$

Example 4.2.13. Let us consider an example when $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ is the labeled event structures for the 2-bounded counter initialized by 1, i.e. 2-$\mathcal{BC}^1$. Figure 4.5 illustrates $\mathcal{E}$ which is obtained from 2-bounded process 2-$\mathcal{BP}$ (see Section 3.3.2). Since $\mathcal{E}$ is deterministic and coherent and $\mathsf{post}^*_{\text{2-}\mathcal{BC}^1} = \mathsf{Codom}(\mathcal{M}) = \{0, 1, 2\}$, $(\mathcal{E}, \mathcal{I}_\mathbb{N})$ is a well-preordered labeled event structure. Moreover, as proved in the next subsection, $(\mathcal{E}, \preccurlyeq^\mathcal{C})$ is also well-preordered with strict compatibility where $\preccurlyeq^\mathcal{C}$ is defined by: for all $C, C' \in \mathcal{C}_\mathcal{E}$, $C \preccurlyeq^\mathcal{C} C'$ iff $\mathcal{M}(C) = \mathcal{M}(C')$ and $|C| \geq |C'|$.

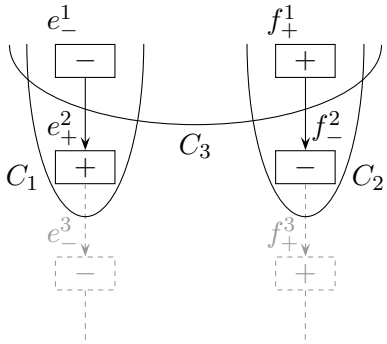


Figure 4.5: Truncation example of a labeled event structure for 2-bounded counter initialized by 1.

One can see that $\mathcal{C}^n_{\mathcal{E}}$ contains only three configurations $\emptyset, \{e^1_-\}$, and $\{f^1_+\}$ whose markings are respectively $1, 0$ and 2. All other configurations could not have a marking out of the set $\{0, 1, 2\}$ and because its size is greater than 1, they are all cutoff configurations if the cutting context is $(\preccurlyeq^{\mathcal{C}}, \mathcal{C}_{\mathcal{E}})$. For example, $C_1 = \{e^1_-, e^2_+\}$, $C_2 = \{f^1_+, f^2_-\}$, and $C_3 = \{e^1_-, f^1_+\}$ are all cutoff ones due to the configuration $\emptyset$.

It follows then from Definition 4.2.12 that the truncation of $\mathcal{E}$ w.r.t. the cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C}_{\mathcal{E}})$ is the set $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}_{\mathcal{E}}) = \{e^1_-, e^2_+, f^1_+, f^2_-\}$. Events which are outside of this truncation are illustrated in Figure 4.5 by dashed line. The truncation, and more precisely the prefix of $\mathcal{E}$ based on $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}_{\mathcal{E}})$ is enough, for instance, for computing all possible markings. Formally, let us simply denote the truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}_{\mathcal{E}})$ by $\mathcal{T}$, and the $\mathcal{T}$-prefix $\mathcal{E}|_{\mathcal{T}}$ by the tuple $\mathcal{E}' = (\mathcal{T}, \leq', \#', \mathcal{L}', \mathcal{M}')$, we obtain that

$$\begin{aligned}\bigcup_{C \in \mathcal{C}_{\mathcal{E}}} \mathcal{M}(C) &= \bigcup_{C \in \mathcal{C}_{\mathcal{E}} : C \subseteq \mathcal{T}} \mathcal{M}(C) \\ &= \bigcup_{C \in \mathcal{C}_{\mathcal{E}'}} \mathcal{M}'(C) \\ &= \{0, 1, 2\}\end{aligned}$$

Remark: By definition, $C \subseteq \mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ for every configuration $C \in \mathcal{C}^n_{\mathcal{E}}$. And all cutoff configurations which are minimal w.r.t. inclusion are also subsets of the truncation. Although $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ is the union of non-outer configurations, it may contain some outer-configurations. As shown in the example above, the truncation is an outer one itself.

Truncation's properties

Because every configuration is a downward closed set w.r.t. causality, a truncation is also downward closed set w.r.t. causality. When there is no risk of confusion, we also call the prefix of a labeled event structure $\mathcal{E}$ based on its truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ for some given cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$, i.e. $\mathcal{E}|_{(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})}$, its truncation.

This truncation is determined somehow by the set of all minimal cutoff configurations w.r.t. the inclusion order $\subseteq$. In other words, the truncation is bounded by events whose local configurations are of course outer ones, but more precisely, by some of them which are minimal w.r.t. the causality. Formally, the set $E_f = \mathsf{Min}_{\leq}(\{e \in E \,/\, {>}(e) \in \mathcal{C}^c_{\mathcal{E}}\})$ is

intuitively the outside-frontier of the $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$-prefix of $\mathcal{E}$, and

$$\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}) = E \setminus {\leq}(E_f)$$

It is due to a consequence of the fact that every configuration $C \in \mathcal{C}_{\mathcal{E}}$ containing a successor of an event $e_f \in E_f$ must contain e_f itself. Therefore, C is an outer configuration because ${>}(e_f) \subset C$ and ${>}(e_f) \in \mathcal{C}^c_{\mathcal{E}}$. In example in Figure 4.5, we have $E_f = \{e^3_-, f^3_+\}$.

Theorem 4.2.14 (Completeness). *Let $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$ be a cutting context of a labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$. If $\preccurlyeq^{\mathcal{C}}$ is converse well-founded and $(\mathcal{E}, \preccurlyeq^{\mathcal{C}})$ has strict compatibility then for all configurations $C \in \mathcal{C}_{\mathcal{E}}$, there exists a configuration $C' \in \mathcal{C}^n_{\mathcal{E}}$ such that $C \preccurlyeq^{\mathcal{C}} C'$.*

Proof. We will prove this theorem by contradiction. Suppose that there exists a configuration $C \in \mathcal{C}_{\mathcal{E}}$ such that for all $C' \in \mathcal{C}^n_{\mathcal{E}}$, $C \not\preccurlyeq^{\mathcal{C}} C'$ (*). It follows the reflexivity of $\preccurlyeq^{\mathcal{C}}$ that $C \notin \mathcal{C}^n_{\mathcal{E}}$, and as a consequence, there are two cases:

- C is a cutoff configuration, i.e. $C \in \mathcal{C}^c_{\mathcal{E}}$. By definition, it is due to another configuration $C_1 \in \mathcal{C} \subset \mathcal{C}_{\mathcal{E}}$ satisfying $C \prec^{\mathcal{C}} C_1$.
- C is an outer configuration, i.e. $C \in \mathcal{C}^o_{\mathcal{E}}$. Once again, there must exist a cutoff configuration C_{cut} and thus another configuration C' such that: $C_{cut} \subset C, C_{cut} \prec^{\mathcal{C}} C'$. Thanks to the strict compatibility of $(\mathcal{E}, \preccurlyeq^{\mathcal{C}})$, there exists a configuration $C_1 \in \mathcal{C}_{\mathcal{E}}$ which may be obtained from C', i.e. $C' \Vdash C_1$, and satisfies that $C \prec^{\mathcal{C}} C_1$.

In both cases, one can conclude that there exists $C_1 \in \mathcal{C}_{\mathcal{E}}$ satisfying $C \prec C_1$. It follows from the hypothesis (*) that $C_1 \notin \mathcal{C}^n_{\mathcal{E}}$. By repeating this reasoning, we obtain an infinite sequence of configurations which is an increasing sequence w.r.t. $\preccurlyeq^{\mathcal{C}}$, that means $C \prec^{\mathcal{C}} C_1 \prec^{\mathcal{C}} C_2, \ldots$ where $C_i \in \mathcal{C}_{\mathcal{E}}$ for all $i \in \mathbb{N}$. This contradicts to the converse well-foundedness of $\preccurlyeq^{\mathcal{C}}$. Therefore, hypothesis (*) thus results in contradictions. □

The "respect inclusion" of cutting contexts is not only a natural property in practice but also gives rise to the converse well-foundedness of the order $\preccurlyeq^{\mathcal{C}}$. And it is thus a key in the proof of Theorem 4.2.14. This theorem is somehow called "truncation completeness" theorem, for instance, for reachability based verification (see Section 4.3). However, such a verification is decidable if the corresponding truncation is finite.

Theorem 4.2.15 (Finiteness). *Let $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$ be a cutting context of a finitely-branching labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$. If $\preccurlyeq^{\mathcal{C}}$ is converse well-preordered and $\mathcal{C}^l_{\mathcal{E}}$ is a subset of $\mathcal{C}$, then the truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ is finite.*

Proof. We first prove that for every event $e \in E$, if its local configuration ${\geq}(e)$ is an outer configuration then $e \notin \mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ (*). Suppose that it is not true for some even e. Since ${\geq}(e)$ is an outer configuration, there exists a configuration $C_{cut} \in \mathcal{C}^c_{\mathcal{E}}$ such that $C_{cut} \subset ({\geq}(e))$. As a consequence, for all configuration $C \in \mathcal{C}_{\mathcal{E}}$ satisfying $e \in C$, C must be an outer configuration due to C_{cut}, more precisely because $C_{cut} \subset ({\geq}(e)) \subseteq C$. In other words, every non-outer configuration can not contain e. It follows from the definition of the truncation (Definition 4.2.12) that $e \notin \mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$.

Now, suppose that the truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ is infinite. Let us consider the DAG (V, E') where $V = \mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ and E' is defined by: for all $e, e' \in V, \langle e, e' \rangle \in E'$ iff $e \lessdot e'$. Since $\mathcal{E}$ is finitely-branching, (V, E') is finitely-branching too. As a consequence of König's lemma, there exists an infinite path $e_1 \lessdot e_2 \lessdot \ldots$ in $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$. Due to the converse well-preorder $\preccurlyeq^{\mathcal{C}}$, there exists indices $k > i$ such that $({\geq}(e_k)) \preccurlyeq^{\mathcal{C}} ({\geq}(e_i))$. Thanks to the inclusion respect condition of cutting contexts, it follows from $({\geq}(e_i)) \subset$

$(\geq(e_k))$ that $(\geq(e_i)) \not\preccurlyeq^{\mathcal{C}} (\geq(e_k))$. We have thus $(\geq(e_k)) \prec^{\mathcal{C}} (\geq(e_i))$. Therefore, $\geq(e_k)$ is a cutoff configuration because both configurations $\geq(e_i)$ and $\geq(e_k)$ are elements of $\mathcal{C}^l_{\mathcal{E}} \subseteq \mathcal{C}$. Hence, for every index $l > k$, the local configuration $\geq(e_l)$ is an outer configuration. The fact that e_l belongs to the truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ contradicts (*). One can finally conclude that $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$ is finite. □

The family of configurations $\mathcal{C}$ of a cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$ is introduced for a general cutting context. We have not found the precise condition on $\mathcal{C}$ for the finiteness of the corresponding truncation stated in Theorem 4.2.15. However, our condition that $\mathcal{C}_{\mathcal{E}} \supseteq \mathcal{C}^l_{\mathcal{E}}$ is enough for partial-ordered verifications detailed in Section 4.3 in which the local cutting context is based on the family of local configurations $\mathcal{C}^l_{\mathcal{E}}$. We also hope that a good choice of $\mathcal{C}$ can help tuning our verification algorithms detailed in the next chapters.

4.2.3 Well-preorders on configurations

All different variants of truncating techniques may be generalized by ours. The preorder on configurations is determined by a pair of orders $(\preccurlyeq^{\mathcal{M}}, \trianglelefteq)$ where $\preccurlyeq^{\mathcal{M}}$ and $\trianglelefteq$ are both orders on configurations, and

- $\preccurlyeq^{\mathcal{M}}$ is based on the marking function,
- $\trianglelefteq$ is based on the inclusion order or/and the labeling function.

The order $\trianglelefteq$, and more precisely its strict order $\triangleleft$, called *adequate order*, must be a strict partial-order on $\mathcal{C}_{\mathcal{E}}$ refining/extending the inclusion order $\subset$, i.e. $C \subset C'$ implies $C \triangleleft C'$. This property corresponds to the inclusion respect condition on cutting contexts. Moreover, this pair of orders $(\preccurlyeq^{\mathcal{M}}, \trianglelefteq)$ must be preserved by finite extensions. This means that for every pair of configurations $C \preccurlyeq^{\mathcal{M}} C'$, and for every extension set X of C, i.e. $C \Vdash X$, there exists an extension set X' of C' such that $(C \cup X) \preccurlyeq^{\mathcal{M}} (C' \cup X')$, and if $C \trianglelefteq C'$ then $(C \cup X) \trianglelefteq (C' \cup X')$. It intuitively coincides with our definition of compatibility of $\preccurlyeq^{\mathcal{M}}$ with extension relation $\Vdash$, and will be proved in the rest of this sub-section. One can find in the literature the following orders:

Marking orders

- $\preccurlyeq^{\mathcal{M}} = \mathcal{I}_{\mathsf{Codom}(\mathcal{M})}$ for finite systems [McM95a] which is the most widely used for unfolding technique on safe Petri nets.
- $\preccurlyeq^{\mathcal{M}} = \approx_{\mathsf{Codom}(\mathcal{M})}$ where configurations' markings may be partitioned into finite classes and $\approx_{\mathsf{Codom}(\mathcal{M})}$ is the corresponding equivalence relation [KK03]. For example, this is used for symmetric Petri nets [CGP01] or Signal Transition Graphs [SY96].
- $\preccurlyeq^{\mathcal{M}}$ is a well-preorder on states of well-structured transition systems [Fin87].

Adequate orders

- $\trianglelefteq = \subseteq$ for finite recovery trees [Fin87, FS01].
- $\trianglelefteq$ based on configurations' sizes, $C \trianglelefteq C'$ iff $|C| \geq |C'|$, for unfolding of Petri nets [McM95a].
- $\trianglelefteq$ based on lexicographic order over Σ^* or Foata normal form of configurations [ERV96].

In this work, we focus only on well-preorders $\preccurlyeq^{\mathcal{C}}$ that are based on some marking preorders $\preccurlyeq^{\mathcal{M}}$ (see Definition 4.2.2) and some adequate orders $\trianglelefteq$ given above. Intuitively,

$\preccurlyeq^{\mathcal{M}}$ gives rise to the compatibility of cutting context while $\trianglelefteq$ guarantees the foundedness of $\preccurlyeq^{\mathcal{C}}$, and as a consequence the decidability of some verification problems.

Notation 4.2.16. Let $\mathcal{E}$ be a labeled event structure and $\preccurlyeq^{\mathcal{M}}, \trianglelefteq$ are respectively a marking order and an adequate order for $\mathcal{E}$. We denote $\preccurlyeq^{\mathcal{C}} = (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$ the binary relation on $\mathcal{C}_{\mathcal{E}}$ that is defined as: denoted by $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$ where

- $C \preccurlyeq^{\mathcal{C}} C'$ iff $\langle C, C' \rangle \in (\preccurlyeq^{\mathcal{M}} \cap \trianglerighteq)$,
- $C \prec^{\mathcal{C}} C'$ iff $C \preccurlyeq^{\mathcal{C}} C'$ and $C \rhd C'$.

The adequate order based on lexicographic order defined below has been first given by Esparza and is widely used in nowadays unfolding techniques for Petri nets. One can find its definition as well as the original idea of how to improve unfolding techniques in [ERV96]. Briefly, the purpose of giving or refining adequate orders is to obtain small truncations.

Definition 4.2.17. Given a labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ and a total order $\ll$ on Σ. The *lexicographic labeling order* on $\mathcal{C}_{\mathcal{E}}$, denoted by $\trianglelefteq_l$, is defined by: for all $C, C' \in \mathcal{C}_{\mathcal{E}}$, $C \trianglelefteq_l C'$ if either:

- $|C| < |C'|$, or
- $|C| = |C'|$ and the linearisation w.r.t. $\ll$ of labels of events in C is lexicographically smaller than or equal to the one of C'.

4.3 Partial-order verification for well-preordered labeled event structures

Based on the general cutting context defined in the previous section, we are going to show different truncation techniques. Each truncation technique is dedicated to a kind of information in labeled event structures that one wants to preserve by means of truncations and its corresponding prefixes. Then, by analyzing these prefixes, one can verify various problems on systems which are modeled by labeled event structures, such as termination and boundedness. There are two types of cutting contexts $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$ which are widely used. It depends on the choice of the configuration set $\mathcal{C}$ which is either the whole set of configurations $\mathcal{C}_{\mathcal{E}}$ or the set of local configurations $\mathcal{C}^l_{\mathcal{E}}$.

The first one, i.e. $\mathcal{C} = \mathcal{C}_{\mathcal{E}}$, is called a *global cutting context*, can result to a compact truncation but contradicts to the partial-order construction of labeled event structures (see Chapter 5). Because, in practice, one must compute and examine all configurations and their markings in order to decide the truncation and whether the constructing algorithm may terminate.

The second one, i.e. $\mathcal{C} = \mathcal{C}^l_{\mathcal{E}}$, may be integrated in the algorithm that constructs labeled event structures. It keeps up the partial-order idea and almost does not slow down the running time of the algorithm in practice (see Chapter 6 for details). In this section, we only consider this kind of cutting context, called *local cutting context*, however, all statements as well as their proofs are also true for the global cutting context.

4.3.1 Local cutting contexts

Let us firstly give some new notions about events that are derived from the ones about configurations.

Definition 4.3.1. Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a labeled event structure and $(\preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$ be a local cutting context of $\mathcal{E}$. An event $e \in E$ is

- a *cutoff event* if $\geq(e)$ is a cutoff configuration,
- an *outer event* if $\geq(e)$ is an outer configuration.

Since cutoff configurations are determined by $\mathcal{C}^l_{\mathcal{E}}$, the second item of Definition 4.3.1 can be stated in another way, event e_{out} is an outer event e if it is a successor of some cutoff event e_{cut}, i.e. $e_{cut} < e_{out}$. By the same manner as in Notation 4.2.11, we also denote the set of cutoff events, the set of outer events by E^c, E^o respectively; and the set of events which are neither cutoff nor outer by E^n, i.e. $E^n = (E \setminus E^c) \setminus E^o$.

Lemma 4.3.2. *Given a local cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$ of a labeled event structure $\mathcal{E}$. We have*

$$\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}}) = E^n \cup \mathsf{Min}_{\leq}(E^c).$$

Proof. In the same manner as the proof of Theorem 4.2.15, we obtain that for all $e \in E^o$, $e \notin \mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$. It follows directly from the definition of outer events that for all $e \in E^n$, $(\geq(e) \cap E^o) = \emptyset$. And moreover, a cutoff event $e_{cut} \in E^c$ is an outer event iff it is a successor of some other cutoff event, or in other words, iff it is not minimal w.r.t. causality over the set of cutoff ones, i.e. $e_{cut} \notin \mathsf{Min}_{\leq}(E^c)$. Therefore, by definition of the truncation (Definition 4.2.12), $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}}) = \bigcup_{C \in (\mathcal{C}_{\mathcal{E}} \setminus \mathcal{C}^o_{\mathcal{E}})} C = (\geq(E^n)) \cup (\geq(\mathsf{Min}_{\leq}(E^c))) = E^n \cup \mathsf{Min}_{\leq}(E^c)$. □

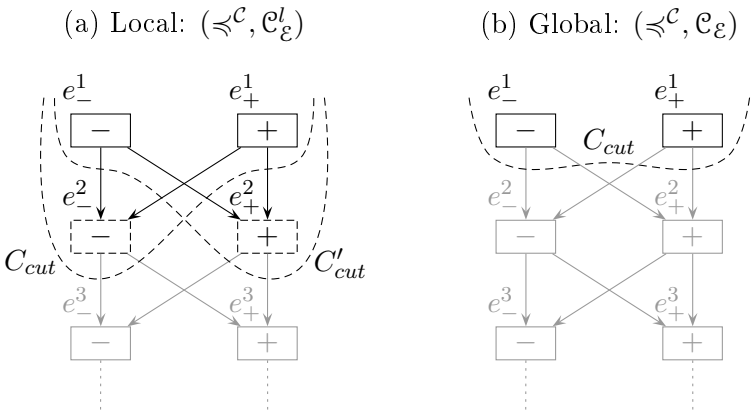


Figure 4.6: Local vs global cutting contexts

Example 4.3.3. Let us denote $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ the labeled event structure for 2-bounded counter initialized by 1, i.e. 2-$\mathcal{BC}^1$, which is isomorphic to $(\{a\}, a, 2)$-$\mathcal{CP}$ for bounded FIFO channels (see Definition 3.3.13 on page 38). And let us take the same well-preorder $\preccurlyeq^{\mathcal{C}}$ as in Example 4.2.13, that is $C \preccurlyeq^{\mathcal{C}} C'$ if $\mathcal{M}(C) = \mathcal{M}(C')$ and $|C| \geq |C'|$. Figure 4.6-a and Figure 4.6-b illustrate $\mathcal{E}$ and its truncations with respectively the local cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$ and the global one $(\preccurlyeq^{\mathcal{C}}, \mathcal{C}_{\mathcal{E}})$.

As shown in Figure 4.6-a, events e^2_- and e^2_+ are cutoff events due to e^1_- and e_+1. And more over, these two events form the minimal set w.r.t. causality of cutoff events, i.e. $\mathsf{Min}_{\leq}(E^c_{\mathcal{E}}) = \{e^2_-, e^2_+\}$, hence give the intuitive frontier/bound of the truncation. However, in Figure 4.6-b, e^2_- and e^2_+ are both "outer events" because its local configurations are outer ones due to configuration $C_{cut} = \{e^1_-, e^1_+\}$ which is not local. One can say

that the truncation with global cutting context, in this example is $\{e^1_-, e^1_+\}$ is smaller than the one with local cutting context, $\{e^1_-, e^1_+, e^2_-, e^2_+\}$, while they preserve the same information on reachability/marking set, i.e.

$$\bigcup_{C\in\mathcal{C}_\mathcal{E}:C\subseteq\mathcal{T}(\mathcal{E},\preccurlyeq^\mathcal{C},\mathcal{C}^l_\mathcal{E})} \mathcal{M}(C) = \{0,1,2\} = \bigcup_{C\in\mathcal{C}_\mathcal{E}:C\subseteq\mathcal{T}(\mathcal{E},\preccurlyeq^\mathcal{C},\mathcal{C}_\mathcal{E})} \mathcal{M}(C)$$

Lemma 4.3.4. *Let $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ be a preordered labeled event structure where $\preccurlyeq^\mathcal{M}$ is a marking preorder. Let $\trianglelefteq$ be the adequate preorder based on configuration size, i.e. $C \trianglelefteq C'$ iff $|C| \leq |C'|$. If $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ has reflexive compatibility then $(\mathcal{E}, \preccurlyeq^\mathcal{M} \Cap \trianglerighteq)$ is also a preordered labeled event structure with strict and reflexive compatibility.*

Proof. Let us denote $\preccurlyeq^\mathcal{C} = (\preccurlyeq^\mathcal{M} \Cap \trianglerighteq)$. The reflexivity and transitivity of $\preccurlyeq^\mathcal{C}$ are trivial so that $\preccurlyeq^\mathcal{C}$ is a preorder on $\mathcal{C}_\mathcal{E}$. Let C, C' be two configurations in $\mathcal{C}_\mathcal{E}$ and assume that $C \preccurlyeq^\mathcal{C} C'$. Since $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ is a preordered labeled event structure, by Definition 4.2.1, for every extension set X of C, there exists an extension set X' of C' such that $(C \cup X) \preccurlyeq^\mathcal{M} (C' \cup X')$. It follows from the reflexive compatibility that $|X'| = |X|$. Because $C \preccurlyeq^\mathcal{C} C'$, we have $|C| \geq |C'|$ and thus $|C \cup X| = |C| + |X| = |C| + |X'| \geq |C'| + |X'| = |C' \cup X'|$ (recall that a configuration and its extension set are disjoint sets). Therefore $(C \cup X) \preccurlyeq^\mathcal{C} (C' \cup X')$ by definition and $(\mathcal{E}, \preccurlyeq^\mathcal{C})$ is preordered with reflexive compatibility because $|X| = |X'|$. Moreover the strict compatibility follows directly from the definition of $(\preccurlyeq^\mathcal{M} \Cap \trianglerighteq)$ which says that $C \prec^\mathcal{C} C'$ iff $C \preccurlyeq^\mathcal{M} C'$ and $C \rhd C'$. □

In the proof above, there are no constraint on labels of events and extension events. However, with a little modification, one can also say that the preorder based on the lexicographic order preserves strong compatibility, i.e. when there is no event labeled by an internal action in Σ^τ.

Lemma 4.3.5. *Given a preordered labeled event structure $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ where $\preccurlyeq^\mathcal{M}$ is a marking preorder, if $(\mathcal{E}, \preccurlyeq^\mathcal{M})$ has strong compatibility then $(\preccurlyeq^\mathcal{M} \Cap \trianglerighteq_l, \mathcal{E})$ [2] is also a preordered labeled event structure with strict and strong compatibility.*

Proof. By the same manner as in the proof of Lemma 4.3.4, this lemma is a consequence that comes from a property of the lexicographic order $\preccurlyeq_l$ corresponding to $\trianglelefteq_l$, i.e. $C \trianglelefteq_l C'$ if $\mathcal{L}(C) \preccurlyeq_l \mathcal{L}(C')$. In fact, for all multisets A, B, C over $\Sigma = \mathsf{Codom}(\mathcal{L})$, we have that $A \preccurlyeq_l B$ implies $(A \oplus C) \preccurlyeq_l (B \oplus C)$ where the operator $\oplus$ represents the union of multisets. □

The lexicography-based order $\trianglelefteq_l$ refines the size-based order $\trianglelefteq$, that means for all configurations $C, C' \in \mathcal{C}_\mathcal{E}$, $C \trianglelefteq_l C'$ implies $C \trianglelefteq C'$. And they both refine the inclusion order $\subseteq$. Hence, $(\preccurlyeq^\mathcal{M} \Cap \trianglerighteq, \mathcal{C})$ and $(\preccurlyeq^\mathcal{M} \Cap \trianglerighteq_l, \mathcal{C})$ are cutting contexts for any family $\mathcal{C} \subseteq \mathcal{C}_\mathcal{E}$ if the compatibility is satisfied. Moreover, if the corresponding truncations are finite, one can easily find out that $\mathcal{T}(\mathcal{E}, \preccurlyeq^\mathcal{M} \Cap \trianglerighteq_l, \mathcal{C})$ is generally smaller and never greater than $\mathcal{T}(\mathcal{E}, \preccurlyeq^\mathcal{M} \Cap \trianglerighteq, \mathcal{C})$, as the word "refine" means. Esparza *et al.* have given an example in [ERV96] showing that the truncation, obtained by using $\trianglelefteq$ is exponential while the one that uses $\trianglelefteq_l$ is linear w.r.t. the size of the original system.

Lemma 4.3.6. *Given a preorder labeled event structure $(\mathcal{E}, \preccurlyeq^\mathcal{M})$. If $\mathcal{E}$ is finitely-branching then $\preccurlyeq^\mathcal{C} = (\preccurlyeq^\mathcal{M} \Cap \supseteq)$ is a converse well-foundeded.*

[2] $\trianglelefteq_l$ is lexicographic labeling order (see Definition 4.2.17 on page 78).

Proof. As a consequence of König's lemma, the finitely-branching property implies that there is no infinite sequence of configuration $C_1 \supset C_2 \supset \ldots$. Hence there is no infinite increasing sequence of configurations w.r.t. the order $\preccurlyeq^{\mathcal{C}} = (\preccurlyeq^{\mathcal{M}} \Cap \supseteq)$. □

Remark: If $\mathcal{E}$ is finitely-branching and $\preccurlyeq^{\mathcal{M}}$ is a converse well-preorder then $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$ as well as $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq_l)$ is a converse well-preorder.

4.3.2 Coverability and quasi-liveness

Definition 4.3.7 (Coverability)**.** Given a labeled transition system $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ and a preorder $\preccurlyeq$ on its state space S, the *coverability problem* is to decide whether a state s is *covered* by some reachable state s', i.e. $s \preccurlyeq s'$ and $s^0 \stackrel{\sigma}{\rightarrow\!\!\!\!\!\rightarrow} s'$ for some $\sigma \in \Sigma^*$.

This problem may be solved by the computation of the downward closure w.r.t. $\preccurlyeq$ of $\mathsf{post}^*_{\mathcal{LTS}}$. While using labeled event structures $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ for modeling systems by means of induced labeled transition systems $(\mathcal{LTS}^{\mathcal{E}}, \preccurlyeq)$, one needs, and this is enough, to compute the downward closure w.r.t. $\preccurlyeq$ of $\mathcal{M}(\mathcal{C}_{\mathcal{E}}) = \bigcup_{C \in \mathcal{C}_{\mathcal{E}}} \mathcal{M}(C)$. Because, by Definition 3.2.4, $\mathcal{M}(\mathcal{C}_{\mathcal{E}}) = \mathsf{post}^*_{\mathcal{LTS}^{\mathcal{E}}}$.

Notation 4.3.8. Let $\mathcal{T}$ be the truncation of a preordered labeled event structure $(\mathcal{E}, \preccurlyeq^{\mathcal{C}})$ w.r.t. some cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C})$. We denote $\mathcal{C}_{\mathcal{T}}$ the family of configurations in $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})$, i.e. $\mathcal{C}_{\mathcal{T}} = \{C \in \mathcal{C}_{\mathcal{E}} \,/\, C \subseteq \mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C})\}$.

In this subsection, we will show that the coverability problem is decidable if there is a reflexive/strong compatibility by using either $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$ or $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq_l)$. As stated by Lemma 4.3.4 and Lemma 4.3.5, both $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq, \mathcal{C}_{\mathcal{E}})$ and $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq_l, \mathcal{C}_{\mathcal{E}})$ are local cutting contexts.

Lemma 4.3.9. *For any finitely-branching and preordered labeled event structure $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ with reflexive (strong) compatibility, let $\trianglelefteq$ be the size-based adequate order (the lexicography-based adequate order, resp.) and $\mathcal{T}(\mathcal{E}, (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}})$ be the corresponding local truncation. Then we have:*

$$\succcurlyeq(\mathcal{M}(\mathcal{C}_{\mathcal{T}})) = \succcurlyeq(\mathsf{post}^*_{\mathcal{LTS}^{\mathcal{E}}})$$

Proof. By definition of induced labeled transition systems (Definition 3.2.4), we have to simply prove that $\succcurlyeq(\mathcal{M}(\mathcal{C}_{\mathcal{T}})) = \succcurlyeq(\mathcal{M}(\mathcal{C}_{\mathcal{E}}))$. Since $\mathcal{C}_{\mathcal{T}} \subseteq \mathcal{C}_{\mathcal{E}}$, it suffices to show that $\mathcal{M}(\mathcal{C}_{\mathcal{E}}) \subseteq (\succcurlyeq(\mathcal{M}(\mathcal{C}_{\mathcal{T}})))$. It follows from Lemma 4.3.6 that the order $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$ is converse well-founded. Let C be any configuration in $\mathcal{C}_{\mathcal{E}}$, by Theorem 4.2.14, there exists a configuration $C' \in \mathcal{C}^n_{\mathcal{E}} \subseteq \mathcal{C}_{\mathcal{T}}$ such that $C \prec^{\mathcal{C}} C'$ where $\preccurlyeq^{\mathcal{C}} = (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$, and so $C \preccurlyeq^{\mathcal{M}} C'$. It follows from the definition of marking preorders (Definition 4.2.2) that $\mathcal{M}(C) \subseteq (\succcurlyeq(\mathcal{M}(C')))$. Therefore, by a same manner as for strong compatibility with lexicography-based adequate order, one can conclude that $(\succcurlyeq(\mathcal{M}(\mathcal{C}_{\mathcal{T}}))) = (\succcurlyeq(\mathcal{M}(\mathcal{C}_{\mathcal{E}}))) = (\succcurlyeq(\mathsf{post}^*_{\mathcal{LTS}^{\mathcal{E}}}))$. □

The downward closure set for coverability may be rewritten in another way as

$$\begin{aligned}\left(\succcurlyeq(\mathcal{M}(\mathcal{C}_{\mathcal{T}}))\right) &= \left(\succcurlyeq\left(\mathcal{M}(\mathsf{Max}_{\preccurlyeq^{\mathcal{M}}}(\mathcal{C}_{\mathcal{T}}))\right)\right) \\ &= \left(\succcurlyeq\left(\bigcup_{C \in \mathsf{Max}_{\preccurlyeq^{\mathcal{M}}}(\mathcal{C}_{\mathcal{T}})} \mathcal{M}(C)\right)\right)\end{aligned}$$

In words, one takes the set of maximal configurations, w.r.t. the marking preorder $\preccurlyeq^{\mathcal{M}}$, in the truncation; then one computes the union of their markings and finally one computes its downward closures.

The correctness of this lemma is based on the one of Theorem 4.2.14 so that the strict compatibility of the cutting context's order $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$, as well as the strict comparison in definition of cutoff configurations (see Definition 4.2.10) is very important for the completeness of the truncation while verifying coverability. One can find a counter-example in [ERV96]. For example, if we replace the condition $C_{cut} \prec^{\mathcal{C}} C$ by $C_{cut} \preccurlyeq^{\mathcal{C}} C$, for some preordered labeled even structure, $(C \cup C_{cut})$ is an outer configuration and the truncation may have no configuration C' which covers $(C \cup C_{cut})$, i.e. $(C \cup C_{cut}) \not\preccurlyeq^{\mathcal{C}} C'$ and $(C \cup C_{cut}) \not\preccurlyeq^{\mathcal{M}} C'$.

Remark: Our covering problem corresponds to the sub-covering one in [FS01, HST07] by duality. The sub-covering problem is decidable for reflexive and downward well-preorder labeled event structures $(\mathcal{LTS}, \preccurlyeq)$ by means of computing upward closure of $\mathsf{post}^*_{\mathcal{LTS}}$. The *downward compatibility* tells that from a smaller state, one can do the same thing as from a greater one w.r.t. $\preccurlyeq$. In this work, we give no downward notion for compatibility. However, such systems corresponds to our well-preordered labeled transition systems $(\mathcal{LTS}, \succcurlyeq)$ where the converse preorder $\succcurlyeq$ is converse well-preordered.

Lemma 4.1.14 combined with Lemma 4.3.9 allows us to reduce backward analysis to forward analysis: to compute $\mathsf{pre}^*_{\mathcal{LTS}^{\mathcal{E}}}(\preccurlyeq(\{s\}))$ in a well-preordered labeled transition $(\mathcal{LTS}^{\mathcal{E}}, \preccurlyeq)$ and finite pred-basis pb, it is sufficient to build the finite truncation of the corresponding well-preordered labeled event structure $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$.

Lemma 4.3.10. *Let $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ be any preordered labeled event structure with reflexive (strong) compatibility and $\trianglelefteq$ be the size-based adequate order (the lexicography-based adequate order, resp.). If $\mathcal{E}$ is finitely-branching then $\mathcal{T}(\mathcal{E}, (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}})$ is finite.*

Proof. Thanks to Lemma 4.3.6, the preorder $(\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$ is a converse well-founded. And as a sequence of Theorem 4.2.15, the truncation is finite. □

As seen in Chapter 5, one can obtain the prefix based on the local cutting context $(\mathcal{E}, (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}})$ by a partial-order construction. Although the covering problem is decidable on $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ due to the finiteness of the truncation, one need compute more or less all markings as well as possible configurations (generally not local ones) in the prefix. This computation after the construction does not suit with partial-order verification.

Definition 4.3.11 (Quasi-liveness)**.** Let $\mathcal{LTS} = (S, \Sigma, s^0, \rightarrow)$ be a labeled transition system. An action $a \in \Sigma$ is *quasi-live* if there is an execution of $\mathcal{LTS}$ in which a is fired.

In the context of labeled event structures, the quasi-liveness of a reduces to the existence of an event labeled by a. Unlike the covering problem, the quasi-liveness would be verified in concurrent with partial-order truncation's construction, and fortunately, without computing configurations that are possibly not local.

Theorem 4.3.12. *Let $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ be any finitely-branching and preordered labeled event structure with reflexive (strong) compatibility and $\trianglelefteq$ be the size-based adequate order (the lexicography-based adequate order, resp.). For any global action $a \in \Sigma$, a is quasi-live in $\mathcal{LTS}^{\mathcal{E}}$ iff a labels an event in $\mathcal{T}(\mathcal{E}, (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}})$.*

Proof. Let us denote by $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ the labeled event structure. By definition of induced labeled transition systems (Definition 3.2.4 on page 28), it suffices to prove that

$\mathcal{L}(\mathcal{T}(\mathcal{E}, (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}})) = \Sigma$ (*) (recall that $\Sigma = \mathsf{Codom}(\mathcal{L})$). Since $\mathcal{T}(\mathcal{E}, (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}}) \subseteq E$, the left-side inclusion of (*) is obvious. For the right-side inclusion, suppose that e be any event $e \in E$. Thanks to Theorem 4.2.14, for the configuration $>(e)$, there exists another configuration $C \in \mathcal{C}^n_{\mathcal{E}}$ such that $\langle >(e), C\rangle \in (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$. Since a is not an internal action, i.e. $a \notin \Sigma^\tau$, because of the reflexive compatibility of $(\mathcal{E}, \preccurlyeq^{\mathcal{C}})$ where $\preccurlyeq^{\mathcal{C}} = (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$, we have $C \vdash e'$ for some extension event e' satisfying $\mathcal{L}(e') = \mathcal{L}(e)$. Moreover, it follows from the downward closed property of the configuration $(C \cup \{e'\})$ that $>(e') \subseteq C$, and as a consequence $>(e')$ does not contain cutoff events. Thanks to Lemma 4.3.2, $e' \in \mathcal{T}(\mathcal{E}, (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}})$. The right-side inclusion of (*) is proved. □

4.3.3 Termination and boundedness

Definition 4.3.13 (Termination)**.** Given a labeled transition system $\mathcal{LTS}$, we say that $\mathcal{LTS}$ *terminates* if $\mathcal{LTS}$ has no infinite execution.

Suppose that we has already a labeled event structure $\mathcal{E}$ for $\mathcal{LTS}$. Then $\mathcal{LTS}$ terminates if $\mathcal{E}$ is finite. Or reversely, $\mathcal{LTS}$ does not terminate if $\mathcal{E}$ has no bound on its configurations' sizes. In order to resolve the termination problem, we use the cutting context based on $\preccurlyeq^{\mathcal{C}} = (\succcurlyeq^{\mathcal{M}} \Cap \supseteq)$. The intuitive idea is that if a configuration C_{cut} is cut due to another configuration C, the compatibility of $\preccurlyeq^{\mathcal{C}}$ implies the existence of infinite sequence of configurations $C_{cut}, C'_{cut}, C''_{cut}, \ldots$ in $\mathcal{C}_{\mathcal{E}}$. An important thing here is that this sequence is increasing while comparing elements' sizes. The reasoning that can be found in following proofs, bases on both C and C_{cut} (1); and the fact that C_{cut} may be obtained from C (2). The first point really differs from the reasoning for coverability and liveness problems in which C_{cut} is somehow useless. And due to the second point, one has that $C \subseteq C'$ so that the adequate order $\subseteq$ is naturally convenient for the termination problem.

Lemma 4.3.14. *Given a preordered labeled event structure $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ where $\preccurlyeq^{\mathcal{M}}$ is a marking preorder, $(\mathcal{E}, (\succcurlyeq^{\mathcal{M}} \Cap \supseteq))$ is also a preordered labeled event structure.*

Proof. Let us denote $\preccurlyeq^{\mathcal{C}} = (\succcurlyeq^{\mathcal{M}} \Cap \supseteq)$. The reflexivity and transitivity of $\preccurlyeq^{\mathcal{C}}$ are trivial so that $\preccurlyeq^{\mathcal{C}}$ is a preorder on $\mathcal{C}_{\mathcal{E}}$. The compatibility of preorder $\preccurlyeq^{\mathcal{C}}$ is directly inherited from the inclusion order. Let C, C' be two configurations in $\mathcal{C}_{\mathcal{E}}$ and assume that $C \preccurlyeq^{\mathcal{C}} C'$. Let X be any extension set of C, i.e. $C \Vdash X$. Since $C \supseteq C'$, we have thus $C' \subseteq (C \cup X)$. Hence, $X' = (C \setminus C') \cup X$ is simply an extension set of C' which guaranties the compatibility of $\preccurlyeq^{\mathcal{C}}$ because $C' \cup X' = C \cup X$. □

As seen in the previous proof, $(\mathcal{E}, (\succcurlyeq^{\mathcal{M}} \Cap \supseteq))$ may not have the strict compatibility needed in Theorem 4.2.14 so that the corresponding local truncation is not complete in view of coverability. However, this truncation preserves enough information for terminating problem. And the verification is decidable due to its finiteness.

Theorem 4.3.15. *Let $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ be any well-preordered labeled event structure. If $\mathcal{E}$ is finitely-branching then $\mathcal{T}(\mathcal{E}, (\succcurlyeq^{\mathcal{M}} \Cap \supseteq), \mathcal{C}^l_{\mathcal{E}})$ is finite.*

Proof. Suppose that the truncation $\mathcal{T}(\mathcal{E}, (\succcurlyeq^{\mathcal{M}} \Cap \supseteq), \mathcal{C}^l_{\mathcal{E}})$ is infinite. Thanks to König's lemma, it follows from the finitely-branching property of $\mathcal{E}$ that the truncation $\mathcal{T}$ contains an infinite sequence of events $e_1 \lessdot e_2 \lessdot \ldots$. As a consequence, $(\geq(e)) \subset (\geq(e_j))$ for all $i < j$. Moreover, since $\preccurlyeq^{\mathcal{M}}$ is well-preordered, by definition, there exist two indices $i < j$ such that $(\geq(e_i)) \preccurlyeq^{\mathcal{M}} (\geq(e_j))$. We obtain that the local configuration $\geq(e_j)$ is a cutoff one, and e_j is a cutoff event due to e_i, or more precisely, due to the local configuration

$\geq(e_i)$. Hence, e_{j+1} is an outer event by definition. The fact that $\mathcal{T}(\mathcal{E}, (\succcurlyeq^{\mathcal{M}} \Cap \supseteq), \mathcal{C}^l_{\mathcal{E}})$ contains e_{j+1} contradicts to Lemma 4.3.2. Therefore, the truncation is finite. □

In covering-based verification, cutoff events as well as cutoff configurations are somehow useless (see Theorem 4.2.14). However, for termination, it is the opposite. The existence of cutoff events in the truncation is enough for deciding whether the corresponding system terminates if there is transitive compatibility.

Theorem 4.3.16 (Termination)**.** *For any well-preordered finitely-branching labeled event structure $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ with transitive compatibility, $\mathcal{LTS}^{\mathcal{E}}$ terminates iff $\mathcal{T}(\mathcal{E}, \succcurlyeq^{\mathcal{M}} \Cap \supseteq, \mathcal{C}^l_{\mathcal{E}})$ contains no cutoff event.*

Proof. Let us denote the labeled event structure by $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, and the local truncation simply by $\mathcal{T}$. Thanks to Theorem 4.3.15, $\mathcal{T}$ is finite. Recall that $\mathcal{T} = E^n \cup \mathsf{Min}_{\leq}(E^c)$, the truncation contains a cutoff event iff there exists a cutoff event in E.

($\Leftarrow$) Assume that $\mathcal{T}$ contains no cutoff event. This implies that there is no outer event and by definition, $E = \mathcal{T}$. The event set E is also finite. We deduce from Lemma 3.2.12 that every execution in $\mathcal{LTS}^{\mathcal{E}}$ which corresponds to linearisation w.r.t. causality of some configuration has length at most $|E|$. Therefore, $\mathcal{LTS}^{\mathcal{E}}$ terminates.

($\Rightarrow$) Assume that $\mathcal{T}$ contains a cutoff event e_{cut}. There exists another event such that $e < e_{cut}$ and $(\geq(e)) \preccurlyeq^{\mathcal{M}} (\geq(e_{cut}))$. Due to the transitive compatibility of $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$, let $X = ((\geq(e_{cut})) \setminus (\geq(e)))$, since X is an extension set of $\geq(e)$ there exists another extension set X' of $\geq(e_{cut})$ such that $(\geq(e) \cup X) = (\geq(e_{cut})) \preccurlyeq^{\mathcal{M}} ((\geq(e_{cut})) \cup X')$ and $|X'| \geq |X| > 0$. By iterating this reasoning, we obtain an infinite sequence of configurations $C \preccurlyeq^{\mathcal{M}} (C \cup X) \preccurlyeq^{\mathcal{M}} (C \cup X \cup X') \preccurlyeq^{\mathcal{M}} \ldots$ so that there is no bound for their size. The induced labeled transition system $\mathcal{LTS}^{\mathcal{E}}$ must have an infinite execution due to Lemma 3.2.12. □

Definition 4.3.17 (Boundedness)**.** A labeled transition system $\mathcal{LTS}$ is *bounded* if it has a finite reachability set, i.e. $\mathsf{post}^*_{\mathcal{LTS}}$ is finite.

In order to decide *boundedness*, we use the same local cutting context as in termination, i.e. based on $(\succcurlyeq^{\mathcal{M}} \Cap \supseteq)$, but we need the notion of *marking-strict* cutoff events.

Definition 4.3.18 (Marking-strict cutoff event)**.** Given a preordered labeled event structure $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ and its local cutting context $(\succcurlyeq^{\mathcal{M}} \Cap \supseteq, \mathcal{C}^l_{\mathcal{E}})$. A *marking-strict* cutoff event is any event e_{cut} such that $\mathcal{M}(\geq(e_{cut})) \succ^{\mathcal{M}} \mathcal{M}(\geq(e))$ and $e_{cut} < e$ for some event e.

It is worth noticing that, in practice, e_{cut} may be a marking-strict cutoff event due to the empty configuration $\emptyset \in \mathcal{C}^l$. We simply say that it is due to the particular event ε. Observe that any marking-strict cutoff event is also a cutoff event. The idea of verifying boundedness is closed to the one of termination. Their corresponding truncations is the same since we use a local cutting context $((\succcurlyeq^{\mathcal{M}} \Cap \supseteq, \mathcal{C}^l_{\mathcal{E}})$. However, the decidability of boundedness verification depends on a strict compatibility of marking preorder $\preccurlyeq^{\mathcal{M}}$, not the strict compatibility of the cutting preorder $(\succcurlyeq^{\mathcal{M}} \Cap \supseteq)$.

Theorem 4.3.19 (Boundedness)**.** *Given $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$ be any well-preordered labeled event structure with transitive and strict compatibility, and $\preccurlyeq^{\mathcal{M}}$ is a partial-order. If $\mathcal{E}$ is coherent and finitely-branching then its induced labeled transition system $\mathcal{LTS}^{\mathcal{E}}$ is bounded iff $\mathcal{T}(\mathcal{E}, \succcurlyeq^{\mathcal{M}} \Cap \supseteq, \mathcal{C}^l_{\mathcal{E}})$ contains no marking-strict cutoff event and $\mathcal{M}(\mathcal{C}_{\mathcal{T}})$ is finite.*

Proof. Let us denote the labeled event structure by $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ and its local truncation simply by $\mathcal{T}$. By definition of induced labeled event structures (Definition 3.2.4), we have $\mathsf{post}^*_{\mathcal{LTS}^{\mathcal{E}}} = \mathcal{M}(\mathcal{C}_{\mathcal{E}})$. Hence, the theorem could be rewritten as: $\mathcal{M}(\mathcal{C}_{\mathcal{E}})$ is finite (1) iff $\mathcal{T}$ contains no marking-strict cutoff event and $\mathcal{M}(\mathcal{C}_{\mathcal{T}})$ is finite (2). We are going to first show that "not (2) implies not (1)" and second "(2) implies (1)". Therefore, one can thus conclude the theorem.

- Since $\mathcal{T} \subseteq E$ and $\mathcal{C}_{\mathcal{T}} \subseteq \mathcal{C}_{\mathcal{E}}$, the marking set $\mathcal{M}(\mathcal{C}_{\mathcal{E}})$ which includes $\mathcal{M}(\mathcal{C}_{\mathcal{T}})$ must be infinite if $\mathcal{M}(\mathcal{C}_{\mathcal{T}})$ is infinite. Now, suppose that $\mathcal{T}$ contains a marking-strict cutoff event e_{cut} due to another event e, that means $e < e'$ and $\mathcal{M}(e) \prec^{\mathcal{M}} \mathcal{M}(e_{cut})$. Due to the strict and transitive compatibility of $(\mathcal{E}, \preccurlyeq^{\mathcal{M}})$, by the same manner as in the proof of Theorem 4.3.16, there must exists an infinite sequence of configuration $C_1 \prec^{\mathcal{M}} C_2 \prec^{\mathcal{M}} C_3 \prec^{\mathcal{M}} \ldots$ in $\mathcal{C}_{\mathcal{E}}$ where $C_1 = \geq(e)$ and $C_2 = \geq(e_{cut})$. And their markings are all belongs to $\mathcal{M}(\mathcal{C}_{\mathcal{E}})$ so that $\mathcal{M}(\mathcal{C}_{\mathcal{E}})$ is infinite.

- We will prove that $\mathcal{M}(\mathcal{C}_{\mathcal{E}}) \subseteq \mathcal{M}(\mathcal{C}_{\mathcal{T}})$ if there is no marking-strict cutoff event in $\mathcal{T}$. As a consequence, $\mathcal{M}(\mathcal{C}_{\mathcal{E}})$ is finite due to the finiteness of $\mathcal{M}(\mathcal{C}_{\mathcal{T}})$. Let $\mathcal{C}^o_{\mathcal{T}}$ denote the set of configurations of which marking does not belong to $\mathcal{M}(\mathcal{C}_{\mathcal{T}})$, i.e. $\mathcal{C}^o_{\mathcal{T}} = \{C \in \mathcal{C}_{\mathcal{E}} \,/\, (\mathcal{M}(C) \setminus \mathcal{M}(\mathcal{C}_{\mathcal{T}})) \neq \emptyset\}$. Suppose that $\mathcal{C}^o_{\mathcal{T}}$ is not empty. Since $\mathcal{E}$ is finitely-branching, the size-based order $\trianglelefteq$ on $\mathcal{C}_{\mathcal{E}}$ is founded so that one can choose a minimal configuration $C \in \mathcal{C}^o_{\mathcal{T}}$ w.r.t. $\trianglelefteq$. It obviously follows from definition that C must contain an outer event, and thus as a consequence, a cutoff event e_{cut} due to another event e. Without lost of generality, suppose that e_{cut} is minimal w.r.t. causality $\leq$ in C, then e_{cut} is also minimal w.r.t. $\leq$ over E because C is downward closed. As a consequence of Lemma 4.3.2, e_{cut} belongs to the truncation $\mathcal{T}$.

  Because there is no marking-strict cutoff event in $\mathcal{T}$ as supposed, for cutoff event e_{cut}, we have thus $e_{cut} > e$ (causality), $\geq(e_{cut}) \succcurlyeq^{\mathcal{M}} \geq(e)$ and $\geq(e_{cut}) \not\succ^{\mathcal{M}} \geq(e)$ (marking). Therefore $\geq(e_{cut}) = \geq(e)$ since $\preccurlyeq^{\mathcal{M}}$ is a partial order. Let $X = C \setminus (\geq(e_{cut})$, as a sequence of the second property of coherence (Definition 3.2.11) of $\mathcal{E}$, it follows from the extension set X of $\geq(e_{cut})$ that there exists an extension set X' of configuration $\geq(e)$ satisfying $|X'| = |X|$ and $\mathcal{M}(\geq(e_{cut}) \cup X) = \mathcal{M}(\geq(e) \cup X')$. The configuration $C' = \geq(e) \cup X'$ has the same marking as C, hence $C' \in \mathcal{C}^o_{\mathcal{T}}$. Moreover,

$$|C'| = |\geq(e)| + |X'| = |\geq(e)| + |X| < |\geq(e_{cut})| + |X| = |C|$$

  This fact contradict to the minimality of chosen configuration C. Therefore, $\mathcal{C}^o_{\mathcal{T}}$ must be empty so that $\mathcal{M}(\mathcal{C}_{\mathcal{E}})$ is finite.

□

Remark: Abdulla *et al.* have given an unfolding algorithm for symbolic verification of unbounded Petri nets in [AIN00]. They adapt an algorithm described in [ACJT96] for backward reachability analysis. This technique is more or less the dual of the ours. One can also find another work on boundedness of Petri nets which based on forward analysis in [DJN04]. In both cases, their algorithms operate on constraints that each configuration may represent an (infinite) upward closed set of Petri nets' markings [3].

[3] A marking in Petri nets is a bit different to our marking for nondeterministic labeled event structures (see Definition 2.5.1 on page 20).

Chapter 5

Compositional unfolding techniques

Contents

Our goal in this chapter is to give algorithms, called *unfolding algorithms*, for building labeled event structures using a method similar to Petri net unfolding. The first Petri net unfolding algorithm was given by McMillan [McM95a]. This algorithm intuitively enlarges some prefix of a labeled occurrence net (see Section 2.5 on page 19) by iteratively adding events to it. A new event is computed from existing conditions in the prefix that possibly enable this event. This idea was later applied to synchronized products of transition systems [ERV96]. In the result: for adding events, component states could be analyzed without focusing global states of the synchronized product. In our case, we adapt this technique to synchronized products of label event structures. One constructs not only prefixes of a synchronized product but also corresponding component prefixes together.

We present the general algorithm in Section 5.1. Intuitively, every event is created and inserted into the being constructed prefix when all its direct predecessors are already there. Then, we detail our unfolding algorithm into two particular cases: for component labeled event structures and for synchronized products of labeled event structures.

First, in each standard labeled event structure defined in Section 3.3, an event and its direct predecessors as well as its direct successors form somehow a motif. For example, in the k-causality process, every increment event has k increment direct successors and one decrement one. As a consequence, based on the definition of a labeled event structure,

one can write a corresponding algorithm to construct it, more precisely, to construct its prefixes. Section 5.2 will give algorithms, for instance, building causality processes for counters and FIFO channels.

Second, in a synchronized product of labeled event structures, a global event is nothing but a synchronization of component events. The unfolding algorithm, in the one hand, successively extends a prefix of the synchronized product, and in the other hand, uses associated unfolding algorithms for components in order to accordingly extend its component prefixes. This algorithm will be detailed in Section 5.3. Moreover, one can consider a synchronized product of labeled event structures as a component of another larger one. As a consequence, one can obtain a global unfolding algorithm for a complex system which is hierarchically modeled by means of labeled event structures. Our contribution is not only a generalization of the unfolding method in [ER99] to parallel composition of labeled event structures, but also gives an algorithm which is capable of exploiting concurrency in components as well as among them.

In Section 5.4, we will explain how to integrate truncating criterion into an unfolding algorithm in order to only construct finite truncations which are complete for certain verification problems given in Chapter 4.

5.1 Unfolding algorithm

Aiming at building a labeled event structure $(E, \leq, \#, \mathcal{L}, \mathcal{M})$, the unfolding algorithm always maintains a prefix of $\mathcal{E}$, w.r.t. isomorphism, and tries to extend it until it is impossible. This prefix under construction is presented by so called *structure variables* $\widehat{\mathcal{E}} = (\widehat{E}, \widehat{\leq}, \widehat{\#}, \widehat{\mathcal{L}}, \widehat{\mathcal{M}})$ that are the main variables in our algorithms. Extending the prefix $\widehat{\mathcal{E}}$ intuitively means that the unfolding algorithm creates new events, adds them to $\widehat{E}$, and accordingly modifies other structure variables, e.g. $\widehat{\leq}, \widehat{\#}, \widehat{\mathcal{L}}, \widehat{\mathcal{M}}$, so that the obtained $\widehat{\mathcal{E}}$ is still a prefix of $\mathcal{E}$ w.r.t. isomorphism.

Remark: In this chapter, for simplicity of proofs, when two labeled event structures are isomorphic w.r.t. some bijection $\mathcal{B}$, we assume that they have a same set of events. On this understanding, one does not need to take care of bijection $\mathcal{B}$ without risk of confusion. Due to this assumption, one can say that $\widehat{\mathcal{E}}$ is a prefix of $\mathcal{E}$ and simply write $\widehat{\mathcal{E}} = \mathcal{E}|_{\widehat{E}}$. That means $\widehat{E}$ is a downward-closed subset w.r.t. $\leq$ of E. And in this case, the causality $\widehat{\leq}$ and the conflict relation $\widehat{\#}$ are respectively restrictions of $\leq$ and $\#$ onto the event set $\widehat{E}$, i.e. $\widehat{\leq} = \leq|_{\widehat{E}}$ and $\widehat{\#} = \#|_{\widehat{E}}$.

To be precise, notice that $\widehat{\mathcal{E}}$ constructed by our algorithm will not be a prefix of $\mathcal{E}$ as in Definition 3.1.12 but rather an isomorphic copy of it. In this chapter, we will never talk about this isomorphism though, and always think of $\widehat{\mathcal{E}}$ as a sub-event structure of $\mathcal{E}$

Algorithm 5.1 represents the pseudo-code of our general unfolding algorithm. Besides structure variable $\widehat{\mathcal{E}}$ representing the prefix being constructed, this algorithm maintains a variable PE, called *possible extensions*. PE contains a set of events in $\widehat{E}$ from which the prefix $\widehat{\mathcal{E}}$ may be extended.

The algorithm starts by initializing the prefix $\widehat{\mathcal{E}}$ as well as PE using function Init (line 2). As the output of Init, $\widehat{\mathcal{E}}$ will be usually the prefix of $\mathcal{E}$ consisting of its minimal events, i.e. $\widehat{\mathcal{E}} = \mathcal{E}|_{\mathsf{Min}_{\leq}(E)}$. At the same time, PE will be the whole event set of $\widehat{\mathcal{E}}$, i.e. $\mathsf{PE} = \widehat{E}$. Then the algorithm proceeds by considering events in PE in turn. For a chosen event e in PE (line 4), it calls the function Extend that is the core of our unfolding algorithm. This function takes e as well as values of structure variable $\widehat{\mathcal{E}}$ and of possible extension

Algorithm 5.1: Unfolding algorithm

```
1 begin
2     (Ê, PE) := Init()
3     while PE ≠ ∅ do
4         take an event e in PE
5         (Ê, PE) := Extend(Ê, PE, e)
6     end while
7 end
```

PE as input (line 6), and does the following:

- finds which direct successors e' of e in $\mathcal{E}$, i.e. $e \lessdot e'$, should be added to $\widehat{E}$: Such successors e' must satisfy that its predecessors are not only in $\widehat{E}$, i.e. $>(e') \subseteq \widehat{E}$, but have also been previously extended, $>(e') \cap \mathsf{PE} = \emptyset$;
- adds such successors e' to $\widehat{E}$ and updates the labeled event structure $\widehat{\mathcal{E}}$ according to these new events; e is removed from PE while its successors is inserted into PE; and
- returns the new prefix $\widehat{\mathcal{E}}$ and the possible extension PE.

The two conditions stated in the first item are important. The first one ensures that adding successor e' does not break the downward-closure w.r.t. the causality $\leq$ of the obtained prefix. And the second one avoids duplication of e' when, for instance, extending another direct predecessor f of e', i.e. $f \lessdot e'$ and $f \neq e$, by calling $\mathsf{Extend}(f)$ afterwards. The unfolding algorithm repeats extending $\widehat{\mathcal{E}}$ by calling function Extend as long as the set PE is not empty (line 3).

Remark: The Algorithm 5.1 does not terminate if the labeled event structure $\mathcal{E}$ being constructed is infinite. In Section 5.4, we will introduce truncating criteria, and consequently, terminating algorithms that construct only finite prefixes.

Function Extend obviously depends on the labeled event structure $\mathcal{E}$ that we want to construct (see Section 5.2 and Section 5.3). However, it is possible to state general correctness criteria for the algorithm. We formulate them as the *unfolding invariant* (Definition 5.1.1) and *correctness criteria* (Definition 5.1.3). The unfolding invariant is guaranteed at any step in the unfolding algorithm including the inputs of Extend as well as the outputs of Init and Extend.

Definition 5.1.1 (Unfolding invariant)**.** $(\widehat{\mathcal{E}}, \mathsf{PE})$ is *correct* w.r.t. $\mathcal{E}$ if

*I*1. $\widehat{\mathcal{E}}$ is a prefix of $\mathcal{E}$, i.e. $\widehat{\mathcal{E}} = \mathcal{E}|_{\widehat{E}}$,
*I*2. PE is a subset of $\widehat{E}$, and
*I*3. for all $e \in E$, $\gtrdot(e) \subseteq (\widehat{E} \setminus \mathsf{PE})$ iff $e \in \widehat{E}$.

The property *I*3 determines which events should be in the prefix $\widehat{\mathcal{E}}$. Recall that for all event e, $>(e)$ is the downward-closure of $\gtrdot(e)$ w.r.t. the causality $\leq$. When $\widehat{\mathcal{E}}$ is a prefix of $\mathcal{E}$, its event set $\widehat{E}$ is a downward-closed set w.r.t. the causality $\leq$ of E. Hence $\gtrdot(e) \subseteq (\widehat{E} \setminus \mathsf{PE})$ means that direct predecessors of e are already extended, and so do all predecessors of e.

Lemma 5.1.2. *If* $(\widehat{\mathcal{E}}, \mathsf{PE})$ *is correct w.r.t.* $\mathcal{E}$ *then*

- $\mathsf{PE} \subseteq \mathsf{Max}_{\leq}(\widehat{E})$, *and*
- *for all* $e \in E$, $e \notin \widehat{E}$ *and* $\gtrdot(e) \subseteq \widehat{E}$ *implies that there exists* $e' \in \mathsf{PE}$ *satisfying* $e' \lessdot e$.

Proof. We will prove the first property by contradiction. Let e be an event in PE and suppose that e is not maximal in $\widehat{E}$ w.r.t. the causality $\leq$. There exists another event $e' \in \widehat{E}$ that is direct successor of e, i.e. $e \lessdot e'$. We have that $\gtrdot(e')$ contains $e \notin (\widehat{E} \setminus \mathsf{PE})$. This is in contradiction with the unfolding invariant $I3$ in Definition 5.1.1. Therefore, $\mathsf{PE} \subseteq \mathsf{Max}_{\leq}(\widehat{E})$.

The second property is a direct consequence of the right-to-left unfolding invariant $I3$ from Definition 5.1.1. □

As stated in Lemma 5.1.2, the set of possible extensions, PE, is always a subset of $\widehat{E}$ that contains only maximal events w.r.t. the causality. When extending from an event e, some of its successors are added to $\widehat{E}$ as well as to PE. Instructions of function Extend must somehow reestablish the invariant $I3$. For example, e should be removed from PE because it is no longer maximal w.r.t. the causality. The second item of Lemma 5.1.2 says that an event e can not be added to $\widehat{E}$ while some of its predecessors e' is not extended yet. It is regardless of the choice of e as input of Extend (line 4) that the order of extending events respects to the causality.

Definition 5.1.3 (Extend's correctness)**.** We say that a function Extend is *correct w.r.t. a given labeled event structure* $\mathcal{E}$ if, for all $(\widehat{\mathcal{E}}, \mathsf{PE})$ that are correct w.r.t. $\mathcal{E}$ (by Definition 5.1.1), and for all $e \in \mathsf{PE}$, the return value $(\widehat{\mathcal{E}}', \mathsf{PE}') = \mathsf{Extend}(\widehat{\mathcal{E}}, \mathsf{PE}, e)$ satisfies:

$C1$. $(\widehat{\mathcal{E}}', \mathsf{PE}')$ is correct w.r.t. $\mathcal{E}$,
$C2$. $\widehat{E} \subseteq \widehat{E}'$, and
$C3$. $\mathsf{PE}' = (\mathsf{PE} \setminus \{e\}) \cup (\widehat{E}' \setminus \widehat{E})$.

The property $C1$ requires that the unfolding invariant (Definition 5.1.1) is preserved. When Extend is correct, due to the property $C2$, it adds new events to the prefix without removing any existing event. The property $C3$ intuitively means that, while e is removed from the possible extension set PE, no new event is left out of PE.

For $k = 0, 1, \ldots$, let $\widehat{\mathcal{E}}_k = (\widehat{E}_k, \widehat{\leq}_k, \widehat{\#}_k, \widehat{\mathcal{L}}_k, \widehat{\mathcal{M}}_k)$ and PE_k denote respectively the values of the variables $\widehat{\mathcal{E}}$ and PE after k steps of the principal loop in Algorithm 5.1. Let e_k be the value of parameter e chosen at the k^{th} step of this loop. Then this unfolding algorithm satisfies following properties:

Proposition 5.1.4. *Given a labeled event structure* $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$*, if the output* $(\widehat{\mathcal{E}}_0, \mathsf{PE}_0) = \mathsf{Init}()$ *is correct w.r.t.* $\mathcal{E}$ *(Definition 5.1.1) and the function* Extend *is correct w.r.t.* $\mathcal{E}$ *(Definition 5.1.3), then*

1. $\widehat{\mathcal{E}}_0, \widehat{\mathcal{E}}_1, \ldots$ *is an increasing sequence of prefixes of* $\mathcal{E}$ *w.r.t. the inclusion order on its event sets, i.e.* $\widehat{E}_0 \subseteq \widehat{E}_1 \subseteq \ldots$;
2. *the order of extending events respects to the causality order, that means* $e_j \not\leq e_i$ *for all* $j > i$;
3. *for every extended event* e_k,

$$\widehat{E}_k \setminus \widehat{E}_{k-1} = \left\{ f \in \lessdot(e_k) \,/\, \gtrdot(f) \subseteq \left((\widehat{E}_{k-1} \setminus \mathsf{PE}_{k-1}) \cup \{e_k\} \right) \right\}.$$

Proof. We will prove these properties in the order in which they are stated.

1. The first property is obtained by induction on k. Because, in the base case, the output $(\widehat{E}_0, \mathsf{PE}_0)$ of the function $\mathsf{Init}()$ is correct w.r.t. $\mathcal{E}$. Moreover, the correctness of Extend w.r.t. $\mathcal{E}$ is enough for inductive step. The increasing order on event sets $\widehat{E}_0, \widehat{E}_1, \ldots$ is a direct consequence of the correctness condition $C2$.

2. Suppose that e_i is extend before e_j, i.e. $j > i$. It follows from the previous property that $E_{i-1} \subseteq E_{j-1}$. Consequently, E_{j-1} contains not only e_j but also e_i. Thanks to Lemma 5.1.2, e_j is maximal w.r.t. the causality in $\widehat{\mathcal{E}}_{j-1}$, and one thus reasons out that $e_j \not\preccurlyeq e_i$. Moreover, due to condition $C3$, Extend inserts only new events into the possible extension PE. Then, $e_i \notin \mathsf{PE}_i$, and moreover, $e_i \notin \mathsf{PE}_k$ for all $k \geq i$. As a consequence, $e_i \neq e_j$. It follows from the partial order $\leq$ that $e_j \not\leq e_i$.

3. Using $C3$ we get

$$\begin{aligned}
\widehat{E}_k \setminus \mathsf{PE}_k &= \widehat{E}_k \setminus \Big((\mathsf{PE}_{k-1} \setminus \{e_k\}) \cup (\widehat{E}_k \setminus \widehat{E}_{k-1}) \Big) \\
&= \Big(\widehat{E}_{k-1} \cup (\widehat{E}_k \setminus \widehat{E}_{k-1}) \Big) \setminus \Big((\mathsf{PE}_{k-1} \setminus \{e_k\}) \cup (\widehat{E}_k \setminus \widehat{E}_{k-1}) \Big) \\
&= \widehat{E}_{k-1} \setminus (\mathsf{PE}_{k-1} \setminus \{e_k\}) \\
&= (\widehat{E}_{k-1} \setminus \mathsf{PE}_{k-1}) \cup \{e_k\}
\end{aligned}$$

   because $e_k \in \mathsf{PE}_{k-1}$. Therefore, except direct successors of e_k, an event $f \in E$ satisfies the left-hand side of the unfolding invariant $I3$ w.r.t. $(\widehat{E}_k, \mathsf{PE}_k)$ iff f satisfies it w.r.t. $(\widehat{E}_{k-1}, \mathsf{PE}_{k-1})$. Hence, the set $(\widehat{E}_k \setminus \widehat{E}_{k-1})$ of added events when calling $\mathsf{Extend}(\widehat{\mathcal{E}}_{k-1}, \mathsf{PE}_{k-1}, e_k)$ contains only direct successors of e_k, i.e. $\widehat{E}_k \setminus \widehat{E}_{k-1} \subseteq \prec(e_k)$. By combining once again with the unfolding invariant $I3$, one obtains the third item of this Proposition.

□

As seen in the proof of Proposition 5.1.4, for any run of the Algorithm 5.1, events in its extending sequence $e_1, e_2, \ldots$ are pairwise different. In other words, the correctness of Extend, and more precisely, the property $C3$ stated in Definition 5.1.3 ensures that no event is extended twice. Moreover, one can see that some implementation of the unfolding algorithm is exhaustive. The operations on variable PE may be implemented so that every inserted element is eventually out, for example using a queue. In such a case, events are taken (line 4) in a same order as they are inserted, and every event will be eventually extended.

It follows from the third property in Proposition 5.1.4 that an event $f \in E$ is added to the prefix $\widehat{\mathcal{E}}$ when extending some of its direct predecessor e_k. If f has many direct predecessors, then e_k should be the predecessor which is extended last. Earlier on, Extend could not create f when extending another direct predecessor of f than e_k. Hence, the correctness of Extend, more precisely, the unfolding invariant $I3$ in Definition 5.1.1, prohibits the unfolding algorithm to create a replication of some event.

Remark: If an event f has no successors in $\mathcal{E}$, i.e. $\prec(f) = \emptyset$, then calling $\mathsf{Extend}(\widehat{\mathcal{E}}, \mathsf{PE}, f)$ intuitively does nothing but removing f from PE. In this case, the input prefix and the output prefix of Extend is the same, i.e. $\widehat{E}' = \widehat{E}$ and $\widehat{\mathcal{E}}' = \widehat{\mathcal{E}}$. Assume that f is added when extending some of its direct predecessor e, i.e. $e \in \succ(f)$. One can write a function Extend so that it does not insert f into PE when extending e. The unfolding invariant $I3$ is still preserved, and in the algorithmic view, one takes advantage of not extending f later. However, the condition $C3$ does not hold and should be changed a little bit.

We prefer to keep the strict condition $C3$ as stated in Definition 5.1.3 because it is more general and the output PE' as well as $\widehat{E}'$ is precisely determined.

5.2 Causality processes' unfolding

As shown in the previous section, while constructing prefixes of a labeled event structure, the unfolding algorithm is correct as stated in Proposition 5.1.4 if the function Extend is (Definition 5.1.3). For example, algorithms for reachability trees and their correctness proofs are not far from the ones for k-causality processes since, in both cases, events have only one direct predecessor excepts those which are minimal w.r.t. causality.

In this section, we present function Extend for k-causality processes (Section 3.3.2) and M-causality processes (Section 3.3.3). In a sense, the function Extend is nothing but an algorithmic computation of event's successor set in a given labeled event structure $\mathcal{E}$. Each event and its direct successors in $\mathcal{E}$ form a motif that may be derived from the definition of $\mathcal{E}$ (see Section 3.3). In other words, if an labeled event structure is obtained by repeating a certain motif, one can follow the schema described below and develop an algorithm for constructing the labeled event structure.

Remark:

- Recall that the causality $\leq$ is the reflexive and transitive closure of the predecessor relation $\lessdot$. The causality relation can be computed from the predecessor one and vice versa. This is true also for their corresponding restrictions onto a given subset $\widehat{E}$ of E, i.e. $\widehat{\leq} = \leq|_{\widehat{E}}$ and $\widehat{\lessdot} = \lessdot|_{\widehat{E}}$. Hence, our algorithms do not directly work with $\widehat{\leq}$ but computes only $\widehat{\lessdot}$.

- In any causality process, the marking function $\mathcal{M}$ could be defined based on events' label, i.e. the labeling function $\mathcal{L}$, and the marking of empty configuration $\mathcal{M}(\emptyset)$ (see Section 3.3 for details). Therefore, in our algorithms, we do not show explicit instructions for the marking function $\mathcal{M}$.

In order to shorten the presentation of the algorithms, we use a function $\mathsf{Create}(\widehat{\mathcal{E}}, \mathsf{PE}, P, l)$ (*cf.* in Algorithm 5.2). Function Create takes four arguments that are: the constructed prefix $\widehat{\mathcal{E}}$, the actual possible extension PE, an event set $P \subseteq \widehat{E}$, and a label $l \in \mathsf{Codom}(\mathcal{L})$. It intuitively creates a new event e of which label is l and the set of direct predecessors is P, i.e. $\gtrdot(e) = P$. We require that the events in P are pairwise concurrent. As shown in Algorithm 5.2, the function adds new event e not only to $\widehat{E}$ (line 4) but also to the possible extension PE (line 5). The modification of the labeling function $\mathcal{L}$ is done in line 6. Then it updates the predecessor relation $\widehat{\lessdot}$ so that e is a direct successor of all events in P.

The loop at lines 8-10 is responsible for updating the conflict relation $\widehat{\#}$. In fact, due to the conflict-inheritance in (prime) event structures by Definition 3.1.1, for every predecessor $p \in P = \gtrdot(e)$, e is conflict with every event f which is conflict with p, i.e. $f \in \#(p)$. This is done in line 9.

The following which is similar to the third condition of Extend's correctness (Definition 5.1.3 on page 90) is straightforward.

Lemma 5.2.1. *Let* $(\widehat{\mathcal{E}}', \mathsf{PE}')$ *denote the return value of some calling* $\mathsf{Create}(\widehat{\mathcal{E}}, \mathsf{PE}, P, l)$. *Then, we have* $\mathsf{PE}' = \mathsf{PE} \cup (\widehat{E}' \setminus \widehat{E})$.

Algorithm 5.2: Function Create

```
1  function Create(Ê, PE, P, l)
2  begin
3      create an event e
4      Ê := Ê ∪ {e}
5      PE := PE ∪ {e}
6      ℒ̂ := ℒ̂ ∪ ⟨e, l⟩
7      ≼̂ := ≼̂ ∪ (P × {e})
8      for each p ∈ P do
9          #̂ := #̂ ∪ (#̂(p) × {e}) ∪ ({e} × #̂(p))
10     end
11     return (Ê, PE, e)
12 end function
```

It is worth noticing that the value of structure variable $\widehat{\mathcal{E}}$ returned by the function Create (line 11) may not be a label event structure. It may lack some conflicts concerning e that do not come from conflict-inheritance. They will be added in the function Extend (see Section 5.2.2). However, the function Create is fully in charge of the causality $\widehat{\leq}$ as well as the labeling function $\widehat{\mathcal{L}}$. The new event e and updated possible extension PE are returned, together with $\widehat{\mathcal{E}}$, by the function Create .

5.2.1 k-causality processes

Algorithm 5.3 represents our implementation of function Init_k for the k-causality process k-$\mathcal{CP}$ (see Definition 3.3.9 on page 35). Recall briefly that, in k-$\mathcal{CP}$, there are only two type of events: decrement events, labeled by '$-$', that have no successors; increment events, labeled by '$+$', so each of them has exactly one decrement direct successor and k increment ones. There are k minimal events, *w.r.t.* the causality, that are all increment events.

Algorithm 5.3: Function Init_k for the k-causality process k-$\mathcal{CP}$

```
1  function Init_k()
2  begin
3      Ê := ∅; ≼̂ := ∅; #̂ := ∅; ℒ̂ := ∅
4      PE := ∅
5      for i := 1 to k do
6          (Ê, PE, e+) := Create(Ê, PE, ∅, +)
7      end for
8      return (Ê, PE)
9  end function
```

One first initializes the prefix $\widehat{\mathcal{E}}$ being constructed to the one containing no event, i.e. $\widehat{E}, \widehat{\leq}, \widehat{\#}$ and $\widehat{\mathcal{L}}$ are all the empty set (line 3). The possible extension PE is empty too. Then, the loop at lines 5-7 successively creates k increment events. The third argument's

value when calling $\mathsf{Create}(\widehat{\mathcal{E}}, \mathsf{PE}, \emptyset, +)$ (line 6) is the empty set, i.e. $P = \emptyset$. So, return event e_+ is minimal w.r.t. the causality $\widehat{\mathcal{E}}$ in $\widehat{E}$. Moreover, there is no modification concerning the conflict relation $\widehat{\#}$ except the assignment in line 3. Because the loop at lines 8-10 of function Create in Algorithm 5.2 is not taken into account when P is empty. Therefore, these k new events are pairwise concurrent.

Notice that the function Init_k uses the variable e_+ as well as the variable PE just for getting return value of calling Create in line 6. The function Create inserts new events in both $\widehat{E}$ and PE. Hence, it follows from the loop's invariant $\widehat{E} = \mathsf{PE}$ in function Init_k that the returned event set $\widehat{E}$ is equal to the returned possible extension PE. In line 8, returning $(\widehat{\mathcal{E}}, \mathsf{PE})$ is the same as returning $(\widehat{\mathcal{E}}, \widehat{E})$. The following is straightforward by DefinitionDefinition 3.3.9 on page 35 of k-causality processes.

Lemma 5.2.2. *The output* $(\widehat{\mathcal{E}}, \mathsf{PE}) = \mathsf{Init}_k()$ *of Algorithm 5.3 is correct w.r.t.* k-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, *moreover* $\widehat{\mathcal{E}} = k\text{-}\mathcal{CP}|_{\mathsf{Min}_{\leq}(E)}$.

Algorithm 5.4 illustrates the function Extend_k for the k-causality process k-$\mathcal{CP}$ (see Definition 3.3.9). It only expands the prefix $\widehat{\mathcal{E}}$ from increment events, that means from e where $\mathcal{L}(e) = +$ (line 4). In a same way as the function Init_k, k increment events are created due to the loop at lines 5-7. In addition, a decrement event is also created (line 8). By calling $\mathsf{Create}(\widehat{E}, \mathsf{PE}, \{e\}, +)$, these $k+1$ new events are all successors of e, and moreover, they have only e as a direct predecessor.

Algorithm 5.4: Function Extend_k for the k-causality process k-$\mathcal{CP}$

```
1  function Extend_k(Ê, PE, e)
2  begin
3      PE := PE \ {e}
4      if L(e) = + then
5          for i := 1 to k do
6              (Ê, PE, e_+) := Create(Ê, PE, {e}, +)
7          end for
8          (Ê, PE, e_−) := Create(Ê, PE, {e}, −)
9      end if
10     return (Ê, PE)
11 end function
```

Notice that in k-causality process, decrement events have no successor. Therefore, when e is a decrement event, i.e. $\widehat{\mathcal{L}}(e) = -$, the test in line 4 fails and the function Extend_k only removes e from the possible extension PE (line 3). The updated prefix $\widehat{\mathcal{E}}$ and possible extension PE are finally returned by the function Extend_k (line 10).

Lemma 5.2.3. *For every* $k \in \mathbb{N}$ *and* $k > 0$, *the function* Extend_k *in Algorithm 5.4 is correct w.r.t. to the* k*-causality process* k-$\mathcal{CP}$.

Proof. We will prove that Extend_k satisfies all correctness properties $C1, C2$ and $C3$ stated in Definition 5.1.3. Let $(\widehat{\mathcal{E}}', \mathsf{PE}') = \mathsf{Extend}_k(\widehat{\mathcal{E}}, \mathsf{PE}, e)$ denote the return value for some input $(\widehat{\mathcal{E}}, \mathsf{PE}, e)$. We assume that $(\widehat{\mathcal{E}}, \mathsf{PE})$ is correct w.r.t. k-$\mathcal{CP}$ and $e \in \mathsf{PE}$. There are two cases depending on the label of e.

- e is a decrement event: Only the instructions in line 3 and 10 in Algorithm 5.4 are brought into effect because the test in line 4 fails. Hence, we have thus $\mathsf{PE}' = \mathsf{PE} \setminus \{e\}$ while the prefix $\widehat{\mathcal{E}}$ is unchanged, i.e. $\widehat{\mathcal{E}}' = \widehat{\mathcal{E}}$. Conditions $C2$ and $C3$ are satisfied. Consider now the condition $C1$ that consists of $I1, I2, I3$ in Definition 5.1.1. Because $\widehat{\mathcal{E}}' = \widehat{\mathcal{E}}$, $\widehat{\mathcal{E}}'$ remains a prefix of k-$\mathcal{CP}$ ($I1$). By removing e from PE, PE' is a subset of PE and is thus a subset of $\widehat{E}' = \widehat{E}$ ($I2$). Let S (S') denote the set of events in k-$\mathcal{CP}$ whose direct predecessors are all in $(\widehat{E} \setminus \mathsf{PE})$ (and $(\widehat{E}' \setminus \mathsf{PE}')$ respectively). It is sufficient to prove the invariant $I3$ that $S' = \widehat{E}'$. It follows from $\widehat{E}' = \widehat{E}$ and $\mathsf{PE}' = \mathsf{PE} \setminus \{e\}$ that S' differs from S only on events e' whose set of direct predecessors contains e. However, by Definition 3.3.7 and Definition 3.3.9 for k-$\mathcal{CP}$, as e is labeled '$-$', it has no successor in k-$\mathcal{CP}$. Hence, such an event e' does not exist, and consequently, $S' = S$. Since $(\widehat{\mathcal{E}}, \mathsf{PE})$ is correct w.r.t. k-$\mathcal{CP}$ as assumed, we have $S = \widehat{E}$ and thus $S' = S = \widehat{E} = \widehat{E}'$. The condition $I3$ is satisfied and one can conclude that Extend_k is correct w.r.t. k-$\mathcal{CP}$ when e is labeled by '$-$'.

- otherwise, i.e. e is an increment event: the condition in line 4 in Algorithm 5.4 is satisfied. Let X_+ denote the set of events that are created by count-controlled loops of size k (lines 5-7) and e_- the event created in line 8. We have thus $\widehat{E}' = \widehat{E} \cup X_+ \cup \{e_-\}$. Lemma 5.1.2 gives us that $\mathsf{PE} \subseteq \mathsf{Max}_{\leq}(\widehat{E})$, hence, $e \in \mathsf{PE}$ implies that e has no successor in $\widehat{\mathcal{E}}$. Therefore, $k+1$ new events in $(X_+ \cup \{e_-\})$ exactly correspond to $k+1$ successors of e in k-$\mathcal{CP}$ by Definition 3.3.7 and Definition 3.3.9. As a consequence, $\widehat{\mathcal{E}}'$ is the prefix of k-$\mathcal{CP}$ where the event set is $\widehat{E}' = \widehat{E} \cup X \cup \{e_-\}$, i.e. $\widehat{\mathcal{E}}' = k\text{-}\mathcal{CP}|_{\widehat{E}'}$ (invariant $I1$ in Definition 5.1.1). Moreover, it follows from the instruction in line 3 and Lemma 5.2.1 that $\mathsf{PE}' = (\mathsf{PE} \setminus \{e\}) \cup X_+ \cup \{e_-\}$, and consequently, PE is the union of subsets of $\widehat{E}'$ so that $\mathsf{PE}' \subseteq \widehat{E}'$ (invariant $I2$). By the same manner as in the first case, let S and S' denote respectively the sets $\{e' \in E \,/\, \gtrdot(e') \subseteq (\widehat{E} \setminus \mathsf{PE})\}$ and $\{e' \in E \,/\, \gtrdot(e') \subseteq (\widehat{E}' \setminus \mathsf{PE}')\}$. We have $S = \widehat{E}$ because $(\widehat{\mathcal{E}}, \mathsf{PE})$ is correct w.r.t. k-$\mathcal{CP}$ by Definition 5.1.1. We will prove that $S' = \widehat{E}'$, and as a consequence, $(\widehat{\mathcal{E}}', \mathsf{PE}')$ also satisfies the invariant $I3$. Notice that every event in k-$\mathcal{CP}$ has at most one direct predecessor, and $X \cup \{e_-\}$ contains all direct successors of e in k-$\mathcal{CP}$. Therefore,

$$
\begin{aligned}
S' &= \left\{e' \in E \,/\, \gtrdot(e') \subseteq (\widehat{E}' \setminus \mathsf{PE}')\right\} \\
&= \left\{e' \in E \,/\, \gtrdot(e') \subseteq \left((\widehat{E} \cup X \cup \{e_-\}) \setminus (\mathsf{PE} \setminus \{e\} \cup X \cup \{e_-\})\right)\right\} \\
&= \left\{e' \in E \,/\, \gtrdot(e') \subseteq \left((\widehat{E} \setminus \mathsf{PE}) \cup \{e\}\right)\right\} \\
&= \left\{e' \in E \,/\, \gtrdot(e') \subseteq (\widehat{E} \setminus \mathsf{PE})\right\} \cup \left\{e' \in E \,/\, \gtrdot(e') = \{e\}\right\} \\
&= \widehat{E} \cup (X \cup \{e_-\}) \\
&= \widehat{E}'
\end{aligned}
$$

  $(\widehat{\mathcal{E}}', \mathsf{PE}')$ satisfies all $I1, I2, I3$ and is thus correct w.r.t. k-$\mathcal{CP}$. The condition $C1$ of Definition 5.1.3 is thus guaranteed. Moreover, both the conditions $C2$ and $C3$ are previously obtained. One can then conclude that Extend_k is correct w.r.t. k-$\mathcal{CP}$.

As we have said above, for clarity, marking functions are not taken into account in this argument. Let us just say that both $\widehat{\mathcal{M}}$ and $\widehat{\mathcal{M}}'$ can be defined as $\mathcal{M}$ in Definition 3.3.9, i.e. for all configurations C, $\widehat{\mathcal{M}}(C) = \widehat{\mathcal{M}}'(C) = |\{e \in C \,/\, \mathcal{L}(e) = +\}| - |\{e \in C \,/\, \mathcal{L}(e) = +\}|$. □

Lemma 5.2.4. *For every $k \in \mathbb{N}$, the function* Extend_k *in Algorithm 5.4 terminates and has a time complexity of* $O(k)$.

Proof. Since Algorithm 5.4 contains only a count-controlled loop of size k (lines 5-7) and the creating process in Algorithm 5.2 for each event (line 6 or line 8) has a time complexity of $O(1)$, the function Extend_k in Algorithm 5.4 has a time complexity of $O(k)$. Because k is finite, this function terminates. □

5.2.2 M-causality processes

First, recall that, for a non-empty alphabet M, the M-causality process M-$\mathcal{CP}$ (*cf.* Definition 3.3.27 on page 46) has only two type of events: sending events with labels in $!M$ and receiving events with labels in $?M$. Each sending event has one receiving direct successor and $|M|$ sending ones that have pairwise different labels. There are $|M|$ minimal events, w.r.t. the causality, in M-$\mathcal{CP}$ and they also correspond one-to-one to messages in M. All sending events (receiving events) are pairwise either in causal or in conflict.

Algorithm 5.5: Function $\mathsf{Init}_{\mathcal{M}}$ for the M-causality process M-$\mathcal{CP}$

```
1  function Init_M()
2  begin
3      Ê := ∅; ≤̂ := ∅; #̂ := ∅; L̂ := ∅
4      PE := ∅
5      for each m ∈ M do
6          (Ê, PE, e_!m) := Create(Ê, PE, ∅, !m)
7      end for
8      # := (Ê × Ê) \ I_Ê
9      return (Ê, Ê)
10 end function
```

Let us explain how Algorithm 5.5 initializes the construction of M-causality process's prefixes. By the same manner as in Algorithm 5.3, it starts with a labeled event structure without events (line 3), then successively inserts $|M|$ new events in the loop at lines 5-7. These events are all sending ones (line 6) and correspond to different messages in M. Moreover, since the argument corresponding to the predecessor set, when calling Create in line 6, is empty, added events are all minimal w.r.t. the causality $\widehat{\leq}$. The conflict relation $\widehat{\#}$ is assigned in line 8 so that events in $\widehat{E}$ are pairwise in conflict. These $|M|$ sending events correspond to minimal events w.r.t. causality in M-$\mathcal{CP}$ by Definition 3.3.27 on page 46. Therefore, we have that $\widehat{\mathcal{E}} = M\text{-}\mathcal{CP}|_{\mathsf{Min}_{\leq}(E)}$ and its event set $\widehat{E} = \mathsf{Min}_{\leq}(E)$ are finally returned by the function $\mathsf{Init}_{\mathcal{M}}$ (lines 9). Lemma 5.2.5 is thus obvious by definition.

Lemma 5.2.5. *The output* $(\widehat{\mathcal{E}}, \widehat{E}) = \mathsf{Init}_{\mathcal{M}}$ *of Algorithm 5.5 is correct w.r.t.* $M\text{-}\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, *moreover* $\widehat{\mathcal{E}} = k\text{-}\mathcal{CP}|_{\mathsf{Min}_{\leq}(E)}$.

Consider now Algorithm 5.6 of our function $\mathsf{Extend}_{\mathcal{M}}$ for the M-causality process M-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$. This function is divided into two parts depending on whether the input event e is a sending event or a receiving one. In both cases, however, one always

removes e from the possible extension PE (line 4) in order to satisfy the correctness condition $C3$ in Definition 5.1.3 on page 90.

The first case is when e is a sending event (lines 5-20). The algorithm gets the message $m \in M$ which corresponds to the label of e, i.e. $\widehat{\mathcal{L}}(e) = !m$ (line 6). Notice that successors of e do not exist in the actual prefix yet because e is a maximal event w.r.t. the causality $\widehat{\leq}$ when calling $\mathsf{Extend}_{\mathcal{M}}(\widehat{\mathcal{E}}, \mathsf{PE}, e)$. The loop at lines 8-11 creates then new sending events corresponding to direct successors of e by calling Create in line 9. There are exactly $|M|$ events created, one for each message in M due to the "for each" criterion (line 8). The variable X declared in line 2 is initialized (line 7) and updated inside the loop (line 10) so that X is the set of new sending successors of e. There are two kinds of conflict concerning these sending successors: the first one comes from e due to inheritance and is done in side the function Create (line 9) and the second one is the conflict between these successors themselves. The set X is used just for modifying conflict in order to respect to the second kind (line 12).

Algorithm 5.6: Function $\mathsf{Extend}_{\mathcal{M}}$ for the M-causality process M-$\mathcal{CP}$

```
1  function Extend_M(Ê, PE, e)
2  var X
3  begin
4     PE := PE \ {e}
5     if L̂(e) ∈ !M then
6        let m ∈ M s.t. L̂(e) = !m
7        X := ∅
8        for each m' ∈ M do
9           (Ê, PE, e_!m') := Create(Ê, PE, {e}, !m')
10          X := X ∪ {e_!m'}
11       end for
12       #̂ := #̂ ∪ ((X × X) \ I_X)
13       if ≽̂(e) = ∅ then
14          (Ê, PE, e_?m) := Create(Ê, PE, {e}, ?m)
15       else
16          take an event e_!m' ∈ ≽̂(e)
17          if exists e_?m' ∈ (≼̂(e_!m') \ PE) s.t. L̂(e_?m') ∈ ?M then
18             (Ê, PE, e_?m) := Create(Ê, PE, {e, e_?m'}, ?m)
19          end if
20       end if
21    else
22       let e_!m ∈ ≽̂(e) s.t. L(e_!m) ∈ !M
23       for each e_!m' ∈ (≼̂(e_!m) \ PE) s.t. L̂(e_!m') ∈ !M do
24          let m' ∈ M s.t. L̂(e_!m') = !m'
25          (Ê, PE, e_?m') := Create(Ê, PE, {e, e_!m'}, ?m')
26       end for
27    end if
28    return (Ê, PE)
29 end function
```

Now, let us look how to determine whether the unique receiving successor of e denoted by $e_{?m}$ must be added to the prefix. If e is a minimal event w.r.t. the causality, formally $\widehat{\gtrdot}(e) = \emptyset$, (lines 13-14) $e_{?m}$ is simply created and inserted into the actual prefix because it has only one predecessor which is e. Otherwise, i.e. when e is not minimal, (lines 15-20) e must be a successor of some sending event $e_{!m'}$. The event $e_{!m'}$ obtained at line 16 is unique, and moreover it has a unique receiving successor denoted by $e_{?m'}$ in M-$\mathcal{CP}$, i.e. $\{e_{?m'}\} = \{e' \in \lessdot(e_{!m'}) \,/\, \mathcal{L}(e') \in ?M\}$. However, $e_{?m'}$ may not be in the event set $\widehat{E}$ of the actual prefix $\widehat{\mathcal{E}}$. Therefore, $e_{?m}$ needs to be created if and only if $\widehat{E}$ contains such $e_{?m}$ and moreover $e_{?m'}$ is already extended (not in PE). It follows from $\widehat{\lessdot} = \lessdot|_{\widehat{E}}$ that one can simply write $\widehat{\lessdot}(e)$ in the place of $\lessdot(e) \cap \widehat{E}$. The test at line 17 formally represents this condition. If it is true, then $e_{?m}$ is created by calling Create (line 18) so that e and $e_{?m'}$ are its only two direct predecessors. Notice that conflict concerning $e_{?m}$ comes only from e, and it is done in side the function Create.

The second case is when e is a receiving event (lines 21-27). We know that e exactly one direct predecessor which is a sending event. The event $e_{!m}$ obtained at line 22 is thus unique. Moreover, $e_{!m}$ corresponds to the send of the message that e is a receive, i.e. $\mathcal{L}(e) = ?m$ and $\mathcal{L}(e_{!m}) = !m$. By definition of M-$\mathcal{CP}$, the event $e_{!m}$ has $|M|$ direct sending successors represented by the set $\{e_{!m'} \in \lessdot(e_{!m}) \,/\, \mathcal{L}(e_{!m'}) \in !M\}$. These successors are pairwise in conflict. Moreover, they are already added to the actual prefix $\widehat{\mathcal{E}}$ because $e_{!m}$ has been expanded before e. We formally write $\{e_{!m'} \in \lessdot(e_{!m}) \,/\, \mathcal{L}(e_{!m'}) \in !M\} = \{e_{!m'} \in \widehat{\lessdot}(e_{!m}) \,/\, \mathcal{L}(e_{!m'}) \in !M\}$ and denote it by $S_!$.

The event e on its turn has exactly M direct successors which are all receiving ones and are represented by the set $\lessdot(e)$. For a given message $m' \in M$ (which may be m), the corresponding receiving event $e_{?m'} \in \lessdot(e)$ has two direct predecessors which are e and $e_{!m'} \in S_!$. Therefore, $e_{?m'}$ could be added in the actual prefix by $\mathsf{Extend}_{\mathcal{M}}$ iff $e_{!m'}$ has been extended, i.e. $e_{!m'} \notin \mathsf{PE}$. This condition is represented in the loop condition in line 23. Each receiving successor $e_{?m'}$ created by the loop at lines 23-26, is at the same time is a direct successor of e and a direct successor of the corresponding $e_{!m'}$ (line 25). Once again, the conflict relation concerning $e_{?m}$, is inherited from either e or $e_{!m}$. And this is done inside the function Create.

As usual, in both cases, the new prefix $\widehat{\mathcal{E}}$ and the possible extension PE are returned in the end of the function $\mathsf{Extend}_{\mathcal{M}}$ (line 28).

Lemma 5.2.6. *For every given alphabet M which is not empty, the function $\mathsf{Extend}_{\mathcal{M}}$ in Algorithm 5.6 is correct w.r.t. to the M-causality process M-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$.*

Proof. Let $(\widehat{\mathcal{E}}', \mathsf{PE}') = \mathsf{Extend}_k(\widehat{\mathcal{E}}, \mathsf{PE}, e)$ denote the return value for some given input $(\widehat{\mathcal{E}}, \mathsf{PE}, e)$ which satisfies that $(\widehat{\mathcal{E}}, \mathsf{PE})$ is correct w.r.t. k-$\mathcal{CP}$ and $e \in \mathsf{PE}$. Let us define $E_! = \{e' \in E \,/\, \mathcal{L}(e') \in !M\}$ and $E_? = \{e' \in E \,/\, \mathcal{L}(e') \in ?M\}$. It follows from the properties of the M-causality process (Definition 3.3.25 on page 44 and Definition 3.3.27 on page 46) that

(1) let $\lessdot_k = (\lessdot \setminus (E_? \times E_?))$, then $\lessdot_k, E_!$ and $E_?$ respectively correspond to the predecessor relation, the set of increment events, and the set of decrement events in $|M|$-causality process (see Definition 3.3.7 on page 34 and Definition 3.3.9 on page 35), and $\mathsf{Min}_{\leq}(E) \subset E_!$;

(2) let $\mathcal{B}_? = \gtrdot \cap (E_? \times E_!)$, then for all $e_?, f_? \in E_?$, $e_? \lessdot f_?$ iff $\mathcal{B}_?(e_?) \lessdot \mathcal{B}_?(f_?)$;

(3) let us denote by $\#_m = \{\langle f, g\rangle \in \# \,/\, {>}(f)\, \widehat{\#^s}\, {>}(g)\}$ (and $\#_{m!} = \{\langle f_!, g_!\rangle \in (E_! \times E_!) \,/\, f_! \neq g_!$ and ${>}(f_!) = {>}(g_!)\}$) the relation of conflict between events in E (and

between sending events in $E_!$ respectively) which does not comes from inheritance, then $\#_m = \#_{m!}$;

(4) for all $e_? \in E_?$, $\Pi_{?M}(\mathcal{L}(e_?)) = \Pi_{!M}(\mathcal{L}(\mathcal{B}_?(e_?)))$; and

(5) for all $e_! \in E_!$, $\mathcal{L}|_{S_!}$ is a bijection between $S_!$ and $!M$ where $S_! = \lessdot(e_!) \cap E_!$.

Since $\widehat{\mathcal{E}}$ is a prefix of M-$\mathcal{CP}$, by definition, we have $\widehat{E}$ is a subset of E and moreover, $\widehat{\lessdot}, \widehat{\#}, \widehat{\mathcal{L}}$ are respectively the restriction of $\lessdot, \#, \mathcal{L}$ onto $\widehat{E}$. We are going to show that $\widehat{\mathcal{E}}'$ is also a prefix of M-$\mathcal{CP}$. There are two cases depending on whether the argument event e of calling $\mathsf{Extend}_{\mathcal{M}}$ is a sending event or is a receiving one.

- When e is a sending event, i.e. $\widehat{\mathcal{L}}(e) = !m$ for some message $m \in M$, let us denote $e = e_{!m}$. Notice that $e_{!m}$ is maximal in $\widehat{E}$ w.r.t. $\widehat{\leq}$ because $(\widehat{\mathcal{E}}, \mathsf{PE})$ is correct w.r.t. M-$\mathcal{CP}$ (Lemma 5.1.2). Event $e_{!m}$ has $|M|$ sending successors in $\widehat{E}'_!$ due to the loop at lines 8-11. Let us denote the set of these sending successors by $X_!$. Moreover, $e_{!m}$ has at most one receiving successor in $\widehat{E}'_?$ depending on whether the conditions at line 13 and line 17 are satisfied or not. This receiving successor, if exists in $\widehat{\mathcal{E}}'$, is the event $e_{?m}$ obtained either in line 14 or in line 18. Moreover, it follows from (1) that, in $\mathcal{E}$, we have $|\lessdot(e_{!m}) \cap E_!| = |M|$ and $|\lessdot e_{!m} \cap E_?| = 1$. Hence, without lost of generality, we suppose that $X_! = \lessdot(e_{!m}) \cap E_!$ and $\{e_{?m}\} = \lessdot(e_{!m}) \cap E_?$. Then, we will prove that $\widehat{\lessdot}', \widehat{\#}', \widehat{\mathcal{L}}'$ are respectively the restrictions of $\lessdot, \#, \mathcal{L}$ onto $\widehat{E}'$.

  If $e_{!m}$ is minimal w.r.t. $(\widehat{E}, \widehat{\leq})$, then a receiving successor $e_{?m}$ of e is created in line 14. One skips all the rest of the algorithm and directly returns final structure $\widehat{\mathcal{E}}'$ and PE' (line 28). Briefly, due to the function Create in Algorithm 5.2, we obtain that:
  - $\widehat{E}' = \widehat{E} \cup X_! \cup \{e_{?m}\}$,
  - $\widehat{\lessdot}' = \widehat{\lessdot} \cup (\{e_{!m}\} \times (X_! \cup \{e_{?m}\}))$,

  Let X shortly denote the set of direct successors of $e_{!m}$ in $\mathcal{E}$, i.e. $X = \lessdot(e_{!m}) = X_! \cup \{e_{?m}\} = \widehat{E}' \setminus \widehat{E}$. We have,

$$\begin{aligned}\lessdot|_{\widehat{E}'} &= \lessdot|_{\widehat{E}} \cup \Big(\lessdot \cap (\widehat{E} \times X)\Big) \cup \Big(\lessdot \cap (X \times \widehat{E}')\Big) \\ &= \widehat{\lessdot} \cup \Big(\lessdot \cap (\widehat{E} \times X)\Big) \cup \Big((\lessdot \cap (X \times \widehat{E})) \cup (\lessdot \cap (X \times X))\Big)\end{aligned}$$

  It follows from (1) that successors of sending event $e_{!m}$ are pairwise not in causal. Hence $\lessdot \cap (X \times X) = \emptyset$. Thanks to Lemma 5.1.2, $e_{!m} = e$ is maximal event in $\widehat{E}$ w.r.t. $\widehat{\leq} = \leq|_{\widehat{E}}$. Successors of $e_{!m}$ may not have predecessors in $\widehat{E}$, and as a consequence, $\lessdot \cap (X \times \widehat{E}) = \emptyset$. Therefore,

$$\lessdot|_{\widehat{E}'} = \widehat{\lessdot} \cup \Big(\lessdot \cap (\widehat{E} \times X)\Big)$$

  Once again, due to (1), every sending event in $X_! \subset X$ has one direct predecessor which is $e_{!m}$. And since $e_{!m}$ is minimal event w.r.t. $\leq$, it has no direct predecessor. It follows from (2) that $e_{?m}$ has no receiving predecessor in $E_?$. In addition to (1), $e_{?m}$ also has only one direct predecessor in $E_!$ which is $e_{!m}$. Hence, for all $e' \in X$, $\gtrdot(e') = \{e_{!m}\}$. Therefore, $\lessdot \cap (\widehat{E} \times X) = \{e_{!m}\} \times X$, and consequently, $\lessdot|_{\widehat{E}'} = \widehat{\lessdot}'$. In words, $\widehat{\lessdot}'$ is the restriction of $\lessdot$ onto $\widehat{E}'$.

Notice that when calling Create at lines 9 and 14, instructions for conflict relation inside the function Create concern only conflict that comes from inheritance. We have then $\widehat{\#}'_m$ (defined by a same manner as $\#_m$ in (3)) is the set $\widehat{\#}_m \cup (X_! \times X_!) \setminus \mathcal{I}_{X_!}$. When proving $\widehat{\#}' = \#|_{\widehat{E}'}$, it is sufficient to show that $\widehat{\#}'_m = \#_m|_{\widehat{E}'}$. Thanks to (3), we have $\#_m = \#_{m!}$ and as a consequence $\widehat{\#}_m = \#_m|_{\widehat{E}} = \#_{m!}|_{\widehat{E}} = \widehat{\#}_{m!}$. We have:

$$\begin{aligned}
\#_m|_{\widehat{E}'} &= \#_{m!}|_{\widehat{E}'} \\
&= \#_{m!}|_{\widehat{E}} \cup \Big(\#_{m!} \cap (\widehat{E}' \times X)\Big) \cup \Big(\#_{m!} \cap (X \times \widehat{E}')\Big) \\
&= \widehat{\#}_m \cup \Big(\#_{m!} \cap (\widehat{E}'_! \times X_!)\Big) \cup \Big(\#_{m!} \cap (X_! \times \widehat{E}'_!)\Big) \\
&= \widehat{\#}_m \cup \Big(\#_{m!} \cap (\widehat{E}_! \times X_!)\Big) \cup \Big(\#_{m!} \cap (X_! \times \widehat{E}_!)\Big) \\
&\qquad \cup (\#_{m!} \cap (X_! \times X_!))
\end{aligned}$$

By definition, for all events $f, g \in E_!$, $f \#_{m!} g$ iff $>(f) = >(g)$ and $f \neq g$. Moreover, since $e_{!m} = e$ is maximal in $\widehat{E}$ w.r.t. $\leq$, hence for all $f \in \widehat{E}'_!$, $e_{!m} \in >(f)$ if $f \in X_!$ and $e_{!m} \notin >(f)$ if $x \in \widehat{E}_!$. We obtain that $\#_{m!} \cap (E' \times X_!) = (E' \times X_!) \cap \#_{m!} = \emptyset$. Therefore,

$$\begin{aligned}
\#_m|_{\widehat{E}'} &= \widehat{\#}_m \cup (\#_{m!} \cap (X_! \times X_!)) \\
&= \widehat{\#}_m \cup ((X_! \times X_!) \setminus \mathcal{I}_{X_!}) \\
&= \widehat{\#}'_m
\end{aligned}$$

The conflict relation $\widehat{\#}'$ is thus the restriction of $\#$ onto $\widehat{E}'$.

The event $e_{!m}$ and its direct receiving successor $e_{?m}$ are related to a same message m obtained in line 6. It respects to (4). Due to the "for each" criterion of the loop at lines 8-11, property (5) is guaranteed for sending event $e_{!m}$. One can verify that $\widehat{\mathcal{L}}' \setminus \widehat{\mathcal{L}} = \mathcal{L}|_{(X_! \cup \{e_{?m}\})}$, and as a consequence, the labeling function $\widehat{\mathcal{L}}'$ is also the restriction of $\mathcal{L}$ onto $\widehat{E}'$. Therefore, $\widehat{\mathcal{E}}$ is a prefix of M-$\mathcal{CP}$.

Now, if $e = e_{!m}$ is not minimal in $\widehat{E}$ w.r.t. $\widehat{\leq}$. It follows from (1) that event $e_{!m'}$ (line 16) exists and is the unique event in $\gtrdot(e_{!m})$. First, if the condition in line 17 is true, one creates event $e_{?m}$ due to line 18. Like previous case, we have

- $\widehat{E}' = \widehat{E} \cup X_! \cup \{e_{?m}\}$, but
- $\widehat{\lessdot}' = \widehat{\lessdot} \cup (\{e_{!m}\} \times (X_! \cup \{e_{?m}\}) \cup \{\langle e_{?m'}, e_{?m} \rangle\}$,

where $e_{?m'}$ is the receiving successors of $e_{!m'}$. And the restriction of $\lessdot$ onto $\widehat{E}'$ is still equal to $\lessdot|_{\widehat{E}} \cup (\lessdot \cap (\widehat{E} \times X))$. It follows from (2) that $e_{?m'} \lessdot e_{?m}$. Because $\mathcal{B}_?(e_{!m'}) = e_{!m'}$, $\mathcal{B}_?(e_{!m}) = e_{!m}$, and $e_{!m'}$ is the direct predecessor of $e_{!m}$, i.e. $e_{!m'} \lessdot e_{!m}$. By the same reasoning as above, one obtain that $\widehat{\lessdot}' = \lessdot|_{\widehat{E}'}$. The proof which show that $\widehat{\#}' = \#|_{\widehat{E}'}$ as well as $\widehat{\mathcal{L}}' = \widehat{\mathcal{L}}|_{\widehat{E}'}$ is also the same. Therefore, $\widehat{\mathcal{E}}'$ is thus a prefix of M-$\mathcal{CP}$. Second, if the test in line 17 fails, no receiving successor of $e_{!m}$ is created. The final $\widehat{\mathcal{E}}'$ is intuitively the prefix of the one in previous case (where condition in line 17 is true). Hence, $\widehat{\mathcal{E}}'$ is also a prefix of M-$\mathcal{CP}$.

- When e is a receiving event, i.e. $\widehat{\mathcal{L}}(e) = ?m$ for some message $m \in M$, let us denote $e = e_{?m}$. It follows from (1) that $e_{?m} \notin E_!$ is not minimal in E. Moreover, it has one and only one sending predecessor. The event $e_{!m}$ obtained in line 22 is

unique and corresponds to the same message m as its subscript means. Once again, due to (1), beside $e_{?m}$, $e_{!m}$ has $|M|$ direct successors which are sending events. Let us denote $X_! = \lessdot(e_{!m})$. By the invariant $I3$ in Definition 5.1.1 on page 89, $e_{?m} \in \widehat{E}$ implies that its predecessor $e_{!m}$ may not be in PE. And every event $e_! \in X_!$, on its turn, we have $e_! \in \widehat{E}$ because $\gtrdot(e_!) = \{e_{!m}\} \subseteq \mathsf{PE}$. Therefore, $X_! \subseteq \widehat{E}$.

We first suppose that $X_! \cap \mathsf{PE} = \emptyset$. The loop at lines 23-26 then creates a receiving event $e_{?m'}$ according to each event $e_{!m'} \in X_!$. Each pair $e_{?m'}$ and $e_{!m'}$ are related by its common message $m' \in M$ (line 24). Thanks to Lemma 5.1.2, $e_{?m}$ is maximal in $\widehat{E}$ w.r.t. $\leq$. So that such a receiving event $e_{?m'}$ may not be in $\widehat{E}$. It follows from the call Create in line 25 that $e_{?m'}$ has two direct predecessors that are $e_{!m'} \in E_!$ and $e_{?m} \in E_?$. As a consequence, $e_{?m'}$ is the unique sending direct successor of $e_{!m'}$. By definition of M-causality process, we have $\mathcal{B}_?(e_{?m'}) = e_{!m'}$ and $\mathcal{B}_!(e_{!m'}) = e_{?m'}$. We will prove that $\widehat{\lessdot}', \widehat{\#}', \widehat{\mathcal{L}}'$ are then restrictions of $\lessdot, \leq, \mathcal{L}$ onto $\widehat{E}'$ respectively. Let $X_? = \mathcal{B}_?(X_!)$ denote the set of new receiving events. It obviously follows from the loop at lines 23-26 that:

- $\widehat{E}' = \widehat{E} \cup X_?$,
- $\widehat{\lessdot}' = \widehat{\lessdot} \cup (\{e_{?m}\} \times X_?) \cup \{\langle \mathcal{B}_?(e_?), e_? \rangle \,/\, e_? \in X_?\}$.

By the same reasoning as in previous case, we obtain:

$$\lessdot|_{\widehat{E}'} = \widehat{\lessdot} \cup (\lessdot \cap (\widehat{E} \times X_?))$$

Let $e_?$ be any event in $X_?$. Due to (1), $e_?$ has a direct predecessor which is a sending event. By definition of $\mathcal{B}_?$, this predecessor is $e_! = \mathcal{B}_?(e_?) \in X_!$. Moreover, $e_!$ has only one predecessor which is the sending event $e_{!m}$ because $e_! \in X_!$. Hence, $\gtrdot(e_!) = \{e_{!m}\}$, and as a consequence of (2), we have $\gtrdot(e_?) \cap E_? = \mathcal{B}_?(\gtrdot(e_!)) = \mathcal{B}_?(e_{!m}) = \{e_{?m}\}$. Therefore, for all $e_? \in X_?$, $\gtrdot(e_?) = (\gtrdot(e_?) \cap E_!) \cup (\gtrdot(e_?) \cap E_?) = \{\mathcal{B}_?(e_?)\} \cup \{e_{?m}\}$. We thus conclude that $\lessdot|_{\widehat{E}'} = \widehat{\lessdot}'$. Since $\widehat{\#}'_m$ is the same as $\#_m$, it is also the restriction of $\#$ onto $\widehat{E}'$. As a consequence, $\widehat{\#}' = \#|_{\widehat{E}'}$. The labeling function $\widehat{\mathcal{L}}'$, on its turns, is $\mathcal{L}|_{\widehat{E}'}$ by simply comparing $\widehat{\mathcal{L}}' \setminus \widehat{\mathcal{L}} = \widehat{\mathcal{L}}'|_{X_?}$ with $\mathcal{L}|_{X_?}$. Therefore, $\widehat{\mathcal{E}}'$ is a prefix of M-$\mathcal{CP}$.

Now, if $X_! \cap \mathsf{PE} \neq \emptyset$, denote then $X'_! = X_! \setminus \mathsf{PE}$. Events in $X'_!$ satisfy the loop condition in line 23 while events in $(X_! \setminus X'_!)$ do not. The final $\widehat{\mathcal{E}}'$ is intuitively a prefix of the one previously obtained where $X'_! = X_!$ (i.e. $X_! \cap \mathsf{PE} = \emptyset$). More precisely, these two prefixes differ, the one from the other, only on events in $\mathcal{B}_?(X_! \setminus X'_!)$. Therefore, $\widehat{\mathcal{E}}'$ is also a prefix of M-$\mathcal{CP}$ in this case.

$(\widehat{\mathcal{E}}', \mathsf{PE}')$ satisfies the conditions $C2$ in Definition 5.1.3 as well as the invariants $I1, I2$ in Definition 5.1.1 on page 89 because $\widehat{\mathcal{E}}'$ is always a prefix of M-$\mathcal{CP}$ as previously proved. The condition $C3$ that says $\mathsf{PE}' = (\mathsf{PE} \setminus \{e\}) \cup (\widehat{E}' \setminus \widehat{E})$, is thus direct from Lemma 5.2.1 and the assignment in line 4. For proving the correctness of the function $\mathsf{Extend}_{\mathcal{M}}$, it is finally sufficient to show that the invariant $I3$ is preserved in $(\widehat{\mathcal{E}}', \mathsf{PE}')$.

Notice that only e is taken from PE (line 4), so $\widehat{E}' \setminus \mathsf{PE}' = (\widehat{E} \setminus \mathsf{PE}) \cup \{e\}$. We need to reestablish the invariant $I3$ by adding some successors e' of e that $\gtrdot(e') \subseteq (\widehat{E} \setminus \mathsf{PE}) \cup \{e\}$ (*). We have two cases depending on whether e is a sending or a receiving event.

- If $\mathcal{L}(e) = !m$ for some message $m \in M$ (lines 5-20) then e has $|M|$ sending and direct successors which are represented by the set $X_!$, i.e. $X_! = \lessdot(e) \cap E_!$ as previously

discussed. Moreover, for all $e' \in X_!$, e' has only one direct predecessor which is e, and as a consequence, e' satisfies (*). Then we should add these successors to $\widehat{E}$ and PE in order to reestablish the invariant $I3$. This is done by the loop at lines 8-11. Apart from these send events, e has a successor $e_{?m}$ that is a receive of a message m, $\mathcal{L}(e_{?m}) = ?m$. And, more precisely $e_{?m} = {\lessdot}(e) \cap E_?$. If e is a minimal event in $\widehat{\mathcal{E}}$ and thus in M-$\mathcal{CP}$, i.e. $e \in \mathsf{Min}_{\widehat{\leq}}(\widehat{E}) \subseteq \mathsf{Min}_{\leq}(E)$, then it follows from (1) and (2) that e is the only predecessor of $e_{?m}$. Hence $e_{?m}$ satisfies (*) and should be added to $\widehat{E}$ and PE in order to guarantee $I3$. This is done in lines 14. If e is not a minimal event then it has a predecessor $e_{!m'}$ that is a sending event. Event $e_{!m'}$ has a successor $e_{?m'}$ that is a receiving event. In M-$\mathcal{CP}$, $e_{?m'}$ is a predecessor of $e_{?m}$ due to (2) because $\mathcal{B}_?(e_{?m'}) = e_{!m'} {\lessdot} e = \mathcal{B}_?(e_{?m})$. The invariant $I3$ requires that we add $e_{?m}$ to $\widehat{E}$ and PE iff $e_{?m'}$ is already in $\widehat{E} \setminus \mathsf{PE}$. This is done in line 18. In brief, the invariant $I3$ is preserved for all successors of e including both in ${\lessdot}(e) \cap E_!$ and in ${\lessdot}(e) \cap E_?$.

- If $\mathcal{L}(e) = ?m$ for some message $m \in M$, then e has a direct predecessor $e_{!m} \in \widehat{E}$ where $\mathcal{L}(e_{!m}) = !m$. Because $e_{!m}$ is not maximal in $\widehat{E}$ w.r.t. $\widehat{\leq}$, PE does not contains $e_{!m}$. Due to the invariant $I3$, every direct successor of $e_{!m}$ which is a sending event, and consequently, has only $e_{!m}$ as direct predecessor, must be already in $\widehat{E}$. Or one can say $X_! = {\lessdot}(e_{!m}) \cap E_!$ is a subset of $\widehat{E}$. Therefore, in M-$\mathcal{CP}$, direct successors of e are all sending events and correspond one-to-one to $X_!$ w.r.t. the bijection $\mathcal{B}_?$. Formally, $X_? = {\lessdot}(e) \subset E_?$ and $X_? = \mathcal{B}_?(X_!)$. Each event $e_?$ in $X_?$ has two direct predecessors which are e and the corresponding event $e_!$ in $X_!$, i.e. $\mathcal{B}_?(e_!) = e_?$. So $e_?$ satisfies (*) if $e_!$ is not in PE. In such a case, the invariant $I3$ requires that $e_?$ must be added to $\widehat{E}$. The loops at lines 23-26 creates and adds event $e_?$ in the set $\mathcal{B}_?(X_! \setminus \mathsf{PE})$. Therefore, the invariant $I3$ is preserved in $(\widehat{\mathcal{E}}', \mathsf{PE}')$ for all direct successors of e.

We can finally conclude that the function $\mathsf{Extend}_{\mathcal{M}}$ in Algorithm 5.6 is correct w.r.t. M-causality process M-$\mathcal{CP}$. □

As seen in the proof of Lemma 5.2.6, a difficult point to show is the correctness of adding receiving events. In general, each receiving event $e_{?m}$ has two predecessors: the first one is its corresponding event $e_{!m}$ and the second one is another receiving event $e_{?m'}$. The predecessor relation between $e_{?m'}$ and $e_{?m}$ intuitively comes from the FIFO ordering of messages. Here, $e_{?m'}$ and $e_{?m}$ respectively correspond to some two messages m' and m. And the message m' is inserted into the FIFO channel just before the message m. The event $e_{?m}$ is created when calling function $\mathsf{Extend}_{\mathcal{M}}$ for either $e_{!m}$ or $e_{?m'}$. However, since $e_{!m}$ and $e_{?m'}$ are concurrent, one does not know which event is extended the first. Thanks to the test in line 17 and the loop criterion in line 23 of Algorithm 5.6, there is no double copy of $e_{?m}$ in the prefixes of M-$\mathcal{CP}$ generated by Algorithm 5.1. This no-redundancy property is formally stated as the second property of Proposition 5.1.4. One will see lately in Section 5.3 that the order of expanding $e_{!m}$ and $e_{?m'}$ is generally determined by the process of unfolding a whole synchronized product of labeled event structures in which M-$\mathcal{CP}$ is simply one of its components.

Notice that conflict in M-$\mathcal{CP}$ comes from sending events. And like the marking function $\mathcal{M}$, the conflict relation $\#$ in M-$\mathcal{CP}$ may be computed based on the causality $\leq$

and the labeling function $\mathcal{L}$. Formally, as a consequence of Proposition 3.3.26, we have:

$$\begin{aligned}\# &= (E \times E) \setminus (\leq \cup \geq) \setminus \| \\ &= (E \times E) \setminus (\leq \cup \geq) \setminus \\ &\quad \{\langle e_?, f_! \rangle, \langle f_!, e_? \rangle \,/\, \mathcal{L}(e_?) \in ?M, \mathcal{L}(f_!) \in !M \text{ and } \mathcal{B}_?(e_?) < f_!\}\end{aligned}$$

The bijection $\mathcal{B}_?$ is defined in Definition 3.3.25 on page 44, and may be calculated from the causality $\leq$ and the labeling function $\mathcal{L}$. Therefore, in Algorithm 5.6, there is no need to verify conflict for adding new events, for labeling them as well as for updating causality. Instructions concerning the conflict relation $\widehat{\#}$, for instance in line 12, are just for computing $\widehat{\#}$ itself. Therefore, one can write a function Extend for M-causality processes that computes only the causality and the labeling function in addition to creating events. The conflict relation may be computed at need.

Lemma 5.2.7. *If one removes the instructions computing the conflict relation in Algorithm 5.2 (lines 8-10), then the algorithm of function* $\mathsf{Extend}_{\mathcal{M}}$ *for* M*-*$\mathcal{CP}$ *in Algorithm 5.6 terminates and has a time complexity of* $O(|M|)$.

Proof. In Algorithm 5.6, there are only loops whose bound does not exceed the number or messages in M. Moreover, when creating a new event by calling Create, one has only to label it and assign the predecessor relation according to at most two other events. Therefore, the time complexity of the algorithm is thus $O(|M|)$. Since the alphabet M contains finite messages, that means $|M|$ is finite, Algorithm 5.6 terminates. □

5.2.3 Generalization

Recall that, causality processes defined in Section 3.3 are similar, the one to the other. In this section, we only give intuitive ideas of how to modify previous algorithms of functions $\mathsf{Init}_{\mathcal{M}}$ and $\mathsf{Extend}_{\mathcal{M}}$ in order to have algorithms that are suitable for any given M-causality process corresponding to a FIFO channel. In a sense, our modifications are just adaptations of $\mathsf{Init}_{\mathcal{M}}$ as well as of $\mathsf{Extend}_{\mathcal{M}}$ to the fact whether the FIFO channel initially has some messages, or whether it is bounded. Even though we explain algorithms for FIFO channels, modifications concerning algorithms for counters are also mentioned at necessary points.

(M, v)-causality processes

By Definition 3.3.31 on page 49, for a given alphabet M and a word $v \in M$, the (M, v) causality process, denoted by (M, v)-$\mathcal{CP}$, intuitively consists of the (M, v)-flushing process (Definition 3.3.30 on page 48) and the M-causality process. Therefore, this fact conducts to a function $\mathsf{Init}_{\mathcal{M}v}$ differs from the function $\mathsf{Init}_{\mathcal{M}}$ in Algorithm 5.5 on page 96, only on whether there exists a receiving $e_?$ which is a minimal event in (M, v)-$\mathcal{CP}$. In other words, $e_?$ exists if the FIFO channel contains some messages at the beginning, i.e. $|v| > 0$. In this case, $e_?$ should be labeled according to the first message m in the word v, i.e. $\mathcal{L}(e_?) = ?m$. We can simply modify Algorithm 5.5 by adding instructions concerning $e_?$ just after line 8 as follows:

For the function $\mathsf{Extend}_{\mathcal{M}v}$, we are in need of two additional functions: Dep and NotConflict. First, the function $\mathsf{Dep}(e)$ takes an event e as argument and is exactly the depth function in Definition 3.3.34 on page 52. As explained in Section 3.3.3, (M, v)-$\mathcal{CP}$ represents behaviors of the FIFO channel initialized by the word $v \in M^*$. And for any configuration C of (M, v)-$\mathcal{CP}$, the whole content of the FIFO channel without removing

```
if |v| > 0 then
    let m be the first message in v
    (Ê, PE, e?) := Create(Ê, PE, ∅, ?m)
end if
```

messages is some word $w = \mathcal{M}(C)$ over M. Moreover, every event e in C corresponds to one message in w, and as a consequence, corresponds to the index of this message in w. Such an index is the return value of $\mathsf{Dep}(e)$.

By Definition 3.3.31 on page 49, (M,v)-$\mathcal{CP}$ consists of a (M,v)-flushing process and a M-causality process. Let us denote by E^f the set of $|v|$ events of (M,v)-$\mathcal{CP}$ which correspond to the flushing process. One can see that E^f contains only receiving events, i.e. $E^f \subset E_?$, and the depth values of these events are pairwise different and are all in the range of $\{1,2,\ldots,|v|\}$. Moreover, for all event $e \in (E \setminus E_f)$, we have $\mathsf{Dep}(e) > |v|$.

Second, the function $\mathsf{NotConflict}(e,d)$ takes an event e and a depth d as input. It intuitively returns the set of events e' whose depth is d and e' is not in conflict with e. And there is a constraint that $\mathsf{NotConflict}(e,d)$ returns only receiving messages if e is a sending one, and vice versa. Formally,

$$\mathsf{NotConflict}(e,d) = \begin{cases} \{e_? \in E_? \,/\, \mathsf{Dep}(e_?) = d \text{ and } e_? \,\overline{\#}\, e\} \text{ if } e \in E_! \\ \{e_! \in E_! \,/\, \mathsf{Dep}(e_!) = d \text{ and } e_! \,\overline{\#}\, e\} \text{ otherwise.} \end{cases}$$

As stated in Lemma 3.3.35, the bijection $\mathcal{B}_! \in (E_! \times (E_? \setminus E^f))$ as well as $\mathcal{B}_? = \mathcal{B}_!^{-1}$ between sending and receiving events in (M,v)-$\mathcal{CP}$ can be determined by function Dep, the labeling function $\mathcal{L}$ and the conflict relation $\#$. However, for a general use of function $\mathsf{NotConflict}$, $\mathsf{NotConflict}$ may not return a singleton due to the variation of argument d. Let us consider some examples of calling $\mathsf{NotConflict}$. One can find two of these examples in Algorithm 5.7.

- For all sending event $e_! \in E_!$, $\mathsf{NotConflict}(e_!, \mathsf{Dep}(e_!))$ returns $\{\mathcal{B}_!(e_!)\}$.
- For all sending event $e_! \in E_!$, $\mathsf{NotConflict}(e_!, \mathsf{Dep}(e_!) - 1)$ returns
  - $\emptyset$ if $|v| = 0$ and $\mathsf{Dep}(e) = 1$, or otherwise
  - $\mathsf{Max}_{\leq}(E^f)$ if $\mathsf{Dep}(e) = |v| + 1$, and $\{\mathcal{B}_!(e_!)\}$ if $\mathsf{Dep}(e) > |v| + 1$.
- For all receiving event $e_? \in E_?$, $\mathsf{NotConflict}(e_?, \mathsf{Dep}(e_?))$ returns
  - $\emptyset$ if $e_? \in E^f$, i.e. $\mathsf{Dep}(e_?) \leq |v|$, and
  - $\{\mathcal{B}_?(e_?)\}$ otherwise.
- For all receiving event $e_? \in E_?$ such that $\mathsf{Dep}(e_?) \geq |v|$, $\mathsf{NotConflict}(e_?, \mathsf{Dep}(e_?) + 1)$ returns
  - $\mathsf{Min}_{\leq}(E_!)$ if $\mathsf{Dep}(e_?) = |v|$, i.e. $e_?$ is the maximal event in E^f, and
  - $\lessdot(\mathcal{B}_?(e_?)) \cap E_!$, i.e. the set of sending successors of $\mathcal{B}_?(e_?)$, otherwise.

Notice that when calling $\mathsf{NotConflict}$ on a prefix $\widehat{\mathcal{E}} = (\widehat{E}, \widehat{\leq}, \widehat{\#}, \widehat{\mathcal{L}}, \widehat{\mathcal{M}})$ of (M,v)-$\mathcal{CP} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, the return set may be not complete and may even be the empty set. Because $\widehat{E}$ is only a subset of E. For instance with some receiving event $e_? \in \widehat{E}$, if $\mathcal{B}_?(e_?)$ is maximal in the poset $(\widehat{E}, \widehat{\leq})$ then $\widehat{\lessdot}(\mathcal{B}_?(e_?)) = \emptyset$ and consequently, $\mathsf{NotConflict}(e_?, \mathsf{Dep}(e_?) + 1) = \emptyset$.

Now, let us give some details of Algorithm 5.7 and show that it is not far from Algorithm 5.6 because M-$\mathcal{CP}$ is just a particular case of (M,v)-$\mathcal{CP}$ where $v = \varepsilon$. The

algorithm is split into two parts corresponding to the type of its input event e. In both case, one start by removing e from the possible extension PE (line 4).

Algorithm 5.7: Function $\mathsf{Extend}_{\mathcal{M}v}$ for (M, v)-$\mathcal{CP}$

```
1  function Extend_Mv(Ê, PE, e)
2  var Y
3  begin
4      PE := PE \ {e}
5      if L(e) ∈ !M then
6          let m ∈ M s.t. L̂(e) = !m
7          for each m' ∈ M do
8              (Ê, PE, e_!m') := Create(Ê, PE, {e}, !m')
9          end for
10         if (|v| = 0) & (Dep(e) = 1) then
11             (Ê, PE, e_?m) := Create(Ê, PE, {e}, ?m)
12         else if exists e_?m' ∈ (NotConflict(e, Dep(e) − 1) \ PE) then
13             (Ê, PE, e_?m) := Create(Ê, PE, {e, e_?m'}, ?m)
14         end if
15     else
16         if Dep(e) < |v| then
17             let m' be the (Dep(e) + 1) message of v
18             (Ê, PE, e_?m') := Create(Ê, PE, {e}, ?m')
19         else
20             Y := NotConflict(e, Dep(e) + 1) \ PE
21             for each e_!m' in Y do
22                 (Ê, PE, e_?m') := Create(Ê, PE, {e, e_!m'}, ?m')
23             end for
24         end if
25     end if
26     return (Ê, PE)
27 end function
```

When extending a sending event (lines 5-14), one gets the message which corresponds to the label of e and denotes it by m (line 6), i.e. $\mathcal{L}(e) = !m$. Accordingly, let us denote e by $e_{!m}$. The "for each" loop at lines 7-9 in Algorithm 5.7 does the same thing as the one at lines 8-11 in Algorithm 5.6. This loop simply inserts $|M|$ sending events which are direct successors of $e_{!m}$ into the prefix $\widehat{\mathcal{E}}$. Notice that these sending events has only one direct predecessor which is e. Since e is no more in PE, the insertion of such sending events respects to the invariant $I3$ of $\mathsf{Extend}_{\mathcal{M}v}$'s correctness by Definition 5.1.3. Now, look at whether a receiving successor, denoted by $e_{?m}$ of $e_{!m}$ has to be created (lines 10-14). In the first case (lines 10-11) where v is the empty word, i.e. $|v| = 0$, and $e_{!m}$ is a minimal event w.r.t. causality, i.e. $\mathsf{Dep}(e_{!m}) = |v| + 1 = 1$, sending event $e_{?m}$, which has only $e_{!m}$ as a direct predecessor, is inserted. This corresponds to the case at lines 13-14 in Algorithm 5.6. In the second case, since v is not the empty word, the event $e_{?m}$ must be a direct successor of another receiving event $e_{?m'}$. As explained previously, we have $\mathsf{NotConflict}(e_{!m}, \mathsf{Dep}(e_{!m}) - 1) \subseteq \{e_{?m'}\}$ (line 12). Notice that $e_{?m'}$ is either the maximal event w.r.t. causality in E^f (when $e_{!m}$ is a minimal one, i.e.

$\mathsf{Dep}(e_{!m}) = |v| + 1$) or the unique event in the set $\mathcal{B}_?(\gtrdot(e_{!m}))$. No matter what $e_{?m'}$ corresponds to, event $e_{?m}$ is added (line 13) if and only if $e_{?m'}$ is already extended. One can see that $\mathsf{NotConflict}(e_{!m}, \mathsf{Dep}(e_{!m}) - 1)$, when $v = \varepsilon$, corresponds more or less to the set $\widehat{\lessdot}(e_{!m'})$ in line 17 in Algorithm 5.6. And if $e_{?m'}$ exists then an common direct successor of e and $e_{?m'}$ should be added to the prefix $\widehat{\mathcal{E}}$ in order to reestablish the invariant $I3$.

When extending a receiving event (lines 15-24), denoted by $e_{?m}$, there are also two cases. The first one which does not exists in Algorithm 5.6, is when $e_{?m} \in E^f$ and $e_{?m}$ is not the maximal one w.r.t. causality in $\widehat{E}^f$ (lines 16-18), i.e. $\mathsf{Dep}(e) < |v|$. One simply creates the unique successor of $e_{?m}$. This successor is also an event in E^f and has only one direct predecessor which is $e_{?m} \notin \mathsf{PE}$. Hence, the invariant $I3$ is guaranteed. In the second case, let us denote $S_! = \mathsf{NotConflict}(e_{?m}, \mathsf{Dep}(e_{?m}) + 1)$. If $e_{?m}$ is the maximal event w.r.t. causality in E^f then $S_!$ is thus the set of minimal sending events, i.e. $S_! = \mathsf{Min}_{\leq}(E_!)$. Otherwise, we have $S_! = \lessdot(\mathcal{B}_?(e_{?m})) \cap E_!$. In both sub-cases, extending $e_{?m}$ requires that one creates the receiving successor of each event $e_{!m'}$ in $S_!$ if $e_{!m'}$ has been extended, i.e. $e_{!m'} \in Y = (S_! \setminus \mathsf{PE})$ (the loop's condition in line 21). Particularly, if $|v| = 0$, the set Y is obvious the set $\widehat{\lessdot}(e_{!m}) \setminus \mathsf{PE} = \widehat{\lessdot}(\mathcal{B}_!(e_{!m})) \setminus \mathsf{PE}$ used in the loop's criterion in line 23 in Algorithm 5.6. Precise instructions for adding these successors (line 22) is the same as in the loop at lines 23-26 in Algorithm 5.6.

As usual, in both cases, the new prefix $\widehat{\mathcal{E}}$ and possible extension PE are returned (line 26).

(M, v, b)-causality processes

The (M, v, b)-$\mathcal{CP}$ defined in Definition 3.3.36 on page 52 may also constructed by our unfolding algorithm. One can slightly modify the Algorithm 5.7 to have an algorithm of function Extend for (M, v, b)-$\mathcal{CP}$. Because, by definition, (M, v, b)-$\mathcal{CP}$ differs from (M, v)-$\mathcal{CP}$ only on the causality which comes from the constraint of boundedness.

$$\leq_b = \{\langle e, f\rangle \in ((E \times E) \setminus \#) \,/\, \mathsf{Dep}(f) \geq \mathsf{Dep}(e) + b\}$$

As illustrated in Figure 3.13 on page 51, this causality based on pairs of a sending event $e_! \in E_!$ and a receiving event $e_? \in E_?$ such that $e_! \,\overline{\#}\, e_?$ and $\mathsf{Dep}(e_!) = \mathsf{Dep}(e_?) + b$. Intuitively, in order to guarantee the bound of b, $e_!$ must be a successor, and more precisely, a direct successor of $e_?$, i.e. $e_? \lessdot e_!$. This fact means that one can insert into the FIFO channel a message indexed by $\mathsf{Dep}(e_!)$ if and only if the message indexed by $\mathsf{Dep}(e_?)$ has been released. Because, the bounded FIFO channel can contains at most b messages at a time. The difference between (M, v, b)-$\mathcal{CP}$ and (M, v)-$\mathcal{CP}$ may be depicted, in another way, by using its predecessor relations.

$$\begin{aligned}\lessdot_b &= \lessdot \setminus \lessdot' \\ &= \{\langle e_?, e_!\rangle \in (E_? \times E_!) \,/\, e_! \,\overline{\#}\, e_? \text{ and } \mathsf{Dep}(e_!) = \mathsf{Dep}(e_?) + b\}\end{aligned}$$

Here, $\lessdot$ and $\lessdot'$ are respectively the predecessor relations of (M, v, b)-$\mathcal{CP}$ and its corresponding (M, v)-$\mathcal{CP}$ (see Definition 3.3.36 on page 52). We are going to show how to modify Algorithm 5.5 as well as Algorithm 5.7 to have adapted algorithms for (M, v, b)-$\mathcal{CP}$. Intuitively, for creating any sending event $e_!$, one needs to take care of not only the direct sending predecessor $e'_!$ of $e_!$, i.e. $e'_! \lessdot' e_!$, but also the corresponding receiving event $e_?$ according to $\lessdot_b$, i.e. $e_? \lessdot_b e_!$. The event $e'_!$ as well as the event $e_?$ may not exist for some event $e_!$. Modification in the function Extend for (M, v, b)-$\mathcal{CP}$ concerns only instructions

for adding sending events to the prefix. And instructions for adding receiving events are the same as the ones in Algorithm 5.7. Notice that we still use two functions Dep and NotConflict previously described.

Remark: Given any (M, v, b)-$\mathcal{CP}$, the bound parameter b must not be zero. Moreover, due to the boundedness, the initial word v can not has a length greater than b, i.e. $|v| \leq b$.

Consider now the function Init. One initializes the prefix $\widehat{\mathcal{E}}$ by the same way as in the function $\mathsf{Init}_{\mathcal{M}v}$ except for minimal sending messages w.r.t. causality. Because for the particular case where the length of v is b, all sending events must be preceded by the maximal receiving event E^f [1] w.r.t. the causality $\leq$. Formally, if $\{e_?\} = \mathsf{Max}_{\leq}(E^f)$ then $\{e_?\} \times \mathsf{Min}_{\leq}(E_!) \subset \lessdot$. As a consequence, the "for each" loop at lines 5-7 in Algorithm 5.5 must be enclosed by a test as in the following.

---

if $|v| < b$ **then**
  for each $m \in M$ **do**
    $(\widehat{\mathcal{E}}, \mathsf{PE}, e_{!m}) := \mathsf{Create}(\widehat{\mathcal{E}}, \mathsf{PE}, \emptyset, !m)$
  end for
end if

Notice that if $|v| = b$, the return prefix $\widehat{\mathcal{E}}$ of Init() contains only the minimal event f of E^f which is a receiving one. It is done as the same manner as in function $\mathsf{Init}_{\mathcal{M}v}$. The minimal sending event in $\mathsf{Min}_{\leq}(E_!)$ will be added to the prefix $\widehat{\mathcal{E}}$ when extending f afterward.

Next, there are two modifications on $\mathsf{Extend}_{\mathcal{M}v}$ according to the type of argument event e of function Extend. First, when e is a sending event, i.e. $e \in E_!$, e has $|M|$ sending successors and one receiving successor in (M, v, b)-$\mathcal{CP}$ which are all direct ones. A such sending successors $e_!$, on its turn, have two common direct predecessors which are e and, possibly, another receiving event $e_?$. The first one is due to the usual total order of messages in channel while the second one is in order to guarantee the boundedness, i.e. $e_? \lessdot_b e_!$. Let us denote the set of direct sending successors of e by $S_!$, we have:

- $\{e_?\} = \mathsf{NotConflict}(e, \mathsf{Dep}(e) - b + 1)$, and
- for all $e_! \in S_!$, $\gtrdot(e_!) = \{e, e_?\}$.

Therefore, one must take care of the existence of $e_?$. If $\mathcal{D}(e) < b$, and consequently $\mathcal{D}(e_!) \leq b$ for all $e_! \in S_!$, then such $e_?$ does not exists in $\mathcal{E}$, i.e. $\gtrdot_b(e_!) = \emptyset$ for all $e_! \in S_!$. In other words, every event $e_! \in S_!$ has only one direct predecessor which is e, and should be add to the prefix $\widehat{\mathcal{E}}$. Otherwise, i.e. $\mathcal{E}$ contains corresponding event $e_?$, the invariant $I3$ in Definition 5.1.1 for Extend's correctness requires that $e_! \in S_!$ is added iff $e_?$ is in the prefix $\widehat{\mathcal{E}}$ and has been extended, i.e. $e_? \in (\widehat{E} \setminus \mathsf{PE})$. The "for each loop" at lines 7-9 in Algorithm 5.7 which generates these events $S_!$ should be modified as follows:

---

if $(\mathsf{Dep}(e) < b)$ **then**
  for each $m' \in M$ **do**
    $(\widehat{\mathcal{E}}, \mathsf{PE}, e_{!m'}) := \mathsf{Create}(\widehat{\mathcal{E}}, \mathsf{PE}, \{e\}, !m')$
  end for
else if exists $e_? \in (\mathsf{NotConflict}(e, \mathsf{Dep}(e) - b + 1) \setminus \mathsf{PE})$ **then**
  for each $m' \in M$ **do**

[1] In (M, v, b)-$\mathcal{CP}$ as well as in (M, v)-$\mathcal{CP}$, E^f is the event sets of the corresponding flushing process.

```
            (Ê, PE, e_{!m'}) := Create(Ê, PE, {e, e_?}, !m')
        end for
    end if
```

When no sending successors is created, it intuitively means that e is extended sooner than $e_?$. An eventual calling $\mathsf{Extend}(e_?)$ will take care of creating events in $S_!$. As discussed above, the boundedness constraint has no influence on whether the direct receiving successor of e is added. Hence, instructions concerning this receiving event at lines 10-14 in Algorithm 5.7 remain unchanged in our function Extend for (M, v, b)-$\mathcal{CP}$.

Second, when e is a receiving event, its direct receiving successors are added to the prefix $\widehat{\preccurlyeq}$ by the same manner as shown at lines 16-24 in Algorithm 5.7. In addition, let $X_!$ denote the set $\prec_b(e) \subset E_!$, we are in need of some new instructions for adding events in $S_!$ if possible. Once again, every event in $\prec_b(e)$ has two direct predecessors which are e and another sending event $e_!$ in the set $\mathsf{NotConflict}(e, \mathsf{Dep}(e)+b-1)$. Therefore, events in $S_!$ is created based on its second direct predecessors. These instructions are shown below and may be inserted, for instance, into Algorithm 5.7 just before line 16.

```
    for each e_! in (NotConflict(e, Dep(e) + b − 1) \ PE) do
        for each m' ∈ M do
            (Ê, PE, e_{!m'}) := Create(Ê, PE, {e, e_!}, !m')
        end for
    end for
```

Each event $e_! \in \mathsf{NotConflict}(e, \mathsf{Dep}(e) + b - 1) \setminus \mathsf{PE})$ gives rise to $|M|$ new successors whose labels are pairwise different due to the inner loop. And because $e_! \in (\widehat{E} \setminus \mathsf{PE})$ as a consequence of the outer loop's criterion, the invariant $I3$ for Extend's correctness is respected. In all cases, new events added to the prefix are also be inserted into the possible extension PE. The function Extend finally returns $\widehat{\mathcal{E}}$ and PE as usual.

Estimation of time complexity

The complexity of function Extend for (M, v)-$\mathcal{CP}$ as well as (M, v, b)-$\mathcal{CP}$ depends on the complexity of the function Dep and especially of the function NotConflict. One can have a function Dep of time complexity $O(1)$ by relating a depth value to each event since it is created. This manner does not increase the space complexity of our algorithms. However, time complexity of $\mathsf{NotConflict}(e, d)$ is somehow in function of d and the depth of event e.

Let us first consider only two particular cases of parameters e and d in Algorithm 5.7. First, e is a sending event in $E_!$ and $d = \mathsf{Dep(e)} - 1$ (line 12). The computation of NotConflict may be more or less instructions at lines 16-17 in Algorithm 5.6. Intuitively, one gets the direct predecessor of e, denoted by $e_!$, and then return the receiving direct successor of $e_!$ if exists. It thus has time complexity $O(1)$. Second, e is a receiving event in $E_?$ and $d = \mathsf{Dep}(e) + 1$ (line 20). Function NotConflict returns a set of at most $|M|$ sending events and its time complexity is proportional to $O(|M|)$. This complexity comes from the corresponding instructions at lines 23-24 in Algorithm 5.6 while assuming that successor set as well as label of an event can be returned in $O(1)$ time. Therefore, one can have an implementation of Dep and NotConflict so that Algorithm 5.7 for (M, v)-$\mathcal{CP}$ has a same time complexity as Algorithm 5.6 for M-$\mathcal{CP}$, which is $O(|M|)$. Notice that the time complexity here is of function $\mathsf{Extend}_{\mathcal{M}v}$ and not of the global unfolding algorithm

(Algorithm 5.1) which must also depends on $|v|$.

Now, consider the algorithm of function Extend for (M, v, b)-$\mathcal{CP}$. Its most expensive part in terms of time complexity is intuitively the third modification described above. Since e is a receiving event in $E_?$ and $d = \mathsf{Dep}(e) + b - 1$, function NotConflict creates the set $S_!$ of sending events which are in causal with the sending predecessor $e_!$ of e. Notice that in (M, v, b)-$\mathcal{CP}$, events in causal with $e_!$, i.e. $\leq(e_!)$, with the causality between them forms an intuitive tree of which the root is $e_!$. Moreover, it follows from the fact $\mathsf{Dep}(e_!) = \mathsf{Dep}(e)$ and $\forall e'_! \in S_! : \mathsf{Dep}(e'_!) = \mathsf{Dep}(e) + b - 1$ that $S_!$ is the set of all node of distance $(b-1)$ from the root $e_!$. Because each node of this tree may also have $|M|$ successors, in the worst case, the size of $S_!$ is $|M|^{b-1}$. Therefore, due to the unique nested loop in the algorithm of function Extend for (M, v, b)-$\mathcal{CP}$, this function Extend has time complexity of $O(|M|.|M|^{b-1}) = O(|M|^b)$.

5.3 Synchronized products' unfolding

Our idea of constructing synchronized products of labeled event structures is similar to that of well-known unfolding algorithm in other works [McM95a, ER99, KK03]. We assume that one does have algorithms for constructing labeled event structures of the components by means of function Extend described in Section 5.2.

Given n labeled event structures $\mathcal{E}_1 = (E_1, \leq_1, \#_1, \mathcal{L}_1, \mathcal{M}_1), \ldots, \mathcal{E}_n = (E_n, \leq_n, \#_n, \mathcal{L}_n, \mathcal{M}_n)$ and an action set $\Sigma \in \otimes_\varepsilon(\mathsf{Codom}(\mathcal{L}_1), \ldots, \mathsf{Codom}(\mathcal{L}_n))$. The synchronized product $\mathcal{E}_{\mathcal{SP}} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ of $\mathcal{E}_1, \ldots, \mathcal{E}_n$ w.r.t. Σ (Definition 3.3.44 on page 57) is a part of the maximal product of n event structures $(E_1, \leq_1, \#_1), \ldots, (E_n, \leq_n, \#_n)$ (Definition 3.3.39 on page 54) that satisfies the synchronization Σ, i.e. constructing events e satisfying $\mathcal{L}_\mathcal{V}(e) \in \Sigma$. Recall that both the labeling function $\mathcal{L} = \mathcal{L}_\mathcal{V}$ and the marking function $\mathcal{M} = \mathcal{M}_\mathcal{V}$ are well defined by those of component labeled event structures and the function $\mathcal{V}$ (see Notation 3.3.43 on page 57). Therefore, in this section, we are going to show how to algorithmically construct $\mathcal{E}_{\mathcal{SP}}$ in terms of product of event structures. In other words, our algorithms compute only prefixes $(\widehat{E}, \widehat{\leq}, \widehat{\#})$ of $(E, \leq, \#)$ and the corresponding vectors $\widehat{\mathcal{V}} \subseteq \mathcal{V}$.

Recall that, by Definition 3.3.44, every event $e \in E$ corresponds to an unique pair $\langle C, v\rangle$ where $C = {>}(e)$ and $v = \mathcal{V}(e) \in \otimes_\varepsilon(E_1, E_2, \ldots, E_n)$. Given an $\widehat{E}$-prefix of $\mathcal{E}_{\mathcal{SP}}$, the unfolding algorithm intuitively finds new pairs $\langle C, v\rangle$ that represent events which may be added to the prefix. Since v must satisfy the labeling constraint: $\langle \mathcal{L}_1(v{\downarrow_1}), \ldots, \mathcal{L}_n(v{\downarrow_n})\rangle = \mathcal{L}(e) \in \Sigma$, one groups such pairs $\langle C, v\rangle$ into disjoint subsets based on different actions $a \in \Sigma$, and searches these subsets separately. The computation of a pair $\langle C, v\rangle$ corresponding to an action a consists more or less of:

1. Initializing C, may be by the empty set.
2. Successively finding $v{\downarrow_i} = e_i \in E_i$ for some e_i satisfying $\mathcal{L}_i(e_i) = a{\downarrow_i}$ for all i such that $a{\downarrow_i} \neq \varepsilon$.
3. For every found $v{\downarrow_i} \neq \varepsilon$, successively enlarging C by adding events in E to C in order to have that C is still a configuration in $\mathcal{C}_{\mathcal{E}_{\mathcal{SP}}}$ and $v{\downarrow_i}$ is an extension of $\mathcal{V}(C){\downarrow_i}$ in $\mathcal{E}_i$.

In our algorithm described later, the two functions ConfigVectorSet and ConfigVectorSet_i are respectively dedicated to the second and the third sub-processes above. ConfigVectorSet successively calls ConfigVectorSet_i for all index i such that $a{\downarrow_i} \neq \varepsilon$. Both ConfigVectorSet and ConfigVectorSet_i may fail that means there is no pair $\langle C, v\rangle$ corresponding to the action a. In the first case, it is because, for instance, there is no event labeled by $a{\downarrow_i}$

in $\mathcal{E}_i$; while in the second case, it is because there is no configuration $C' \supseteq C$ in the $\widehat{E}$-prefix of $\mathcal{E}$ such that $\mathcal{V}(C')\downarrow_i \vdash_i e_i$ for previously obtained e_i.

By calling ConfigVectorSet, the function $\mathsf{Init}_{\mathcal{SP}}$ as well as the function $\mathsf{Extend}_{\mathcal{SP}}$ for synchronized products has to initialize action a as well as configuration C so that all and only pairs $\langle C, v\rangle$ corresponding to new events $e \in (E \setminus \widehat{E})$ will be found. In the same way as function Extend for component labeled event structures, it is due to the use of the possible extension PE and by limiting C to subsets of $\widehat{E} \setminus \mathsf{PE}$.

Before giving details on ConfigVectorSet_i as well as ConfigVectorSet, let us introduce the notion of *config-vector* which is the base type of these functions' parameter.

Notation 5.3.1. Given $n \in \mathbb{N}$ and n sets $X_1, X_2, \ldots, X_n$, for any $x \in \otimes_\varepsilon(X_1, X_2, \ldots, X_n)$, we denote $I(x)$ the set of indices $i \in \{1, 2, \ldots, n\}$ satisfying $x\downarrow_i \neq \varepsilon$.

Definition 5.3.2 (Config-vector)**.** Given a synchronized product $\mathcal{E}_{\mathcal{SP}}$ of some n labeled event structures $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ w.r.t. Σ. A *config-vector* of $\mathcal{E}_{\mathcal{SP}}$ is any triple $\langle C, v, a\rangle$ where $C \subseteq E$, $v \in \otimes_\varepsilon(E_1, E_2, \ldots, E_n)$, and $a \in \Sigma$ such that:

1. C is a configuration in $\mathcal{C}_{\mathcal{E}_{\mathcal{SP}}}$,
2. for all $i \in \{1, 2, \ldots, n\}$, either $v\downarrow_i = \varepsilon$ or $\mathcal{L}_i(v\downarrow_i) = a\downarrow_i$.
3. for all $i \in \{1, 2, \ldots, n\}$, if $v\downarrow_i \neq \varepsilon$ then $v\downarrow_i \notin \mathcal{V}(C)\downarrow_i$ and $v\downarrow_i$ is not in conflict with any event in C, i.e. $\{v\downarrow_i\} \overline{\#}_i^s \mathcal{V}(C)\downarrow_i$.

A config-vector $\langle C, v, a\rangle$ is *partially complete* for an index i if $v\downarrow_i$ is an extension event of $\mathcal{V}(C)\downarrow_i$ in $\mathcal{E}_i$, i.e. $\mathcal{V}(C)\downarrow_i \vdash_i v\downarrow_i$.

Definition 5.3.3 (Complete config-vector)**.** A config-vector $\langle C, v, a\rangle$ is *complete* if

1. $I(v) = I(a)$,
2. $\langle C, v, a\rangle$ is partially complete for every index $i \in I(v)$, and
3. for every event $e \in \mathsf{Max}_{\leq}(C)$, there exists an index i such that $\mathcal{V}(e)\downarrow_i \lessdot_i v\downarrow_i$.

Let us take an example of a config-vector $cv = \langle {>}(e), \mathcal{V}(e), \mathcal{L}(e)\rangle$ where e is a given event in $\widehat{E}$. By definition, cv is complete. Let C be any configuration so that e is one of its extension event, i.e. $C \vdash e$. One can verify that the triple $\langle C, \mathcal{V}(e), \mathcal{L}(e)\rangle$ is also a config-vector which is partially complete for every index $i \in I(\mathcal{V}(e)) = I(\mathcal{L}(e))$. However, the third property of a complete config-vector in Definition 5.3.3 may not hold. The reason is that C may contain some event e' which is concurrent with e. In other words, the this property requires somehow that C is equal to ${>}(e)$.

Recall that, in the synchronized product $\mathcal{E}_{\mathcal{SP}}$, there may exist other events f also satisfying $\mathcal{V}(f) = \mathcal{V}(e)$, consequently, $\mathcal{L}(f) = \mathcal{L}(e)$, and $\langle {>}(f), \mathcal{V}(e), \mathcal{L}(e)\rangle$ is a complete config-vector. Therefore, for a same vector v corresponding to some label a, one could have many complete config-vectors $\langle C, v, a\rangle$.

Lemma 5.3.4. *Given a synchronized product $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ of n labeled event structures $\mathcal{E}_1 = (E_1, \leq_1, \#_1, \mathcal{L}_1, \mathcal{M}_1), \ldots, \mathcal{E}_n = (E_n, \leq_n, \#_n, \mathcal{L}_n, \mathcal{M}_n)$ w.r.t. some synchronization constraint Σ. Let v be any vector in $\otimes_\varepsilon(E_1, \ldots, E_n)$ satisfying that $\langle \mathcal{L}_1(v\downarrow_1), \ldots, \mathcal{L}_n(v\downarrow_n)\rangle = a$ for some given $a \in \Sigma$. Let CS be the set of configurations C such that $\langle C, v, a\rangle$ is a complete config-vector. Then*

$$CS = \{{>}(e) \,/\, e \in E \text{ and } \mathcal{V}(e) = v\}$$

Proof. Let C be any configuration in CS. Let e' be any maximal event w.r.t. the causality in C, i.e. $e' \in \mathsf{Max}_{\leq}(C)$. It follows from Definition 5.3.2 that there exists an index $i \in \{1, 2, \ldots, n\}$ satisfying $\mathcal{V}(C)\downarrow_i \vdash_i v\downarrow_i$, we have $\mathcal{V}(e')\downarrow_i \lessdot_i v\downarrow_i$. Due to the

maximality w.r.t. isomorphism of $\mathcal{E}$, by Definition 3.3.44 of synchronized products of labeled event structures, there exists an event $e \in E$ such that $\mathcal{V}(e) = v$ and $e' \lessdot e$ for all $e' \in \mathsf{Max}_{\leq}(C)$. One can simply write $C = {>}(e)$ for some event $e \in E$.

Conversely, it is straightforward from Definition 5.3.2 and Definition 5.3.3 that, for every event e, if $\mathcal{V}(e) = v$ then $\langle {>}(e), v, a\rangle$ is a complete config-vector. This lemma is thus proved. □

Remark: The two functions ConfigVectorSet_i and ConfigVectorSet in the next subsections are recursive functions. In order to compute config-vectors of $\mathcal{E}_{\mathcal{SP}}$, they have access to the being constructed prefix $\widehat{\mathcal{E}}_{\mathcal{SP}}$ of the synchronized product $\mathcal{E}$ as well as its corresponding prefixes $\widehat{\mathcal{E}}_i$, $i = \{1, 2, \ldots, n\}$, of all component labeled event structures $\mathcal{E}_i$. However, these functions do not modify or add anything to the prefixes. This is done in the function Extend.

5.3.1 Function ConfigVectorSet_i

The function ConfigVectorSet_i(F, i, C, v, a) is given in Algorithm 5.8 and has 5 parameters. The first parameter is a downward-closed set of events in $\widehat{E}$ while the second one is an index in $I(a)$. The three last parameters form a config-vector, and there are two additional conditions on the input of ConfigVectorSet_i:

- $v{\downarrow_i}$ is some event in $\widehat{E}_i$, i.e. $v{\downarrow_i} \neq \varepsilon$, and as a consequence from Definition 5.3.2, $v{\downarrow_i} = a{\downarrow_i}$;
- C is a subset of F, i.e. $C \subseteq F$.

The output of this function is the set of all config-vectors $\langle C', v, a\rangle$ which are partially complete for the index i, moreover, C is a subset of C', and C' is a subset of F at the same time, i.e. $C \subseteq C' \subseteq F$.

Let us explain intuitive ideas of Algorithm 5.8. The set P_i obtained at line 3 is the set of $v{\downarrow_i}$'s direct predecessors in component $\widehat{\mathcal{E}}_i$ which are not included in $\widehat{\mathcal{V}}(C){\downarrow_i}$. If this set P_i is empty (line 4), by definition, the config-vector $\langle C, v, a\rangle$ is already partially complete for the index i, hence one just returns the singleton $\{\langle C, v, a\rangle\}$ (line 5). Otherwise (lines 6-17), one needs to add events in F to C in order to satisfy the partial completeness in the component i. These added events, if exist, correspond 1-to-1 to the component events in P_i. The finding procedure is recursive.

The algorithm takes an event e'_i in P_i (line 7). Intuitively, e'_i is a missing predecessor of $v{\downarrow_i}$ so that $\langle C, v, a\rangle$ is still not partially complete for index i. The set X obtained in line 8 is the set of all events e' in F which is related to e'_i and is not in conflict with events in C. It worth to notice that by definition of synchronized product of event structures (Definition 3.3.39 on page 54), e' can not be in C and, as a consequence, $(C \cap X) = \emptyset$ because $\widehat{\mathcal{V}}(e'){\downarrow_i} = e'_i \in P_i$ and $P_i \cap \widehat{\mathcal{V}}(C){\downarrow_i} = \emptyset$.

If X is empty (line 9), there is no partially complete config-vector $\{C', v, a\}$ where $C' \supseteq C$. The function simply returns the empty set (line 10). Otherwise (lines 11-16), for each event $e' \in X$, one tries to search partially complete config-vector from $\langle C \cup \widehat{\geq}(e'), v, a, i\rangle$ by calling ConfigVectorSet_i itself. Found config-vectors are inserted into the set CVS (line 14) which will be finally returned by the function ConfigVectorSet_i (line 16).

Lemma 5.3.5. *Let* CVS *be the return set of some call* ConfigVectorSet_i(F, i, C, v, a).

Algorithm 5.8: Function ConfigVectorSet_i

```
1  function ConfigVectorSet_i(F, i, C, v, a)
2  begin
3     P_i := ⩾̂_i(v↓_i) \ V̂(C)↓_i
4     if P_i = ∅ then
5        return {⟨C, v, a⟩}
6     else
7        take a component event e'_i in P_i
8        X := {e' ∈ F / V̂(e')↓_i = e'_i and {e'} #̂^s C}
9        if X = ∅ then
10          return ∅
11       else
12          CVS := ∅
13          for each e' ∈ X do
14             CVS := CVS ∪ ConfigVectorSet_i(F, i, C ∪ ⩾̂(e'), v, a)
15          end for
16          return CVS
17       end if
18    end if
19 end function
```

- CVS *contains only config-vectors* $\langle C', v, a\rangle$ *which are partially complete for the component* i*; and*
- *the projection of* CVS *on the first component, i.e.* CVS $\downarrow_1$*, is equal to the set* $\mathsf{Min}_{\subseteq}\{C' \in \mathcal{C}_{\widehat{\mathcal{E}}_{\mathcal{SP}}} / C \subseteq C' \subseteq F$ *and* $\widehat{\mathcal{V}}(C')\downarrow_i \vdash_i v\downarrow_i\}$.

Proof. As the invariant of ConfigVectorSet_i's input, we have $C \subseteq F$ and $\langle C, v, a\rangle$ is a config-vector. Since event $v\downarrow_i$ has only finite direct predecessors in $\widehat{\mathcal{E}}_i$, we will prove this lemma by induction on the size $k = |\widehat{\geqslant}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C)\downarrow_i|$.

- The first property: When $k = 0$, due to lines 4-5 in Algorithm 5.8, we have CVS $= \{\langle C, v, a\rangle\}$, where $\langle C, v, a\rangle$ is a config-vector. It follows from Definition 5.3.2 for config-vector $\langle C, v, a\rangle$ that $v\downarrow_i \notin \widehat{\mathcal{V}}(C)\downarrow_i$ and $\{v\downarrow_i\} \overline{\widehat{\#}_i^s}\, \widehat{\mathcal{V}}(C)\downarrow_i$. Hence, $\widehat{\mathcal{V}}(C)\downarrow_i \vdash_i v\downarrow_i$ because $\widehat{\geqslant}_i(v\downarrow_i) \subseteq \widehat{\mathcal{V}}(C)\downarrow_i$. As a consequence, $\langle C, v, a\rangle$ is partially complete for component i by definition. Suppose that this property is correct for some $k \geq 0$ and $P_i = \widehat{\geqslant}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C)\downarrow_i$ (line 3) has $k+1$ events. There are two cases depending on the set X obtained in line 8. First, if X is empty, then CVS contains nothing, and consequently, this property is true. Second, if X is not empty. For every event e' taken in the loop (line 13), by definition of X in line 8, we have that the downward-closed set $\widehat{\leq}(C \cup \{e'\}) = C \cup \widehat{\leq}(e')$ is a configuration. And $(C \cup \widehat{\leq}(e')) \subseteq F$ because F also downward-closed w.r.t. the causality $\widehat{\leq}$ and $e' \in F$. Event e' can not be in C because $\widehat{\mathcal{V}}(e') = e'_i \notin \widehat{\mathcal{V}}(C)\downarrow_i$. Let us denote $C'' = C \cup \widehat{\geq}(e)$, $\langle C', v, a\rangle$ is thus a config-vector. Once again, it follows from the definition of X that $(\widehat{\geqslant}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C'')\downarrow_i) = (\widehat{\geqslant}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C)\downarrow_i \setminus e'_i)$, and consequently, $|(\widehat{\geqslant}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C'')\downarrow_i)| = k+1-1 = k$. By induction hypothesis, calling ConfigVectorSet_i(F, i, C'', v, a) returns only partially complete config-vector $\langle C', v, a\rangle$ for component i. So do the final return set CVS in line 16. The first property is thus proved.

- The second property: Let CS denote the set $\{C' \in \mathcal{C}_{\widehat{\mathcal{E}}_{\mathcal{SP}}} / C \subseteq C' \subseteq F \text{ and } \widehat{\mathcal{V}}(C')\downarrow_i \vdash_i v\downarrow_i\}$. When $k = 0$, i.e. $\widehat{\gtrdot}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C)\downarrow_i = \emptyset$, the return set CVS contains only $\langle C, v, a\rangle$ (line 5). Thanks to the first property, $\langle C, v, a\rangle$ is partially complete for component i. By Definition 5.3.2, we have $\widehat{\mathcal{V}}(C')\downarrow_i \vdash_i v\downarrow_i$. Hence, C is included in CS, moreover, C is the minimal configuration in CS w.r.t. the inclusion order $\subseteq$. Because every configuration C' in CS satisfies $C' \supseteq C$ by definition of CS. This property thus holds in the base case. Suppose that it holds for some number k, and we have $|\widehat{\gtrdot}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C)\downarrow_i| = k + 1$.

  First, let C' be any configuration in the set $\mathsf{Min}_{\subseteq} CS$, we will prove that $C' \in \mathsf{CVS}\downarrow_1$ (1). Let e'_i be the event in $\widehat{\mathcal{E}}_i$ obtained at line 7. It follows from $\widehat{\mathcal{V}}(C')\downarrow_i \vdash_i v\downarrow_i$ that $e'_i \in \widehat{\mathcal{V}}(C')\downarrow_i$. Thanks to the exponentially downward closure of C', there exists an event $e' \in C'$ satisfying $\widehat{\mathcal{V}}(e') = e'_i$. Notice that e' can not be in C because $\widehat{\mathcal{V}}(e') = e'_i \notin \widehat{\mathcal{V}}(C)\downarrow_i$. In addition, since $C' \subseteq F$ is also configuration, e' must be in the set X obtained at line 8. By the for-loop at lines 13-15, $\langle C', v, a\rangle$ must be returned when calling $\mathsf{ConfigVectorSet\_i}(C \cup \widehat{\geq}(e'), v, a, i)$ (line 16) and is thus included in the final set CVS (line 18). Because, in the one hand, $C' \in \mathsf{Min}_{\subseteq}\{C' \in \mathcal{C}_{\widehat{\mathcal{E}}_{\mathcal{SP}}} / C \subseteq C' \subseteq F \text{ and } \widehat{\mathcal{V}}(C')\downarrow_i \vdash_i v\downarrow_i\}$ and $e' \in C'$ implies that $C' \in \mathsf{Min}_{\subseteq}\{C' \in \mathcal{C}_{\widehat{\mathcal{E}}_{\mathcal{SP}}} / (C \cup \widehat{\geq}(e')) \subseteq C' \subseteq F \text{ and } \widehat{\mathcal{V}}(C')\downarrow_i \vdash_i v\downarrow_i\}$; and in the other hand, the second set is the return value of $\mathsf{ConfigVectorSet\_i}(F, i, C \cup \widehat{\geq}(e'), v, a)$ due to the induction hypothesis (*) where $|\widehat{\gtrdot}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C \cup \widehat{\geq}(e'))\downarrow_i| = |\widehat{\gtrdot}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C)\downarrow_i \setminus \{e'_i\}| = k + 1 - 1 = k$. Therefore (1) is true.

  Second, let $\langle C', v, a\rangle$ be any config-vector in the return set CVS. We will prove that C' must be in the set CS and moreover, it is minimal w.r.t. inclusion order (2). This config-vector must come from some call $\mathsf{ConfigVectorSet\_i}(F, i, C \cup \widehat{\geq}(e'), v, a, i)$ in line 14 for some event e' obtained at lines 7-8. Once again, thanks to induction hypothesis (*), we must have $C' \supseteq (C \cup \widehat{\geq}(e')) \supset C$ and $\widehat{\mathcal{V}}(C')\downarrow_i \vdash_i v\downarrow_i$. Hence, C' is in CVS. Suppose that C' is not minimal w.r.t. inclusion order, that means there exists another configuration $C'' \in CS$ satisfying $C'' \subset C'$. Since $\widehat{\mathcal{V}}(C'')\downarrow_i \vdash_i v\downarrow_i$ there exists an event $e'' \in C''$ such that $\widehat{\mathcal{V}}(e'')\downarrow_i = \widehat{\mathcal{V}}(e')\downarrow_i \in \widehat{\gtrdot}_i(v\downarrow_i)$. It follows from $e' \in C'$, $e'' \in C'' \subset C'$ and the conflict-free of C' that e' and e'' must be the same event, i.e. $e' = e''$. As a consequence, $C'' \supseteq (C \cup \widehat{\geq}(e'))$. Therefore, C' is not minimal, w.r.t. inclusion order, in the return set of $\mathsf{ConfigVectorSet\_i}((C \cup \widehat{\geq}(e'))$ due to the existence of C''. It contradicts to the induction hypothesis. Hence, C' is thus minimal configuration in CVS, and (2) is true.

  From (1) and (2), the induction hypothesis (*) holds for all finite number k. And the lemma is thus proved by induction.

□

It is worth giving some details here on the minimality property w.r.t. the inclusion order $\subseteq$ of config-vectors returned by $\mathsf{ConfigVectorSet\_i}$. Let e be any event in the synchronized product, suppose that e has two direct predecessors f, g, i.e. $\gtrdot(e) = \{f, g\}$. Thanks to Lemma 5.3.4, $cv = \langle \gtrdot(e), \mathcal{V}(e), \mathcal{L}(e)\rangle$ is a complete config-vector. Let us denote $C = (\gtrdot(e)) \setminus \{f, g\}$. It is obvious that $\langle C, \mathcal{V}(e), \mathcal{L}(e)\rangle$, $\langle C \cup \{f\}, \mathcal{V}(e), \mathcal{L}(e)\rangle$ and $\langle C \cup \{g\}, \mathcal{V}(e), \mathcal{L}(e)\rangle$ are config-vectors. But they are not complete due to some component indices. Assume that $\langle C, \mathcal{V}(e), \mathcal{L}(e)\rangle$ is not partially complete for some component i but

$\langle C \cup \{f\}, \mathcal{V}(e), \mathcal{L}(e)\rangle$ is. When calling the function ConfigVectorSet_i$(F, i, C, \mathcal{V}(e), \mathcal{L}(e))$, for a given downward-closed set $F \supseteq (>(e))$, the value $\langle C \cup \{f\}, \mathcal{V}(e), \mathcal{L}(e)\rangle$ is returned. Although cv is partially complete for index i, it is not returned. One can see that $(C \cup \{f\}) \subset >(e)$, and it respects to the second property in Lemma 5.3.5. The minimality property on configurations of returned config-vectors intuitively means that one adds only necessary event to C in order to complete the config-vector $\langle C, \mathcal{V}(e), \mathcal{L}(e)\rangle$ for index i. The config-vector cv may be returned afterward, for instance, when one tries to complete $\langle C \cup \{f\}, \mathcal{V}(e), \mathcal{L}(e)\rangle$ for another index j by calling ConfigVectorSet_i$(F, j, C \cup \{f\}, \mathcal{V}(e), \mathcal{L}(e))$.

Lemma 5.3.6. *If F is finite, then function* ConfigVectorSet_i(F, i, C, v, a) *in Algorithm 5.8 terminates.*

Proof. The recursive call of ConfigVectorSet_i can not be infinite. Suppose the opposite that means there exists an infinite sequence of configurations $C_1, C_2, \ldots$ where $C_1 = C$ and ConfigVectorSet_i(F, i, C_j, v, a) calls to ConfigVectorSet_i(F, i, C_{j+1}, v, a) for all $j \geq 1$. We have not only $C_j \subset C_{j+1}$ due to line 14 but also $(\widehat{\gtrdot}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C_j)) \supset (\widehat{\gtrdot}(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C_{j+1}))$ for all $j \geq 1$. The infiniteness of decreasing sequence $\widehat{\gtrdot}_i(v\downarrow_i) \setminus \widehat{\mathcal{V}}(C_j)$ w.r.t. inclusion order contradicts to the finiteness of predecessor set of $v\downarrow_i$ in $\widehat{\mathcal{E}}_i$ by definition of event structures (see Definition 3.3.39 on page 54). Moreover, since F is finite too, for every call of ConfigVectorSet_i, the set X obtained in line 8 is finite. As a consequence, the loop at lines 15-17 is finite. Therefore, the function ConfigVectorSet_i terminates. □

Remark: One can consider the unfolding algorithm in [ER99] as the one for synchronized product of labeled event trees, and the unfolding algorithm in [McM95a] as the one for synchronized product of labeled event structures modeling simple Petri net's place. In both case, one still has some function like ours ConfigVectorSet_i. It is much simpler though because every component event has at most one predecessor. As a consequence, there is no need to use the recursion shown in Algorithm 5.8.

5.3.2 Function ConfigVectorSet

Algorithm 5.9 represents the function ConfigVectorSet(F, C, v, a) which takes a downward-closed set, w.r.t. the causality, of events F and a config-vector $\langle C, v, a\rangle$ as parameters. The config-vector $\langle C, v, a\rangle$ must (satisfy the invariant that it is) partially complete for all index $i \in I(v)$. The function ConfigVectorSet(F, C, v, a) then computes and returns all complete config-vectors $\langle D, w, a\rangle$ such that

- $w\downarrow_i = v\downarrow_i$ for all $i \in I(v)$, and
- D is a subset of F, i.e. $D \subseteq F$.

By Definition 5.3.2, it follows from the partial completeness of config-vectors $\langle D, w, a\rangle$ that $w\downarrow_i$ is an extension event of $D\downarrow_i$ for all $i \in I(a)$. As a consequence, direct predecessors of $w\downarrow_i$ are included in $D\downarrow_i \subseteq F\downarrow_i$. Therefore, aiming at computing such complete config-vectors in a prefix $\widehat{\mathcal{E}}_{\mathcal{SP}}$ of the synchronized product, one requires that $\widehat{E}_i$ contains all events whose predecessors are in $F\downarrow_i$ for all i. Formally, for all i, $e_i \in \widehat{E}_i$ if $>_i(e_i) = \widehat{\gtrdot}_i(e_i) \subseteq F\downarrow_i$. The following is straightforward.

Lemma 5.3.7. *Let $\langle C, v, a\rangle$ be a complete config-vector where C is a subset of a downward closed set F w.r.t. the causality. If $\widehat{E}_i \supseteq \{e_i \in E_i \,/\, >_i(e_i) \subseteq F\downarrow_i\}$ for all i, then $v \in \otimes_\varepsilon(\widehat{E}_1, \widehat{E}_2, \ldots, \widehat{E}_n)$.*

In Algorithm 5.9, in the base case where $I(v) = I(a)$ (line 3), it follows from the constraint on parameters that the config-vector $\langle C, v, a\rangle$ is already complete. The function simply returns the singleton $\{\langle C, v, a\rangle\}$ (line 4). In the general case (lines 5-18), by definition, $I(v)$ must be a subset of $I(a)$. The algorithm takes any index i in the different set $(I(a) \setminus I(v))$ (line 6) and tries to partially complete the config-vector $\langle C, v, a\rangle$ for the component i. Then, it tries to assign some component event e'_i in $\widehat{\mathcal{E}}_i$ to $v{\downarrow_i}$ (line 11). By definition of config-vectors, $v{\downarrow_i}$ should be labeled by $a{\downarrow_i}$. The set X_i obtained at line 7 thus represents the set of such component events e'_i. Notice that, for every component event $e'_i \in E_i$, if either $e'_i \in \widehat{\mathcal{V}}(C){\downarrow_i}$ or e'_i is in conflict with some event in $\widehat{\mathcal{V}}(C){\downarrow_i}$, exploiting such an event e'_i does not give rise to any complete config-vector from C. Therefore, the restriction of X_i line 9, in the one hand, is an algorithmic amelioration, and in the other hand, is in order to guarantee the invariant that one always works with config-vectors.

The loop at lines 10-13 searches all partially complete config-vectors $\langle C', v', a\rangle$ for the index i by calling $\mathsf{ConfigVectorSet\_i}(F, i, C, v', a)$. Due to assignments at lines 8 and 11, vectors v and v' are different only on index i. More precisely, we have $v{\downarrow_i} = \varepsilon$ while $v'{\downarrow_i} = e'_i$ for some $e'_i \in X_i$. All found config-vectors $\langle C', v', a\rangle$ are stocked in the set $\mathsf{CVS_i}$. Notice that if X_i is empty, the algorithm skips this loop, and $\mathsf{CVS_i}$ is thus empty. As a consequence, in this case, the algorithm skips also the loop at lines 15-17 and return the empty set CVS (line 20). Otherwise, i.e. $X_i \neq \emptyset$, every config-vector $\langle C', v', a\rangle$ is partially complete for all index in the set $I(v') = I(v) \cup \{i\}$. As a consequence, $\langle C', v', a\rangle$ may be used as parameter for the function $\mathsf{ConfigVectorSet}$ itself (line 16). Due to the loop at lines 15-17, the return set CVS (line 18) hopefully contains complete config-vectors $\langle D, w, a\rangle$ where $D \supseteq C$ and $I(w) = I(a) \supset I(v)$.

Algorithm 5.9: Function $\mathsf{ConfigVectorSet}$

```
1  function ConfigVectorSet(F, C, v, a)
2  begin
3      if I(v) = I(a) then
4          return {⟨C, v, a⟩}
5      else
6          take an index i in (I(a) \ I(v))
7          X_i := {e'_i ∈ (Ê_i \ V̂(C)↓_i) / L̂_i(e'_i) = a↓_i  and {e'_i} #̂^s_i V(C)↓_i}
8          v' := v
9          CVS_i := ∅
10         for each e'_i ∈ X_i do
11             v'↓_i := e'_i
12             CVS_i := CVS_i ∪ ConfigVectorSet_i(F, i, C, v', a)
13         end for
14         CVS := ∅
15         for each ⟨C', v', a⟩ ∈ CVS_i do
16             CVS := CVS ∪ ConfigVectorSet(F, C', v', a)
17         end for
18         return CVS
19     end if
20 end function
```

Lemma 5.3.8. *Let F be any downward-closed set, w.r.t. the causality. Let $\langle C, v, a\rangle$ be any config-vector such that:*

1. *C is a subset of F, i.e. $C \subseteq F$,*
2. *$\langle C, v, a\rangle$ is partially complete for every index $i \in I(v)$, and*
3. *for every $e \in \mathsf{Max}_{\leq}(C)$, there exists an index $i \in I(v)$ satisfying that $\langle C \setminus \{e\}, v, a\rangle$ is not partially complete for index i.*

If $\widehat{E}_i \supseteq \{e_i \in E_i \,/\, >_i(e_i) \subseteq F{\downarrow_i}\}$ for all i, then the return value of $\mathsf{ConfigVectorSet}(F, C, v, a)$ is the set of complete config-vectors $\langle D, w, a\rangle$ such that $C \subseteq D \subseteq F$ and $w{\downarrow_i} = v{\downarrow_i}$ for all $i \in I(v)$.

Proof. Let CVS denote the return value of $\mathsf{ConfigVectorSet}(F, C, v, a)$ and CS the set of complete config-vectors $\langle D, w, a\rangle$ satisfying $C \subseteq D \subseteq F$ and $w{\downarrow_i} = v{\downarrow_i}$ for all $i \in I(v)$. We will prove by induction on the finite size k of the set $(I(a) \setminus I(v))$, i.e. $k = |I(a) \setminus I(v)|$, that $\mathsf{CVS} = CS$.

- We first prove that $\mathsf{CVS} \subseteq CS$: In the base case, i.e. $I(v) = I(a)$, due to the assumption that $\langle C, v, a\rangle$ is partially complete for all indices $i \in I(v)$, $\langle C, v, a\rangle$ is thus complete by Definition 5.3.3. And the return set $\mathsf{CVS} = \{\langle C, v, a\rangle\}$ is of course a subset of CVS. In the general case, by definition, $\langle C, v', a\rangle$ (line 12) is a config-vector for all event e'_i chosen in the set X_i obtained in line 7. Thanks to Lemma 5.3.5, the set $\mathsf{CVS_i}$ obtained after the loop at lines 10-13 in Algorithm 5.9 contains config-vectors $\langle C', v', a'\rangle$ which satisfies $C \subseteq C' \subseteq F$, and at the same time, is partially complete for not only for the indices in $I(v)$ but also for the index i obtained in line 6. Moreover, let e be any maximal event in C' w.r.t. the causality, i.e. $e \in \mathsf{Max}_{\leq}(C') = \mathsf{Max}_{\leqslant}(C')$. If $e \in (C' \setminus C)$, then $\langle C' \setminus \{e\}, v', a\rangle$ is not partially complete for index i. Because, suppose the opposite, $\langle C', v', a\rangle$ is returned by calling $\mathsf{ConfigVectorSet\_i}(F, i, C, v', a)$ but C' is not the minimal configuration of the set $\{C'' \in \mathcal{C}_{\mathcal{E}_{\mathcal{SP}}} \,/\, C \subseteq C'' \subseteq F \text{ and } \widehat{\mathcal{V}}(C''){\downarrow_i} \vdash_i v'{\downarrow_i}\}$ due to the existence of the configuration $(C' \setminus \{e\}) \subseteq C'$. It contradicts to Lemme Lemma 5.3.5. Therefore, the config-vector $\langle C', v, a\rangle$ satisfy thus the three property stated by this Lemma as the condition of $\mathsf{ConfigVectorSet}$'s input. So that, when calling $\mathsf{ConfigVectorSet}(F, C', v', a)$ in line 16, since $(I(a) \setminus I(v')) \subset (I(a) \setminus I(v))$, by induction hypothesis, its return value is a subset of CS. And so does the final return value of $\mathsf{ConfigVectorSet}(F, C, v, a)$ (line 18).

- We now prove that $CS \subseteq \mathsf{CVS}$: Thanks to Lemma 5.3.7, for every complete config-vector $\langle D, w, a\rangle$, since $D \subseteq F$, we have $w \in \otimes_{\varepsilon}(\widehat{E}_1, \widehat{E}_2, \ldots, \widehat{E}_n)$. In the base case, i.e. $I(v) = I(a)$, one has $\mathsf{CVS} = \{\langle C, v, a\rangle\}$. Suppose that there exists a complete config-vector $\langle D, w, a\rangle \in CS$ which is not included in CVS. By definition, one obtains $C \subset D \subseteq F$ and $v = w$. Let e be any event in $\mathsf{Max}_{\leqslant}(D \setminus C) \subseteq \mathsf{Max}_{\leqslant}(D)$. Its follows from the conflict-freeness of configuration D that, for all i, $\widehat{\mathcal{V}}(e){\downarrow_i} \notin \widehat{\mathcal{V}}(C){\downarrow_i}$. Moreover, since $\langle C, v, a\rangle$ is partially complete for all indices $i \in I(v) = I(a)$, we have $\widehat{\mathcal{V}}(C){\downarrow_i} \vdash_i v{\downarrow_i}$, and consequently, $\widehat{\gtrdot}_i(v{\downarrow_i}) \subseteq \widehat{\mathcal{V}}(C){\downarrow_i}$ for all $i \in I(v)$. Therefore, the third property in Definition 5.3.3 does not hold for $\langle D, v, a\rangle$ because, for all $i \in I(v)$, $\widehat{\mathcal{V}}(e) \downarrow_i \; \overline{\widehat{\lessdot}_i} \, v \downarrow_i$. It contradicts to the completeness of $\langle D, v, a\rangle$. As a consequence, $CS = \{\langle C, v, a\rangle\}$ is thus a subset of CVS.

  In the general case, i.e. $I(v) \subset I(a)$, let $\{D, w, a\}$ be any complete config-vector in CS. Let i be the value obtained at line 6 in Algorithm 5.9, and $e'_i = w \downarrow_i$.

One must have $C \subset D$ because if otherwise, $\langle D, w, a\rangle = \langle C, w, a\rangle$ can not be complete. Since $\langle D, w, a\rangle$ is a config-vector and $C \subseteq D$, one has then $w\downarrow_i \notin \widehat{\mathcal{V}}(C)\downarrow_i$ and $\{w\downarrow_i\} \overline{\#}_i^s \widehat{\mathcal{V}}(C)\downarrow_i$ by Definition 3.3.39 on page 54. It follows from $\mathcal{L}_i(w\downarrow_i) = a\downarrow_i$ (by Definition 5.3.2) that $w\downarrow_i$ must be in the set X_i obtained at line 7. As a consequence, in the loop at lines 10-13, there is a step where $v'\downarrow_i = w\downarrow_i$ and $v'\downarrow_j = v\downarrow_j$ for all $j \neq i$. Thanks to Lemma 5.3.5, when calling $\mathsf{ConfigVectorSet\_i}(F, i, C, v', a)$, it must return some config-vectors. Thanks to Lemma 5.3.5, among such return config-vectors in CVS_i, there exists $\langle C', v', a\rangle$ such that $C' \subseteq D$. Because configuration D satisfies $\widehat{\mathcal{V}}(D)\downarrow_i \vdash_i v'\downarrow_i$. By the same reasoning as in the previous case, one can verify that $\langle C', v', a\rangle$ satisfies the input's variant of the function ConfigVectorSet like $\langle C, v, a\rangle$. So that by the induction hypothesis, the complete config-vector $\langle D, w, a\rangle$ must be found in the final set CVS due to some call $\mathsf{ConfigVectorSet}(F, C', v', a)$ in line 16.

One can finally conclude that CVS is equal to CS. □

The third property on configuration C as input of ConfigVectorSet is important. As seen in the proof above, it corresponds more or less to the third property in the definition of complete config-vectors (see Definition 5.3.3 on page 110). Without such a property on input configurations C, calling $\mathsf{ConfigVectorSet}(F, C, v, a)$ may return some config-vector $\langle D, w, a\rangle$ which is partially complete for all indices $i \in I(w) = I(a)$. However, $\langle D, w, a\rangle$ is not complete, and as a consequence of Lemma 5.3.4 on page 110, it corresponds to no event in the synchronized product $\mathcal{E}_{\mathcal{SP}}$. Aiming at constructing prefixes of $\mathcal{E}_{\mathcal{SP}}$, the computation of config-vector $\langle D, w, a\rangle$ is useless.

Lemma 5.3.9. *If $\widehat{E}$ as well as $\widehat{E}_1, \widehat{E}_2, \ldots \widehat{E}_n$ is finite, then the function* ConfigVectorSet *in Algorithm 5.9 terminates.*

Proof. Let (F, C, v, a) denote some input of ConfigVectorSet. We will prove this Lemma by induction on the size k of $(I(a) \setminus I(v))$ since k can not exceeds the number of components, and is thus finite. When $k = 0$, i.e. $I(v) = I(a)$, one falls into the base case of the function ConfigVectorSet's recursion (lines 3-4). The function ConfigVectorSet just terminates. Suppose that ConfigVectorSet terminates for all k smaller than some number $m > 0$. We will prove that it also terminates for $k = m$. For every value of i obtained at line 6, the set X_i is finite because $\widehat{E}_i$ is finite. Thanks to Lemma 5.3.6, $\mathsf{ConfigVectorSet\_i}(F, C', v', a, i)$ terminates because $F \subseteq \widehat{E}$ is finite, and at the same time, thanks to Lemma 5.3.5, its return set has a cardinal smaller than or equal to the one of $\{C'' \in \mathcal{C}_{\widehat{\mathcal{E}}_{\mathcal{SP}}} / C'' \supseteq C\}$. As a consequence, the loop at lines 10-13 terminates and the set $\mathsf{CVS_i}$ obtained afterward is finite. Consider now the loop at lines 15-17. Because $|I(a) \setminus I(v')| = |I(a) \setminus I(v) \setminus \{i\}| = m - 1$, by induction hypothesis, the call of $\mathsf{ConfigVectorSet}(C', v', a)$ at line 16 terminates. This loop with finite bound $|\mathsf{CVS_i}|$ should terminates. As a consequence, the function ConfigVectorSet terminates. □

5.3.3 Functions $\mathsf{Init}_{\mathcal{SP}}$ and $\mathsf{Extend}_{\mathcal{SP}}$

In this subsection, we assume that one has already n functions Init_i as well as n functions Extend_i, $i \in \{1, 2, \ldots, n\}$, for unfolding n component labeled event structures. In addition to the prefix $\widehat{\mathcal{E}}_{\mathcal{SP}}$ of $\mathcal{E}_{\mathcal{SP}}$ which is being constructed, we always have n prefixes of the components. The function $\mathsf{Init}_{\mathcal{SP}}$ as well as $\mathsf{Extend}_{\mathcal{SP}}$ should use these $2n$ functions for expanding components prefixes if necessary. Moreover, component possible extensions PE_i, $i \in \{1, 2, \ldots, n\}$ can be accessed from $\mathsf{Init}_{\mathcal{SP}}$ and $\mathsf{Extend}_{\mathcal{SP}}$.

As stated by Lemma 5.3.4, in a synchronized product of labeled event structures, complete config-vectors correspond to events. Formally, we say that $\langle C, v, a\rangle$ corresponds to event e if $C = {>}(e), v = \mathcal{V}(e)$, and $a = \mathcal{L}(e)$. In addition, due to the maximality and no-duplication property in Definition 3.3.44 on page 57, this correspondence is a bijection.

By definition, a minimal event e, w.r.t. the causality, in the synchronized product is a synchronization of minimal events in component labeled event structures. Formally, $e \in \mathsf{Min}_{\leq}(\mathcal{E}_{\mathcal{SP}})$ iff ${>}(e) = \emptyset$ and $\mathcal{V}(e) \in \otimes_{\varepsilon}(\mathsf{Min}_{\leq_1}(E_1), \ldots, \mathsf{Min}_{\leq_1}(E_1))$. In order to building the prefix $\mathcal{E}_{\mathcal{SP}}|_{\mathsf{Min}_{\leq}(E)}$, function $\mathsf{Init}_{\mathcal{SP}}$ intuitively computes complete config-vectors $\langle \emptyset, v, a\rangle$ where $v \in \otimes_{\varepsilon}(\mathsf{Min}_{\leq_1}(E_1), \ldots, \mathsf{Min}_{\leq_n}(E_1))$ and $a = \langle \mathcal{L}_1(v{\downarrow_1}), \ldots, \mathcal{L}_n(v{\downarrow_n})\rangle$ is included in synchronization constraint Σ.

Algorithm 5.10: Function $\mathsf{Init}_{\mathcal{SP}}$ for synchronized products

```
1  function Init_SP()
2  constant v_ε = ⟨ε, ε, ..., ε⟩
3  begin
4      Ê := ∅; ≼̂ := ∅; V̂ := ∅
5      for i := 1 to n do
6          (Ê_i, PE_i) := Init_i()
7      end for
8      CVS := ∅
9      for each a in Σ do
10         CVS := CVS ∪ ConfigVectorSet(∅, ∅, v_ε, a)
11     end for
12     for each ⟨C, v, a⟩ ∈ CVS do
13         e' := Create(Ê, PE, ∅, a)
14         V̂ := V̂ ∪ {⟨e', v⟩}
15     end for
16     return (Ê, Ê)
17 end function
```

Algorithm 5.10 represents the function $\mathsf{Init}_{\mathcal{SP}}$. The synchronized prefix is initialized without events (line 4). Component prefixes are also initialized by calling the corresponding functions Init_i for all $i \in \{1, 2, \ldots, n\}$ due to the loop at lines 5-7. As seen in Section 5.2, after this loop, each component prefix $\widehat{\mathcal{E}}_i$, $i \in \{1, 2, \ldots, n\}$, contains only minimal events w.r.t. its causality $\widehat{\leq}_i$. Due to the loop at lines 9-11, the algorithm thus computes all complete config-vectors $\langle C, v, a\rangle$ by calling $\mathsf{ConfigVectorSet}(\emptyset, \emptyset, v_{\varepsilon}, a)$ for all actions a in the synchronization constraint Σ. The constant v_{ε} declared in line 2 is just for initializing the third parameter when calling ConfigVectorSet. Since the first parameter F of ConfigVectorSet is the empty set, C is empty too. Each complete config-vector $\langle \emptyset, v, a\rangle$ in CVS intuitively corresponds to an minimal event e' w.r.t. the causality in the global labeled event structure because ${>}(e') = \emptyset$. Function $\mathsf{Init}_{\mathcal{SP}}$ finally creates events according to config-vectors in CVS and accordingly updates function $\widehat{\mathcal{V}}$. This is done by the loop at lines 12-15. Recall that the conflict relation $\widehat{\#}$ and the labeling function $\widehat{\mathcal{L}}$ may be computed from the ones of component prefixes, i.e. $\widehat{\#}_i$ and $\widehat{\mathcal{L}}_i$, the predecessor relation $\widehat{\prec}$ and the vector $\widehat{\mathcal{V}}$. In Algorithm 5.10, only instructions for $\widehat{\prec}$ and $\widehat{\mathcal{V}}$ are shown. The following is straightforward.

Lemma 5.3.10. *If* $\mathsf{Init}_i()$ *is correct that means its return value* $(\widehat{\mathcal{E}}_i, \mathsf{PE}_i)$ *is correct w.r.t.* $\mathcal{E}_i$ *and* $\widehat{\mathcal{E}}_i = \mathsf{Init}_i() = \mathcal{E}_i|_{\mathsf{Min}_{\leq_i}(E_i)}$*, then the return value* $(\widehat{\mathcal{E}}, \widehat{E})$ *is correct w.r.t. the synchronized product* $\mathcal{E}_{\mathcal{SP}}$*, moreover,* $\widehat{\mathcal{E}} = \mathcal{E}|_{\mathsf{Min}_{\leq}(E)}$.

Now, we can go into details of the function $\mathsf{Extend}_{\mathcal{SP}}$. Recall the principal mechanism of Extend: one avoids adding two times a same event, and at the same time, does not omit any possible event. In order to do that:

- One adds only successors e' of event e where e is the parameter of Extend. By Definition 3.3.39, the complete config-vector $\langle C, v, \mathcal{L}(e')\rangle$ corresponding to e' must satisfies that $\mathcal{V}(e)\downarrow_i \lessdot_i v\downarrow_i$ for some index i, and at the same time, C contains e.
- One adds only and all those events e' whose predecessors have been extended, i.e. $\gtrdot(e') \subseteq (\widehat{E} \setminus \mathsf{PE})$ where $\widehat{E}$ is the event set of the actual prefix, and $\mathsf{PE} \subseteq \widehat{E}$ is the set of events that have not been extended.

Let us explain the instructions of $\mathsf{Extend}_{\mathcal{SP}}$ given in Algorithm 5.11. As usual, when extending event e, the algorithm first removes e from the possible extension PE (line 4). Then, it extends the component events corresponding to e by the loop at lines 5-9 if necessary. Let i be an index in $I(\widehat{\mathcal{V}}(e))$, when the test $\widehat{\mathcal{V}}(e)\downarrow_i \in \mathsf{PE}_i$ (line 6) is false, it may be due to extending another event e' before e where e' concerns the same component event as e, i.e. $\widehat{\mathcal{V}}(e)\downarrow_i = \widehat{\mathcal{V}}(e')\downarrow_i$. In this case, the algorithm does nothing so that Extend_i is called with argument $\mathcal{V}(e)\downarrow_i$ at most one time. Otherwise, i.e. $\mathcal{V}(e)\downarrow_i \in \mathsf{PE}_i$, it means that component event $\mathcal{V}(e)\downarrow_i$ in $\mathcal{E}_i$ has not been extended. One must extend it by calling $\mathsf{Extend}_i(\widehat{\mathcal{E}}_i, \mathsf{PE}_i, \mathcal{V}(e)\downarrow_i)$ (line 7). After this instruction, $\mathcal{V}(e)\downarrow_i$ is no more in PE_i.

Algorithm 5.11: Function $\mathsf{Extend}_{\mathcal{SP}}$ for synchronized products

```
1  function Extend_SP(Ê, PE, e)
2  constant v_ε = ⟨ε, ε, ..., ε⟩
3  begin
4      PE := PE \ {e}
5      for each i ∈ I(V̂(e)) do
6          if V̂(e)↓_i ∈ PE_i then
7              (Ê_i, PE_i) := Extend_i(Ê_i, PE_i, V̂(e)↓_i)
8          end if
9      end for
10     CVS_i := ∅
11     for each i ∈ I(V̂(e)) do
12         v' := v_ε
13         for each v_i ∈ ⩽̂_i(V̂(e)↓_i) do
14             v'↓_i := v_i
15             for each a ∈ {a ∈ Σ / a↓_i = L̂_i(v_i)} do
16                 CVS_i := CVS_i ∪ ConfigVectorSet_i((Ê \ PE), i, ⩾̂(e), v', a)
17             end for
18         end
19     end for
20     CVS := ∅
21     for each ⟨C', v', a⟩ ∈ CVS_i do
22         CVS := CVS ∪ ConfigVectorSet((Ê \ PE), C', v', a)
23     end for
```

```
24      for each ⟨C, v, a⟩ ∈ CVS do
25          e' := Create(Ê, PE, Max_≲(C), a)
26          V̂ := V̂ ∪ {⟨e', v⟩}
27      end for
28      return (Ê, PE)
29  end function
```

Remark: In our unfolding algorithm for a synchronized product of labeled event structures, we only use Algorithm 5.1 for the global product, and not for its components. The function $\mathsf{Extend}_{\mathcal{SP}}$ of the global product, on the one hand, is responsible for constructing prefixes $\widehat{\mathcal{E}}_{\mathcal{SP}}$ of the synchronized product, and on the other hand, takes control of how to develop prefixes $\widehat{\mathcal{E}}_i$, $i \in \{1, 2, \ldots, n\}$, of the components. The choice of extending component events is no more random as seen at line 4 in Algorithm 5.1. In other words, a event e_i in some component $\widehat{\mathcal{E}}_i$ is extended by calling Extend_i only when a global event e in the synchronized product which concerns e_i, i.e. $\mathcal{V}(e)\downarrow_i = e_i$, is extended.

The computing process of complete config-vectors CVS which correspond to direct successors e' of e is started by finding partially complete config-vectors $\langle C', v', a\rangle$ for some index $i \in I(\mathcal{V}(e))$ (the nested loops at lines 11-19).

- Since e' is a direct successor of e, by the third property of Definition 3.3.39, $\mathcal{V}(e')\downarrow_i$ must be a direct successor of $\mathcal{V}(e)\downarrow_i$ in some component labeled event structures $\mathcal{E}_i$. Hence, one restricts index i on the set $I(\mathcal{V}(e))$ (line 11).
- The component event $\mathcal{V}(e)\downarrow_i$ may have many direct successors in $\mathcal{E}_i$. For such a direct successor v_i (line 13), one initializes v' so that it is different from the constant vector v_ε (line 2) only on the index i due to instructions at lines 12 and 14. All partially complete config-vectors obtained afterward must be based on the same vector v'.
- One groups partially complete config-vectors $\langle C', v', a\rangle$ into different sets which based on label $a \in \Sigma$. By Definition 5.3.2, one has, of course, $a\downarrow_i = \mathcal{L}_i(\mathcal{V}(v')\downarrow_i = \mathcal{L}_i(v_i)$ where v_i is an event in $\mathcal{E}_i$ obtained previously.
- The calling $\mathsf{ConfigVectorSet\_i}$ with the configuration parameter $\geq(e)$ (line 16) guarantees somehow that return config-vectors $\langle C', v', a\rangle$ satisfies $C' \supseteq (>(e))$. As a consequence, new event e' corresponding to $\langle C', v', a\rangle$ satisfies $(>(e')) \supseteq (\geq(e))$ and is thus a direct successor of e.

All partially complete config-vectors in $\mathsf{CVS_i}$ are thus used for computing complete config-vectors by the loop at lines 21-23. One simply call the function ConfigVectorSet (line 33). Notice that obtained config-vectors in CVS may be different on its vectors or its corresponding actions. However, the configuration of such config-vectors always contains e, and is thus a superset of $\geq(e)$.

Finally, as in the function Init, for every complete config-vector $\langle C, v, a\rangle$ in CVS, a new event e' is created and inserted into the global prefix by the loop at lines 24-27. Since only events in $\mathsf{Max}_{\lesssim}(C)$ are direct predecessor of e'. $\mathsf{Max}_{\lesssim}(C)$ is passed as the parameter value of ConfigVectorSet (line 25). The function $\widehat{\mathcal{V}}$ is also modify for adapting to new events (line 26). The set of new events is returned as usual (line 28). However, the set CVS as well as $\mathsf{CVS_i}$ may be empty. In this case, the loop at lines 21-24 as well as the one at lines 24-27 is algorithmically skipped. And as a consequence, no new successors of e is created and returned in the end of the call $\mathsf{Extend}(e)$.

Lemma 5.3.11. *If* Extend_i *is correct w.r.t.* $\mathcal{E}_i$ *for all* $i \in \{1, 2, \ldots, n\}$ *then the function*

$\mathsf{Extend}_{\mathcal{SP}}$ *in Algorithm 5.11 is correct w.r.t. the synchronized product* $\mathcal{E}_{\mathcal{SP}} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ *of these* n *labeled event structures* $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ *w.r.t.* Σ.

Proof. Let us denote by $\widehat{\mathcal{E}} = (\widehat{E}, \widehat{\leq}, \widehat{\#}, \widehat{\mathcal{L}}, \mathcal{M}')$, $\widehat{\mathcal{E}}' = (\widehat{E}', \widehat{\leq}', \widehat{\#}', \widehat{\mathcal{L}}', \widehat{\mathcal{M}'})$ respectively the values of structure variables of the synchronized products just before and after calling $\mathsf{Extend}_{\mathcal{SP}}(e)$ for some $e \in (\{\varepsilon\} \cup E')$; and by the same manner, $\mathcal{E}'_i = (E'_i, \leq'_i, \#'_i, \mathcal{L}'_i, \mathcal{M}'_i)$, $\mathcal{E}''_i = (E''_i, \leq''_i, \#''_i, \mathcal{L}''_i, \mathcal{M}''_i)$ for the values of structure variables of every component $i \in \{1, 2, \ldots, n\}$.

Let S be the set of successors e' of e in $\mathcal{E}_{\mathcal{SP}}$ whose predecessors has been already extended, i.e. $S = \{e' \in E \,/\, e \lessdot e' \text{ and } {>}(e') \subseteq (\widehat{E} \setminus \mathsf{PE}')\}$. Here $\mathsf{PE}' = (\mathsf{PE} \setminus \{e\}) \cup (\widehat{E}' \setminus \widehat{E})$ due to line 4 and the calls of Create in line 25. We will prove that there is a bijection between S and the final config-vectors set CVS in the function $\mathsf{Extend}_{\mathcal{SP}}$ (*).

Due to the loop at lines 5-9 that tries to extend component events each time one extends a global event, one can easily prove by induction that for all i, $(\widehat{E}_i \setminus \mathsf{PE}_i) \supseteq (\widehat{E} \setminus \mathsf{PE}){\downarrow_i}$ just before instructions for computing CVS in the function Extend (line 10). As a consequence, for every event $e' \in E$, if $({>}(e')) \subseteq (E' \setminus \mathsf{PE})$ then $({>_i}(\mathcal{V}(e'){\downarrow_i})) \subseteq ({>}(e'){\downarrow_i} \subseteq (E' \setminus \mathsf{PE}){\downarrow_i} = (E'_i \setminus \mathsf{PE}_i)$. Moreover, due to the correctness of Extend_i, $({>_i}(\mathcal{V}(e'){\downarrow_i})) \subseteq (E'_i \setminus \mathsf{PE}_i)$ implies that event $\mathcal{V}(e'){\downarrow_i}$ must be already in the event set E'_i of component prefix $\mathcal{E}'_i$. Now, for each successor $e' \in S$, let us associate it to the complete config-vector $\langle {>}(e'), \mathcal{V}(e'), \mathcal{L}(e')\rangle$. We have then $({>}(e')) \subseteq (\widehat{E} \setminus \mathsf{PE}')$ and $\mathcal{V}(e') \in \otimes_\varepsilon(E'_1, E'_2, \ldots, E'_n)$. In other words, event sets $E'_1, E'_2, \ldots, E'_n$ are sufficient for computing the complete config-vectors $\langle {>}(e'), \mathcal{V}(e'), \mathcal{L}(e')\rangle$.

Consider the nested loops at lines 11-19. Since e' is a successor of e, by definition of synchronize products of event structures (see Definition 3.3.39), there exists an index i such that $\mathcal{V}(e){\downarrow_i} \lessdot_i \mathcal{V}(e'){\downarrow_i}$. Moreover, the action $a = \mathcal{L}(e') \in \Sigma$ satisfies that $a{\downarrow_i} = \mathcal{L}_i(\mathcal{V}(e'){\downarrow_i})$. Let v' is the vector satisfying $v'{\downarrow_i} = \mathcal{V}(e'){\downarrow_i}$ and $v'{\downarrow_i} = \varepsilon$ otherwise. Thanks to Lemma 5.3.5, after this nested loop, $\mathsf{CVS_i}$ must contain some partially complete config-vector $\langle C', v', a\rangle$ for the index i due to the call $\mathsf{ConfigVectorSet\_i}(\widehat{E} \setminus (\mathsf{PE} \setminus e), {\geq}(e), v', a)$ in line 16. Moreover, $({\geq}(e)) \subseteq C'$ and $C' \subseteq ({>}(e'))$ due to its minimality w.r.t. the inclusion order. Therefore, it follows from Lemma 5.3.8 that CVS contains $\langle {>}(e'), \mathcal{V}(e'), \mathcal{L}(e')\rangle$ (1) because of the call ConfigVectorSet $(\widehat{E} \setminus \mathsf{PE}, C', v', a)$ in line 33.

Reversely, let $\langle C, v, a\rangle$ be any complete config-vector in CVS in the end of the loop at lines 21-23. Once again, C is a supset of another configuration C' which contains e; and v is based on another vector v' which satisfying that $v'{\downarrow_i} = \mathcal{V}(e){\downarrow_i}$ for some index i. Thanks to Lemma 5.3.4, there exists an event $e' \in E$ such that $\mathcal{V}(e') = v$ and $C = {>}(e')$. Notice that $C \subseteq (E' \setminus (\mathsf{PE} \setminus \{e\}))$ because the first parameter F when calling ConfigVectorSet as well as ConfigVectorSet_i is always equal to $\widehat{E} \setminus (\mathsf{PE} \setminus \{e\})$. Moreover, since e is a maximal event w.r.t. the causality in $\widehat{E}$ by Lemma 5.1.2 on page 89, e is also a maximal one in $C = {>}(e')$. The event e is just a direct predecessor of e' and as a consequence, e' must be in S (2). It follows from (1) and (2) that (*) is obvious. The bijection, denoted by $\mathcal{B}$, may be defined by $\mathcal{B}(e') = \langle {>}(e'), \mathcal{V}(e'), \mathcal{L}(e')\rangle$.

The for loop at lines 24-27 simply adds events in S to the $\widehat{E}$-prefix of the synchronized product. Because e is maximal event in $\widehat{E}$, its successors do not exist in $\widehat{E}$. As a consequence, adding these successors guaranties that final $\widehat{\mathcal{E}}'$ is a prefix of $\mathcal{E}_{\mathcal{SP}}$. The unfolding invariant $I3$ in Definition 5.1.1 on page 89 is a direct consequence of (*) while invariants $I1, I2$ as well as conditions $C2, C3$ of the Extend's correctness are straightforward. The function $\mathsf{Extend}_{\mathcal{SP}}$ is thus correct w.r.t. $\mathcal{E}_{\mathcal{SP}}$. □

Lemma 5.3.12. *The function* $\mathsf{Extend}_{\mathcal{SP}}$ *terminates if* $\widehat{E}$ *as well as* $\widehat{E}_i$, $i \in \{1, 2, \ldots, n\}$,

is finite.

Proof. Thanks to Lemma 5.3.6 and Lemma 5.3.9, ConfigVectorSet_i and ConfigVectorSet terminates. The size of the set returned by ConfigVectorSet_i (Algorithm 5.8 on page 112) does not exceed the number of subsets of $\widehat{E}$, and is thus finite. On its turn, the function ConfigVectorSet (Algorithm 5.9 on page 115) has two finite loops: the first one calls to terminating function ConfigVectorSet_i and the second one which calls to ConfigVectorSet itself. Since the number n of components is finite, the depth of recursion is bounded by n. Then, the function $\mathsf{Extend}_{\mathcal{SP}}$ in Algorithm 5.11 terminates. □

In order to find new direct successors of an event in the actual prefix $\widehat{\mathcal{E}}$ of a synchronized product $\mathcal{E}_{\mathcal{SP}}$, i.e. $\widehat{\mathcal{E}} = \mathcal{E}_{\mathcal{SP}}|_{\widehat{E}}$, the functions ConfigVectorSet and ConfigVectorSet_i are the most important and slowest parts of unfolding algorithms. Although $\mathcal{E}_{\mathcal{SP}}$ is simply a synchronized product of Petri nets' places (or of counters), the complexity of ConfigVectorSet as well as $\mathsf{Extend}_{\mathcal{SP}}$ in the worst case is a NP-complete problem as stated in [ERV96, Hel99]. The question of how to efficiently compute such successors is still open. Some concrete ideas on Petri nets can be found in [Kho03].

In practice, one aims only at a finite prefix of the synchronized product $\mathcal{E}_{\mathcal{SP}}$ which is complete for some verification problem. As seen in the next section, the better cutting context is, the more compact prefixes one obtains. Because the complexity in time and in space of $\mathsf{Extend}_{\mathcal{SP}}$ depends on the size of these prefixes, a good choice of cutting context could reduces this complexity (see Chapter 6).

5.4 Truncating

Once we have a correct function Extend for constructing prefixes of a labeled event structure $\mathcal{E}$, we wish to modify the unfolding algorithm given in Algorithm 5.1 on page 89 to obtain some truncations of $\mathcal{E}$. A computed truncation, if it exists and is finite, will be used to verify the corresponding decidable problem (see Chapter 4).

As seen in previous sections, our unfolding algorithm as well as various functions Extend respect the idea of partial-order. Intuitively, one does not need to look at the whole set of configurations when computing new events and extending prefixes. In order to integrate truncation technique into the unfolding algorithm, we are particularly interested in local cutting contexts $(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$ (see Section 4.3.1). Moreover, recall that well-preorders $\preccurlyeq^{\mathcal{C}}$ over configurations are restricted to the ones defined in Section 4.2.3. Formally, that means $\preccurlyeq^{\mathcal{C}} = (\preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq)$ where $\trianglelefteq$ is an adequate order over configurations.

Remark: Without the risk of confusion, we simply write $e \trianglelefteq e'$ in the place of $(\geq(e)) \trianglelefteq (\geq(e'))$ for all events $e, e' \in E$. Therefore, $\trianglelefteq$ may be considered as an order over the event set E. And it is well-founded when $\mathcal{E}$ is finitely-branching.

Recall that, the truncation of $\mathcal{E}$ w.r.t. to a local cutting context $(\preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$ is the maximal subset of events that contains no outer one (*cf.* Definition 4.2.12 on page 74 and Lemma 4.3.2 on page 79). Algorithm 5.12 represents our truncating algorithm which aims at constructing the the prefix $\widehat{\mathcal{E}}$ of $\mathcal{E}$ based on the truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$, i.e. $\widehat{\mathcal{E}} = \mathcal{E}|_{\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})}$. While trying to keep the being constructed prefix $\widehat{\mathcal{E}}$ of $\mathcal{E}$ away from outer events, one simply does not extend cutoff events.

In Algorithm 5.12, we use a variable CE to stock cutoff events. This variable is, of course, initialized by the empty set (line 2). The algorithm starts with the prefix of $\mathcal{E}$ based on its minimal events w.r.t. the causality $\leq$ (line 3) as the same manner as in Algorithm 5.1. However, the loop for enlarging the actual prefix terminates if possibly

Algorithm 5.12: Truncating algorithm

```
1  begin
2      CE := ∅
3      (Ê, PE) := Init()
4      while (PE \ CE) ≠ ∅ do
5          take an event e in Min⊴(PE \ CE)
6          if isCutoff(e) then
7              CE := CE ∪ {e}
8          else
9              (Ê, PE) := Init(Ê, PE, e)
10         end if
11     end while
12 end
```

extensible events in PE are all cutoff ones (line 4). For each event e obtained at line 5, one must test whether e is cutoff event by calling the function isCutoff. If e is a cutoff event, one simply inserts it into the set CE (line 7). Notice that, in this case, e always belongs to the set PE. If e is not a cutoff event, one extends e by calling Extend by the same manner as in Algorithm 5.1.

If the principal loop terminates, in the end of truncating algorithm, we obtain an finite prefix of $\mathcal{E}$. In the result, CE is equal to PE and is the set of minimal cutoff events w.r.t. the causality $\leq$. Moreover, the event set of the final prefix is a superset of the truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^{l}_{\mathcal{E}})$. Let $\widehat{E}$ denote the event set of the actual prefix along an execution of the truncating algorithm, it is worth noticing some intuitive ideas in Algorithm 5.12:

- In order to prevent adding outer events as well as its successors afterward, cutoff events should not be removed from PE. Because calling $\mathsf{Extend}(\widehat{\mathcal{E}}, \mathsf{PE}, e)$ returns successors of e whose predecessors are all in $\widehat{E} \setminus \mathsf{PE}$ due to the correctness of Extend.
- CE is always a subset of both PE and $\mathsf{Min}_{\leq}(\widehat{E}^c)$ where $\widehat{E}^c$ is the set of cutoff event in the $\widehat{E}$-prefix of $\mathcal{E}$ w.r.t. $\preccurlyeq^{\mathcal{C}}$. They converge only when the loop terminates.
- The implementation of the function isCutoff is not far from the definition of cutoff events (see Definition 4.2.10 on page 74 and Definition 4.3.1 on page 79).

$$\mathsf{isCutoff}(e) \text{ returns } \begin{cases} \mathsf{true} & \text{if exists } e' \in (\widehat{E} \setminus \mathsf{CE}) : (\geq(e)) \prec^{\mathcal{C}} (\geq(e') \\ \mathsf{false} & \text{otherwise} \end{cases}$$

However, computation in isCutoff does not base on the whole labeled event structure $\mathcal{E}$ but only on one of its finite prefix, here is the $\widehat{E}$-prefix.

5.4.1 Algorithmic cutoff events

Consider an execution of Algorithm 5.12, let us simply call CE the set of *algorithmic cutoff events*. Recall that, as stated by Proposition 5.1.4 on page 90, one obtains an increasing sequence of prefixes of $\mathcal{E}$. By definition, every algorithmic cutoff event is a cutoff event in $\mathcal{E}$. However, there may be some cutoff event in $\mathcal{E}$ which is not detected as an algorithmic cutoff event, and is eventually extended. This fact is the cause of an final prefix, if algorithm terminates, which is bigger than the desired truncation.

Therefore, the choice of extending some minimal event e w.r.t. the adequate order $\trianglelefteq$ in $(\mathsf{PE} \setminus \mathsf{CE})$ (line 5) is very important. First, it reduces the risk of adding a cutoff event to the actual prefix without perceiving it as an algorithmic cutoff event and inserting it into CE. Second, for every prefixe obtained along the execution after extending e, $\mathsf{isCutoff}(e)$ always returns the same value.

Proposition 5.4.1. *Let us denote by* $\widehat{\mathcal{E}}_k = (\widehat{E}_k, \widehat{\leq}_k, \widehat{\#}_k, \widehat{\mathcal{L}}_k, \widehat{\mathcal{M}}_k)$ *and* PE_k *respectively the value of structure variables and* PE *after* k *steps,* $k = 0, 1, \ldots$, *of the principal loop in Algorithm 5.12 (lines 4-11). Let* e_k, $k = 1, 2, \ldots$, *the value of variable* e *chosen at the* k^{th} *step of this loop. Suppose that* Extend *is correct w.r.t.* $\mathcal{E}$. *Then* $\mathsf{isCutoff}(e_k)$ *(line 6) returns* true *iff* e_k *is a cutoff event in* $\widehat{\mathcal{E}}_n$ *for all* $n \geq (k-1)$.

Proof. Thanks to Proposition 5.1.4 on page 90, since Extend is correct w.r.t. $\mathcal{E}$, for every k, $\widehat{\mathcal{E}}_k$ is a prefix of $\mathcal{E}$. And moreover, $\widehat{E}_0 \subseteq \widehat{E}_1 \subseteq \ldots$. As a consequence, e_k is a cutoff event in $\widehat{E}_{k-1}$, i.e. $\mathsf{isCutoff}(e_k)$ returns true, implies that e_k is a cutoff event in $\mathcal{E}_n$ for all $n \geq (k-1)$. Now, suppose that $\mathsf{isCutoff}(e_k) = \mathsf{false}$ but e_k is a cutoff event in $\mathcal{E}_n$ for some $n \geq k$ (*). Without lost of generality, one can assume that k is the minimal number satisfying (*). There exists thus another event $e' \in \widehat{E}_n$ such that $(\geq(e_k)) \prec^{\mathcal{C}} (\geq(e'))$. Then, $e' \lhd e_k$. Since adequate order $\trianglelefteq$ refines the inclusion order $\subseteq$, we have that $e'' \lhd e' \lhd e_k$ for all $e'' \in (>(e'))$. Due to the choice of extending minimal event w.r.t. $\trianglelefteq$ at line 5 in Algorithm 5.12, e'' must be extended before e_k for all $e'' \in (>(e'))$. It follows from the correctness of Extend that $\widehat{E}_{k-1}$ contains e'. This contradicts to $\mathsf{isCutoff}(e_k) = \mathsf{false}$ and to (*). This proposition is thus proved. □

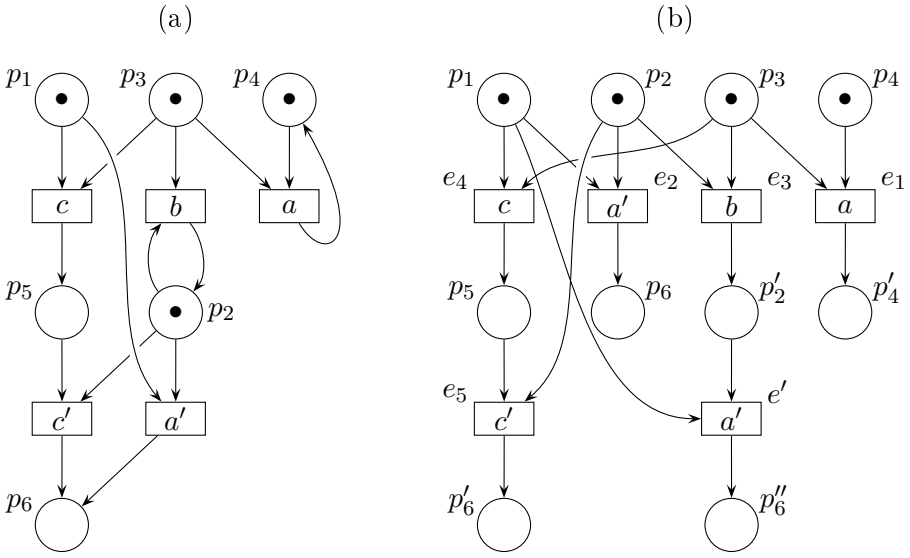


Figure 5.1: (a) An one-safe Petri net and (b) its corresponding labeled occurrence net.

Example 5.4.2. Figure 5.1.a represents an one-safe Petri net $(\mathcal{N}, m^i)$ which has 6 places, 5 transition and $m^i(p) = 1$ if $p \in \{p_1, p_2, p_3, p_4\}$ and $m^i(p) = 0$ otherwise. Its labeled occurrence net is also an one-safe Petri net $(\mathcal{N}', m'^i)$ and may be obtained by the well-known unfolding technique [McM95a]. These two Petri nets are obviously bisimilar. Figure 5.1.b illustrates $(\mathcal{N}', m'^i)$. By considering more or less only transitions of $(\mathcal{N}', m'^i)$, one intuitively obtained a equivalent labeled event structure $\mathcal{E}$ which is illustrated in Figure 5.2.a.

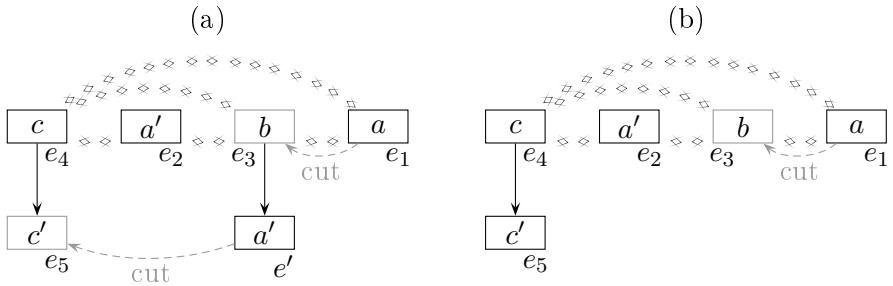


Figure 5.2: (a) The labeled event structure corresponding to the labeled occurrence net in Figure 5.1.b, and (b) the final prefix generated by Algorithm 5.12.

Among 6 events in $\mathcal{E}$, there are two couples of events whose local configurations are the same. Intuitively, both $\geq(e_1)$ and $\geq(e_3)$ lead to the marking where only places p_1, p_2, p_4 have a token; while both $\geq(e_5) = \{e_5, e_5\}$ and $\geq(e') = \{e', e_3\}$ lead to another marking where only places p_4, p_6 has a token. Consider the lexicographic labeling order $\trianglelefteq$ based on the total order $\ll$ over labels such that $a \ll a' \ll b \ll c \ll c'$ (see Definition 4.2.17 on page 78. In this deterministic labeled event structure $\mathcal{E}$, e_3 is a cutoff event w.r.t. the local cutting context $(\mathcal{E}, \mathcal{I} \Cap \trianglerighteq, \mathcal{C}^l_{\mathcal{E}})$ due to e_1, and e_5 is a cutoff event due to e'.

Since the choice of expanding events in Algorithm 5.12 respects $\trianglelefteq$, e_3 is determined as an algorithmic cutoff event and its successor e' should not be added to the constructing prefix. And as a consequence, e_5 is neither an algorithmic cutoff event nor a cutoff event in the final prefix generated by the truncating algorithm. The final prefix which consists of 5 events in $E' = \{e_1, e_2, e_3, e_4, e_5\}$ is represented by Figure 5.2.b, while the truncation $\mathcal{T}(\mathcal{E}, \mathcal{I} \Cap \trianglerighteq, \mathcal{C}^l_{\mathcal{E}})$ is the set $\{e_1, e_2, e_3, e_4\}$ by definition.

Remark: We use the notation of *algorithmic* cutoff event for distinguishing between an event determined by Algorithm 5.12 and a cutoff event by Definition 4.3.1 on page 79. However, both kinds of cutoff conditions depend on the cutting context $(\mathcal{E}, \preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq, \mathcal{C}^l_{\mathcal{E}})$. One can deduce from Algorithm 5.12 a inductive definition of CE together with event set $\widehat{E}$ as follows:

- $e' \in \widehat{E}$ if $(\widehat{>}(e')) \cap \mathsf{CE} = \emptyset$, and
- $e' \in \mathsf{CE}$ if $e' \in \widehat{E}$ and e' is a cutoff event w.r.t. $(\mathcal{E}', \preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq, \mathcal{C}^l_{\widehat{\mathcal{E}}})$ where $\widehat{\mathcal{E}}$ is the $\widehat{E}$-prefix of $\mathcal{E}$.

In this way, the sets $\widehat{E}$ and CE are similar to the sets of *feasible events* and of *static cutoff events* in [Kho03]. The difference only comes from the fact that we use a local cutting context $(\mathcal{E}, \preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq, \mathcal{C}^l_{\mathcal{E}})$ in the place of the global one $(\mathcal{E}, \preccurlyeq^{\mathcal{M}} \Cap \trianglerighteq, \mathcal{C}_{\mathcal{E}})$. In other words, an algorithmic cutoff event is due to some local configuration while a static cutoff event is due to arbitrary configuration in $\widehat{E}$.

5.4.2 Complete prefixes

Theorem 5.4.3 (Termination)**.** *Let $(\mathcal{E}, \preccurlyeq^{\mathcal{C}})$ be a converse well-preordered labeled event structure where $\mathcal{E}$ is finitely-branching. If* Extend *is correct w.r.t. $\mathcal{E}$ and then Algorithm 5.12 terminates.*

Proof. Thanks to Definition 5.1.3, structure variables always give rise to some E'-prefix of $\mathcal{E}$ where E' is its actual event set. Since Extend is correct and $\mathcal{E}$ is finitely-branching,

Algorithm 5.12 does not terminates only if it calls the function Extend an infinite number of times. Parameters e of such calls are pairwise different due to the instruction at line 9. Let us denote by e_k, $k = 1, 2, \ldots$, the parameter of the k^{th} calling of Extend. By the same reasoning as in the proof of Theorem 4.2.15 on page 76, since $\mathcal{E}$ as well as its prefixes is finitely-branching, the infinite sequence $e_1, e_2, \ldots$ must contain an infinite subsequence $e_{i_1}, e_{i_2}, \ldots$ which are in causal order where $i_1, i_2, \ldots$ is a increasing sequence of indices. It follows from the converse well-preorder $\preccurlyeq^{\mathcal{C}}$ that there exits indices i_l and i_m such that $(\geq(e_{i_m})) \preccurlyeq^{\mathcal{C}} (\geq(e_{i_l}))$ and $i_l < i_m$. Moreover, since e_{i_l} is strictly smaller than e_{i_m} w.r.t. the causality, one has $(\geq(e_{i_m})) \prec^{\mathcal{C}} (\geq(e_{i_l}))$. Event e_{i_m} is thus a cutoff event and it contradicts to the fact that $\mathsf{isCutoff}(e_{i_m})$ returns false in the test at line 6. Therefore, Algorithm 5.12 must terminate. □

Theorem 5.4.4 (Termination)**.** *If* Extend *is correct w.r.t. $\mathcal{E}$ and Algorithm 5.12 terminates then the truncation $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$ is a subset of the final event set E' computed by this truncating algorithm.*

Proof. Let E^n denote the set of events in E which are neither a cutoff event nor an outer event w.r.t. the cutting context $(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$. Thanks to Proposition 5.4.1, for every event $e \in (E' \cap E^n)$, $\mathsf{isCutoff}(e)$ returns true and e should be extended by calling $\mathsf{Extend}(e)$ (line 10 in Algorithm 5.12). As a consequence of Definition 5.1.3 on page 90, one can prove by induction on local configurations' size that E^n is a subset of E'. Let E^c denote the set of cutoff events in $\mathcal{E}$, and let e' be any event in $\mathsf{Min}_{\leq}(E^c)$. Due to its minimality w.r.t. the causality $\leq$, one has $(>(e')) \cap E^c = \emptyset$. Hence, $(>(e')) \subseteq E^n \subseteq E'$. Event e' must be inserted into E' while extending some predecessor of e'. Therefore, E^n and $\mathsf{Min}_{\leq}(E^c)$ are both subset of E'. This lemma is thus a consequence of Lemma 4.3.2 on page 79 which states that $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}}) = E^n \cup \mathsf{Min}_{\leq}(E^c)$. □

The inclusion order $\subseteq$ is a particular case of adequate order $\trianglelefteq$ in local cutting contexts (see termination and boundedness problems in Section 4.3.3). Since the order of adding events Algorithm 5.12 respects the inclusion order, an event is algorithmic cutoff event if and only if it is a cutoff one. The following is straightforward.

Corollary 5.4.5. *When* Extend *is correct w.r.t. $\mathcal{E}$ and Algorithm 5.12 terminates, its generated prefix is the $\mathcal{T}(\mathcal{E}, \preccurlyeq^{\mathcal{C}}, \mathcal{C}^l_{\mathcal{E}})$-prefix of $\mathcal{E}$ if $\preccurlyeq^{\mathcal{C}}$ includes $\supseteq$.*

The final prefix obtained by the truncating algorithm sometimes is much bigger, in terms of number of events, than necessary. This problem may be reduces by using a better adequate order. Esparza has given in [ERV96] an example showing that one can obtain a prefix of polynomial size with a lexicography-based adequate order in the place of a prefix of exponential size with a sized-based adequate order. In our framework of modeling system by synchronized products of labeled event structures in a hierarchical way, the truncating technique using cutoff conditions may be applied only at the top level, i.e. the global labeled event structure.

However, each verification problem discussed in Section 4.3 may have a more suitable algorithm which is derived from Algorithm 5.12. For example, when deciding boundedness of a labeled event structure $\mathcal{E}$ such an algorithm can terminate if the first strictly cutoff event has been found. This on-the-fly algorithm, in the case where $\mathcal{E}$ is unbounded, generates in general a prefix which is more compact than the corresponding truncation.

Finally, it is worth noticing that when the adequate order is a total order over the event set E, Algorithm 5.12 is deterministic. Otherwise, the algorithm is non-deterministic and it is not clear that every run of it returns the same prefix. This

phenomenon was stated for the case of Petri nets in [HKK02]. It was shown there that, for any adequate order, all possible runs give the same prefix. We conjecture that it is the case also for our algorithm.

Chapter 6

Experimental results

Contents

We firstly demonstrate how to model an heterogeneous system and use our technique for verifying some properties on this system. The *Alternating Bit Protocol* (ABP) is taken as the case study in Section 6.1. Then, our model-checker Esu is briefly described in Section 6.2. The auto-concurrency problem of the unfolding technique will be discussed in this section. We also detail our technique for reducing redundancy that is integrated in Esu in order to attack the auto-concurrency problem. Finally, Section 6.3 is dedicated to experimental results as well as a comparison of Esu and other well-known tools.

6.1 Modeling and verification of heterogeneous systems

6.1.1 Alternating Bit Protocol

The *Alternating Bit Protocol (ABP)* [BSW69] is a connection-less protocol for transferring messages in one direction between two entities. These entities, called the *sender* and the *receiver*, exchange messages by means of two FIFO channels. This protocol guarantees the retransmission of lost or corrupted messages by using acknowledge bits. Intuitively, each message from both the sender and the receiver contains a bit, i.e. a value that is either 0 or 1. When the sender sends a message m, it sends it continuously, until it receives an acknowledgment bit from the receiver that is the same bit in m. When it happens, the sender starts transmitting the next message with the complement

of this bit. At the receiver side, when it receives a message with bit 0, it starts sending bit 0 as acknowledgment, and keep doing so until it receives another message with bit 1. Then, it starts sending bit 1, and so on.

In this example, we are not interested in the fact that the channels may corrupt messages as well as the way that the sender and the receiver decide whether a message is correct. For simplicity, we assume that exchanged messages are the bits 0 and 1 themselves. We will see further that ABP is tolerant to lost messages in channels.

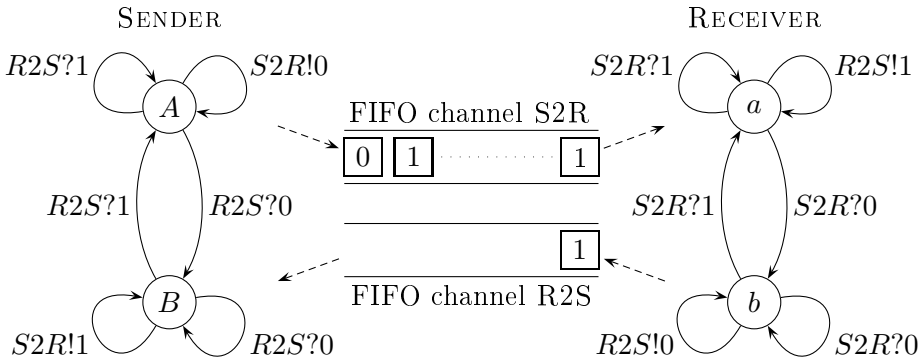


Figure 6.1: A model for the Alternating Bit Protocol

In other works, the ABP is generally modeled by two finite-state machines, that correspond to the sender and the receiver, communicating through two channels [AAB99]. Figure 6.1 illustrates such a model in which both the sender and the receiver have only two states. As previously assumed, these channels, named $S2R$ and $R2S$, are FIFO channels over $M = \{0, 1\}$. The sender, at the left side of Figure 6.1, may either insert messages into the channel $S2R$ or remove messages from the channel $R2S$. These actions are graphically represented by loops or curved arrows over the sender's states A and B, or more precisely by the label of these loops and arrows. In addition, an arc connecting two states intuitively means that the sender changes its state while a loop does not. For example, when the sender's state is A, it inserts only messages 0 into the channel $S2R$ until it removes a message 0 from the channel $R2S$. In this case, the sender changes its state to B and starts inserting messages 1 into the channel $S2R$. In the same manner, the receiver is shown at the right side of Figure 6.1.

6.1.2 Modeling the ABP as a synchronized product

The first thing we have to do is to decompose the ABP into some simple components. There are two reasons. First, we can use standard labeled transition systems for modeling these components, and as a consequence, we obtain their corresponding labeled event structures that are introduced in Section 3.3. The ABP may be modeled by both the synchronized product of these component labeled transition systems and the synchronized product of these labeled event structures. Second, by using unfolding algorithms of Chapter 5, we can iteratively construct such labeled event structures, and moreover, the synchronized product one inherits the concurrency of its components.

In our example, the ABP can be naturally considered as a composition of four components: a sender S, a receiver R, two FIFO channels $S2R$, $R2S$ over $\{0, 1\}$ for messages from the sender to the receiver and for messages from the receiver to the sender respectively. In addition, we assume that there is a counter in order to compute the number of successfully transmitted messages.

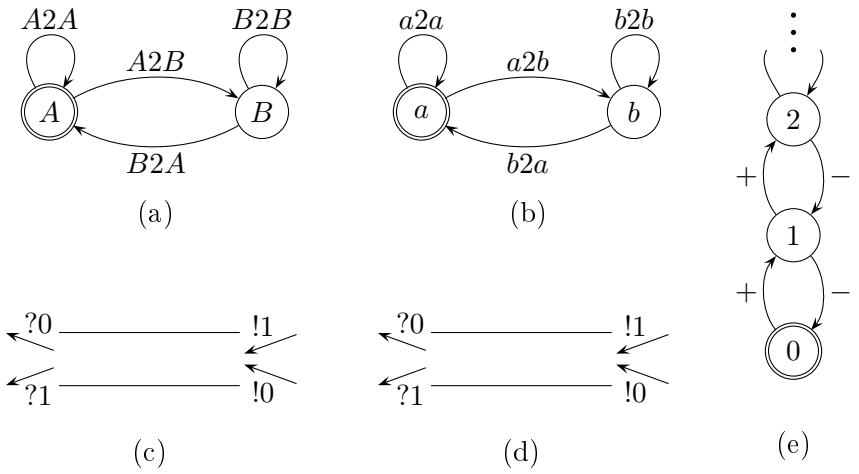


Figure 6.2: Components modeling the ABP: (a) Sender S, (b) Receiver R, (c) Channel $S2R$, (d) Channel $R2S$, (e) (Unbounded) counter.

The sender, as well as the receiver, can be simply modeled by a labeled transition system with two states A, B, and four transitions that correspond to the fact of staying on a same state: $A2A$, $B2B$; or changing from one state to the other state: $A2B$, $B2A$. We suppose that, at the initial state, the sender is in state A, the receiver is in state a, the two FIFO channels are empty, and the counter is set to 0. Hence, these two FIFO channels $S2R$ and $R2S$ may be represented by the $(\{0,1\},\varepsilon)$-$\mathcal{FF}$ labeled transition system (see Definition 3.3.22 on page 43), and the counter is formally defined by the 0-$\mathcal{CT}$ labeled transition system (see Definition 3.3.6 on page 34). Figure 6.2 illustrates these five components.

Sender	Receiver	FIFO S2R	FIFO R2S	Counter	*Vector name*
$A2A$	ε	!0	ε	ε	$A2AS!0$
$A2A$	ε	ε	?1	ε	$A2AR?1$
$A2B$	ε	ε	?0	ε	$A2BR?0$
$B2B$	ε	!1	ε	ε	$B2BS!1$
$B2B$	ε	ε	?0	ε	$B2BR?0$
$B2A$	ε	ε	?1	ε	$B2AR?1$
ε	$a2a$	ε	!1	ε	$a2aR!1$
ε	$a2a$	?1	ε	ε	$a2aS?1$
ε	$a2b$	?0	ε	+	$a2bS?0+$
ε	$b2b$	ε	!0	ε	$b2bR!0$
ε	$b2b$	?0	ε	ε	$b2bS?0$
ε	$b2a$	?1	ε	+	$b2aS?1+$

Table 6.1: Synchronization constraint for the ABP with counter of successfully transmitted messages.

The ABP is then modeled by a synchronized product of these five labeled transition systems. Therefore, its set of states is $S = \{A,B\}\times\{a,b\}\times\{0,1\}^*\times\{0,1\}^*\times\mathbb{N}$. Table 6.1 shows all synchronization vectors, i.e. actions in Σ, of the synchronized product. For

example, the first vector $A2AS!0 = \langle A2A, \varepsilon, !0, \varepsilon, \varepsilon \rangle$ means that the action $A2A$ of the sender (Figure 6.2.a) must be synchronized with the sending action !0 of the FIFO channel $S2R$ (in Figure 6.2.c). This global action intuitively corresponds to the loop with label $S2R!0$ over the state A of the sender in Figure 6.1. Notice that, since the counter computes the number of successfully transmitted messages, its increment action '+' should be synchronized with receiving actions of the channel $S2R$ (?0 or ?1) that change the state of the receiver R ($a2b$ or $b2a$).

6.1.3 Verification of counter's boundedness

We investigate the question whether the number of successfully transmitted messages is bounded if the two channels are bounded. Suppose that these two FIFO channels are bounded by 2. The answer "no" means that by using the ABP, we can transmit as many messages as we want from the sender to the receiver.

To exploit the concurrency, we prefer to use labeled event structures than labeled transition systems in order to model components of the ABP, and to verify the property above. The synchronized product of these labeled event structures w.r.t. the synchronization vectors in Table 6.1 represents then all behaviors of the ABP. Hence, the synchronized product is sufficient for verifying this property. Here, we use the $(\{0,1\}, \varepsilon, 2)$-causality process, denoted by $\mathcal{E}_{\mathcal{FF}} = (\{0,1\}, \varepsilon, 2)\text{-}\mathcal{CP}$ (see Section 3.3.3), for both bounded channels $S2R$ and $R2S$. Figure 6.3 graphically shows $\mathcal{E}_{\mathcal{FF}}$. Labeled event trees introduced in Section 3.3.1 are convenient for the sender as well as for the receiver, because they have no concurrency. Finally, one can let any causality process given in Section 3.3.2 model the counter. Let $\mathcal{E}_S, \mathcal{E}_R, \mathcal{E}_{\mathcal{CT}}$ respectively denote the labeled event structures for the sender, the receiver, and the counter.

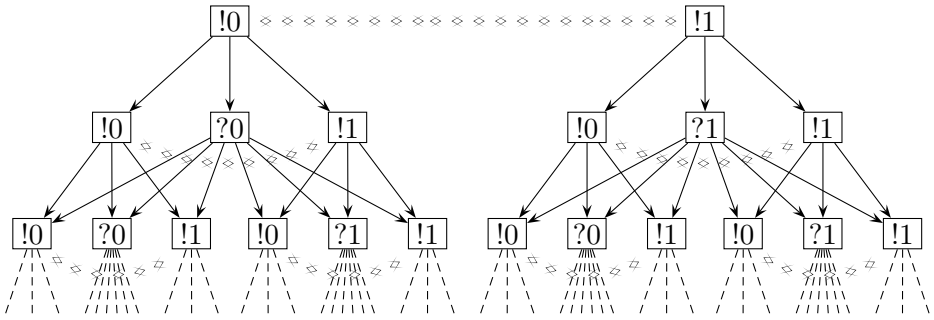


Figure 6.3: The $(\{0,1\}, \varepsilon, 2)$-causality process for the empty-initialized FIFO channel over $\{0,1\}$ that is bounded by 2, i.e. $(\{0,1\}, \varepsilon, 2)\text{-}\mathcal{BC}$.

Notice that these component labeled event structures $\mathcal{E}_{\mathcal{FF}}, \mathcal{E}_S, \mathcal{E}_R$, and $\mathcal{E}_{\mathcal{CT}}$ are all deterministic and coherent. The identity relation "=" is compatible with $\mathcal{E}_{\mathcal{FF}}, \mathcal{E}_S, \mathcal{E}_R$ due to the finiteness of their marking sets, and moreover, with a reflexive and strict compatibility. One can verify that $(\mathcal{E}_{\mathcal{CT}}, \leq)$ is also a well-preordered labeled event structure with reflexive and strict compatibility where $\leq$ is the "less than or equal" relation over $\mathbb{N}$. Let $\mathcal{E}$ denote the synchronized product of $\mathcal{E}_S, \mathcal{E}_R, \mathcal{E}_{\mathcal{FF}}, \mathcal{E}_{\mathcal{FF}}, \mathcal{E}_{\mathcal{CT}}$ w.r.t. the synchronization constraint Σ shown in Table 6.1; and let $\preccurlyeq$ be the product preorder $\otimes(=, =, =, =, \leq)$. Thanks to Lemma 4.1.8 on page 65, $(\mathcal{E}, \preccurlyeq)$ is a well-preordered labeled event structure with reflexive and strict compatibility. Here, there is no internal action, i.e. $\Sigma^\tau = \emptyset$.

Therefore, we can use the truncation technique to answer the boundedness problem (as described in Section 4.3.3).

Let us give some details on how the truncation algorithm (Algorithm 5.12 on page 123) works based on the local cutting context $(\succcurlyeq \Subset \supseteq, \mathcal{C}^l)$. It is worth noticing that the adequate order is the subset order over (local) configurations, i.e. $\trianglelefteq = \subseteq$. One may obtain the answer "yes" which means that the ABP is unbounded without completely constructing the truncation $\mathcal{T}(\mathcal{E}, \succcurlyeq \Subset \subseteq, \mathcal{C}^l)$. In other word, any strict cut-off event is enough to conclude the unboundedness. Moreover, the verifying process can quickly terminate while using depth-first-search. That means, in the unfolding algorithm, when an event e is extended before another event e', the successors of e should be extended before e' too if these successors are not cut-off events. In order to do so, we simply implement the possible extension PE (see Algorithm 5.12) as a stack based on the principle last-come-first-served.

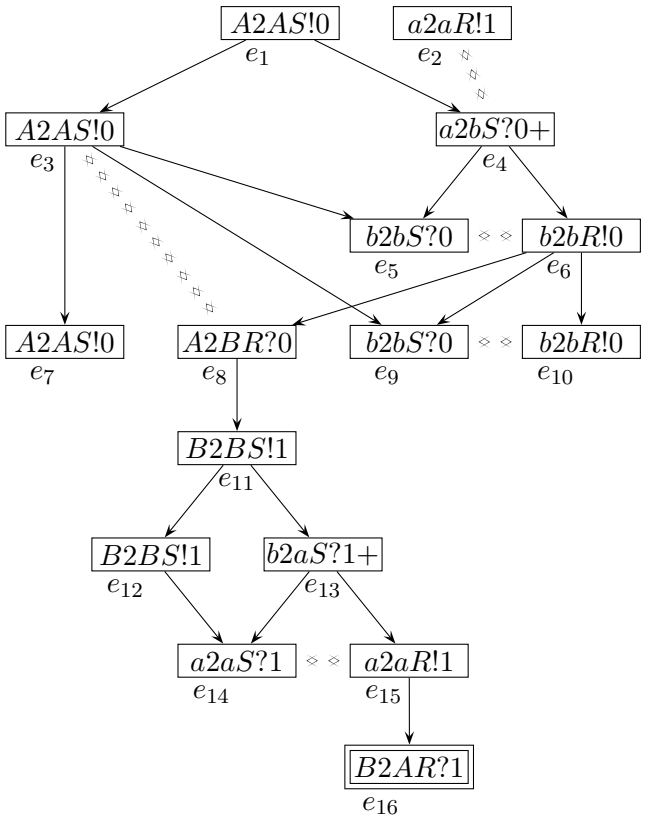


Figure 6.4: Obtained prefix for boundedness problem of the ABP initialized by $s^0 = \langle A, a, \varepsilon, \varepsilon, 0\rangle$.

Figure 6.4 illustrates the prefix of $\mathcal{E}$ that is generated by our algorithm. Its event set is thus a subset of the truncation $\mathcal{T}(\mathcal{E}, \succcurlyeq \Subset \subseteq, \mathcal{C}^l)$. The local configuration of event e_{16} is the set $\geq(e_{16}) = \{e_1, e_4, e_6, e_8, e_{11}, e_{13}, e_{15}, e_{16}\}$, and its marking is $\mathcal{M}(\geq(e_{16})) = \langle A, a, \varepsilon, \varepsilon, 2\rangle$. Event e_{16} is thus a marking-strict cutoff event due to the particular event ε, or more precisely, due to the empty configuration $\emptyset \in \mathcal{C}^l$, because $\mathcal{M}(\emptyset) = \langle A, a, \varepsilon, \varepsilon, 0\rangle \prec \langle A, a, \varepsilon, \varepsilon, 2\rangle = \mathcal{M}(\geq(e_{16}))$ (see Definition 4.3.18 on page 84). Thanks to Theorem 4.3.19, the ABP is unbounded. Hence one can conclude that the counter counting successfully transmitted messages is unbounded too because the sender and the receiver have finite

states and the two channels are bounded.

6.1.4 Verification of lossy FIFOs' coverability

Now, assume that the two channels $S2R$ and $R2S$ may loose messages. There are different formal models for these lossy channels, e.g. the v-initialized lossy FIFO channels over $\{0,1\}$ with or without internal actions Σ^τ (see Definition 4.1.2 on page 63 and Definition 4.1.3 on page 63). However, for the simplicity of the demonstration, we prefer to define lossy FIFO channels over $\{0,1\}$ as the labeled transition systems $\mathcal{FL} = (M^*, !M \cup ?M, \rightarrow_{\mathcal{FL}}, \varepsilon)$ where $M = \{0,1\}$, and

$$\begin{aligned}\rightarrow_{\mathcal{FL}} = &\{\langle w, !m, w'\rangle \,/\, m \in M, w, w' \in M^* \text{ and } w' \preccurlyeq w.m\} \\ &\cup \{\langle w''.m.w, ?m, w'\rangle \,/\, m \in M, w, w', w'' \in M^* \text{ and } w' \preccurlyeq w\}\end{aligned}$$

In the definition above, $\preccurlyeq$ is the subword order over M^* (see Definition 2.2.1 on page 12). The labeled transition system $\mathcal{FL}$ intuitively means that the channel may loose messages at the moment of sending or receiving a message. In other words, for example, a receiving operation $?m$ from a word $w''.m.w$ consists of the loss of its prefix w'', the normal receiving action $?m$, and finally another loss of messages in w. Moreover, one could find out that the subword order $\preccurlyeq$ is reflexively compatible with the transition relation $\rightarrow_{\mathcal{FL}}$.

Let us use the same labeled transition systems as in the previous section for the sender, the receiver, and the counter, and denote them respectively by $\mathcal{LTS}_S, \mathcal{LTS}_R$, and $\mathcal{CT}$. The lossy ABP, denoted by $\mathcal{SP}$, is the synchronized product of $\mathcal{LTS}_S, \mathcal{LTS}_R, \mathcal{FL}, \mathcal{FL}, \mathcal{CT}$ w.r.t. the synchronization constraint $\Sigma_{\mathcal{SP}}$ shown in Table 6.1. One can easily verify that the product preorder $\preccurlyeq_{\mathcal{SP}} = \otimes(=, =, \preccurlyeq, \preccurlyeq, \leq)$ is compatible with $\mathcal{SP}$ with a reflexive and strong compatibility.

In the alternating bit protocol, the sender continues to transmit a new message only if the receiver has already received the previous one and has replied by sending an acknowledgment. Assume that, in our simple model of the ABP, the old message corresponds to some consecutive messages 0 in the channel $S2R$. The possibility of transmitting a new message corresponds thus to the fact that the sender's state is B. In this case, one may obviously deduce that, in the channel $S2R$, such consecutive messages 0 may not be preceded by some message 1. Hence, all states $\langle B, s, 10, w, n\rangle$ where $s \in \{a, b\}, w \in M^*, n \in \mathbb{N}$, are not reachable in $\mathcal{SP}$. For instance, the state $\langle B, b, 10, \varepsilon, 0\rangle$ is not covered in the well-preordered transition system $(\mathcal{SP}, \preccurlyeq_{\mathcal{SP}})$, i.e. $\langle B, b, 10, \varepsilon, 0\rangle \notin \succcurlyeq_{\mathcal{SP}}(\mathsf{post}^*_{\mathcal{SP}})$. In the view of backward analysis, it is formulated as the following:

$$\langle A, a, \varepsilon, \varepsilon, 0\rangle \notin \mathsf{pre}^*_{\mathcal{SP}}(\preccurlyeq_{\mathcal{SP}}(\langle B, b, 10, \varepsilon, 0\rangle))$$

We will verify this statement based on our forward analysis technique discussed in Section 4.1.3. Let us define a function $\mathsf{pb} : (!M \cup ?M) \times M^* \rightarrow \mathcal{P}_f(M^*)$ such that, for all $m \in M$ and $w' \in M^*$,

- $\mathsf{pb}(!m, w'.m) = \{w', w'.m\}$,
- $\mathsf{pb}(!m, w') = \{w'\}$ if $w' \in (M^* \setminus (M^*.m))$, and
- $\mathsf{pb}(?m, w') = \{m.w'\}$.

The two first properties correspond to the sending actions $!m$ in the lossy FIFO channel $\mathcal{FL}$ above while the last one corresponds to the receiving actions $?m$ where $m \in M$. One can easily verify that pb is a finite pred-basis for $(\mathcal{FL}, \preccurlyeq)$ by Definition 4.1.13. As a

consequence, let us denote $\mathcal{R}_{\mathcal{FL}} = (M^*, !M \cup ?M, \rightarrow_{\mathcal{R}_{\mathcal{FL}}}, \varepsilon)$ the pb-reverse of $(\mathcal{FL}, \preccurlyeq, \mathsf{pb})$. We obtain thus

$$\begin{aligned}
\rightarrow_{\mathcal{R}_{\mathcal{FL}}} = & \{\langle w'.m, !m, w'\rangle, \langle w'.m, !m, w'.m\rangle \,/\, m \in M, w' \in M^*\} \\
& \cup \{\langle w', !m, w'\rangle \,/\, m \in M, w' \in (M^* \setminus (M^*.m))\} \\
& \cup \{\langle w', ?m, m.w'\rangle \,/\, m \in M, w' \in M^*\} \\
= & \{\langle w'.m, !m, w'\rangle \,/\, m \in M, w' \in M^*\} \\
& \cup \{\langle w', !m, w'\rangle \,/\, m \in M, w' \in M^*\} \\
& \cup \{\langle w', ?m, m.w'\rangle \,/\, m \in M, w' \in M^*\}
\end{aligned}$$

Here, for the purpose of an easy understanding, we rename the actions in $\mathcal{R}_{\mathcal{FL}}$ by using $?m \in ?M$ instead of $!m \in !M$ and reversely. Then, $\mathcal{R}_{\mathcal{FL}}$ differs from the ε-initialized FIFO channel over M, i.e. (M, ε)-$\mathcal{FF}$, only on the *reception-error* transitions $\{\langle w', ?m, w'\rangle \,/\, m \in M, w' \in M^*\}$. These transitions intuitively correspond to sending actions of messages that the channel loses afterward. Therefore, we associate a simple labeled event structure that is derived from the $\{0,1\}$-causality process $\{0,1\}$-$\mathcal{CP}$ (see Definition 3.3.27) in order to representing $\mathcal{R}_{\mathcal{FL}}$. This labeled event structure, denoted by $\mathcal{E}_{\mathcal{R}_{\mathcal{FL}}}$, is shown in Figure 6.5.a. The additional events illustrated by double frames correspond to reception-errors. They are not in conflict with any other events. In our example, for simplicity, we restrict reception-error events of a given label $?m \in ?M$ to be pairwise causal. While considering $\mathcal{R}_{\mathcal{FL}}$ as a FIFO channel like system where its state is a word, messages are inserted at the beginning of the word by sending actions, and one removes messages at the end of the word by receiving actions.

Let us denote by $E_!, E_?$, and $E'_?$ respectively the sets of sending events, of (normal) receiving events and of reception-error events in $\mathcal{E}_{\mathcal{R}_{\mathcal{FL}}}$. These event sets are pairwise disjoint. The marking function $\mathcal{M}$ is then defined like in the $(\{0,1\}, \varepsilon)$-causality process (Definition 3.3.27) that formally is $\mathcal{M}(C) = (\Pi^{\mathcal{W}}_{?M}(\mathcal{L}^{\mathcal{W}}(\sigma_?)))^{-1}(\Pi^{\mathcal{W}}_{!M}(\mathcal{L}^{\mathcal{W}}(\sigma_!)))$ where $\sigma_!$ and $\sigma_?$ are respectively the linearisations, w.r.t. the converse relation $\geq$ of the causality $\leq$, of $(C \cap E_!)$ and $(C \cap E_?) = (C \setminus E_! \setminus E'_?)$. Recall that reception-errors in $\mathcal{R}_{\mathcal{FL}}$ do not change the content of the channel, the marking function $\mathcal{M}$ does not take reception-errors events $E'_?$ into account. More interestingly, events in $E'_?$ give the strong compatibility in the well-preordered labeled transition system $(\mathcal{E}_{\mathcal{R}}, \succcurlyeq)$.

Figure 6.5.b illustrates our labeled event structure $\mathcal{E}_{\mathcal{R}_{\mathcal{FL}'}}$ for the pb-reverse of $(\mathcal{FL}', \preccurlyeq, \mathsf{pb})$ where $\mathcal{FL}'$ is the labeled transition system modeling the 10-initialized lossy FIFO channel over $\{0,1\}$, i.e. $\mathcal{FL}' = (M^*, !M \cup ?M, 10, \rightarrow_{\mathcal{FL}})$. Intuitively, $\mathcal{E}_{\mathcal{R}_{\mathcal{FL}'}}$ comes from the $(\{0,1\}, 01)$-causality process (see Definition 3.3.31 on page 49).

For the sender $\mathcal{LTS}_S$, the receiver $\mathcal{LTS}_R$, and the counter $\mathcal{CT}$, we simply define three finite pre-basis $\mathsf{pb}_S, \mathsf{pb}_R$, and $\mathsf{pb}_{\mathcal{CT}}$ respectively as follows:

- $\mathsf{pb}_S(l, s) = \{s' \,/\, s' \xrightarrow{l}_S s\}$ for all $\langle l, s\rangle \in \{A2A, A2B, B2A, B2B\} \times \{A, B\}$,
- $\mathsf{pb}_R(l, s) = \{s' \,/\, s' \xrightarrow{l}_R s\}$ for all $\langle l, s\rangle \in \{a2a, a2b, b2a, b2b\} \times \{a, b\}$, and
- $\mathsf{pb}_{\mathcal{CT}}(l, n) = \{n' \,/\, n' \xrightarrow{l}_{\mathcal{CT}} n\}$ for all $\langle l, n\rangle \in \{+, -\} \times (\mathbb{N} \setminus \{0\})$,
  $\mathsf{pb}_{\mathcal{CT}}(+, 0) = \{1\}$, and $\mathsf{pb}_{\mathcal{CT}}(-, 0) = \{0\}$.

It is worth noticing that $\mathcal{LTS}_S, \mathcal{LTS}_R, \mathcal{CT}$ are all deterministic and their dual labeled transition systems are themselves respectively (see Definition 4.1.12 on page 66). The

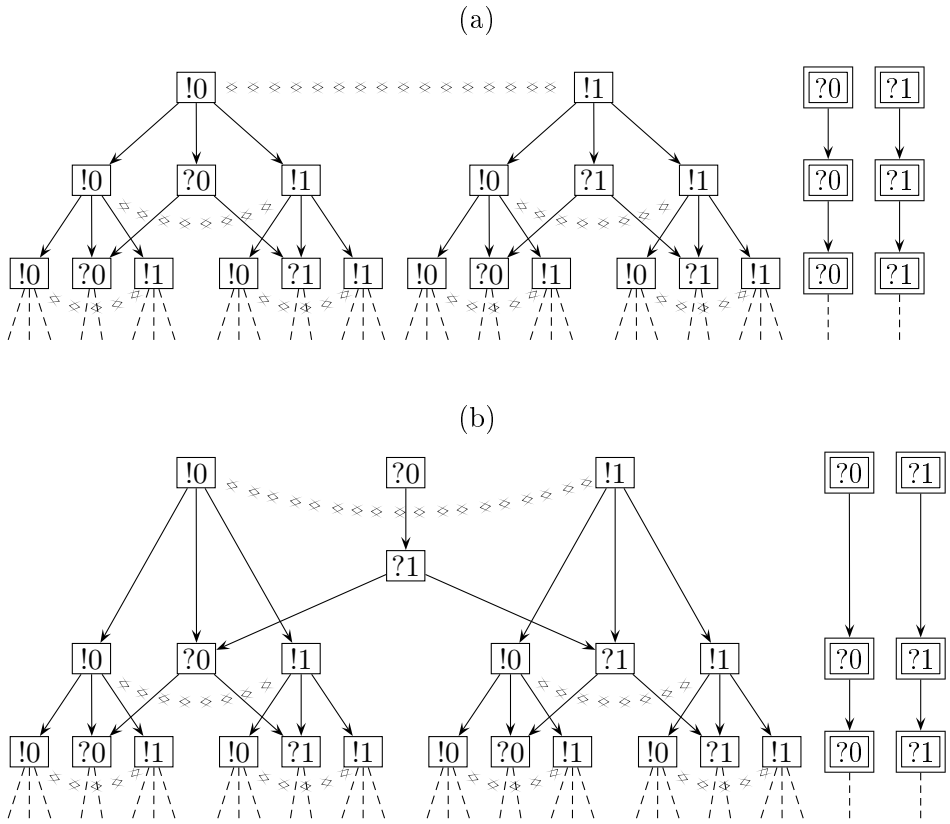


Figure 6.5: Labeled event structures modeling the pb-reverse $\mathcal{R}_{\mathcal{FL}}$ and $\mathcal{R}_{\mathcal{FL}'}$ of lossy FIFO channels $\mathcal{FL}$ and $\mathcal{FL}'$ over $\{0,1\}$ where: (a) $\mathcal{FL}$ is initially empty, and (b) the initial state of $\mathcal{FL}'$ is 10.

three finite pre-basis above give rise to these (dual) labeled transition systems by means of their pb-reverse. For example, the pb-reverse of $(\mathcal{LTS}_S, =, \mathsf{pb}_S)$, denoted by $\mathcal{R}_S$, is intuitively the well-preordered transition labeled system $(\mathcal{LTS}_S, =)$ if one renames actions in $\mathcal{R}_S$ so that $A2B$ becomes $B2A$, and $B2A$ becomes $A2B$ reversely. Therefore, one may use the preordered labeled event tree $\mathcal{E}_S$ ($\mathcal{E}_R$) for representing the pb-reverse $\mathcal{R}_S$ ($\mathcal{R}_R$ resp.) (see Section 3.3.1) and any k-causality process $\mathcal{CP}$ for representing the pb-reverse $\mathcal{R}_{\mathcal{CT}}$ of $(\mathcal{CT}, \leq, \mathsf{pb}_{\mathcal{CT}})$ (see Section 3.3.2).

Now, let us define the finite pre-basis $\mathsf{pb}_{\mathcal{SP}}$ of the lossy ABP $\mathcal{SP}$ such that, for all $l = \langle l_S, l_R, l_{S2R}, l_{R2S}, l_{\mathcal{CT}}\rangle \in \Sigma_{\mathcal{SP}}$, $s = \langle s_S, s_R, s_{S2R}, s_{R2S}, s_{\mathcal{CT}}\rangle \in S_{\mathcal{SP}} = \{A, B\} \times \{a, b\} \times (!M \cup ?M) \times (!M \cup ?M) \times \{+, -\}$,

$$\mathsf{pb}_{\mathcal{SP}}(l, s) = \mathsf{pb}_S(l_S, s_S) \times \mathsf{pb}_R(l_R, s_R) \times \mathsf{pb}(l_{S2R}, s_{S2R}) \times \mathsf{pb}(l_{R2S}, s_{R2S}) \times \mathsf{pb}_{\mathcal{CT}}(l_{\mathcal{CT}}, s_{\mathcal{CT}})$$

The pb-reverse $\mathcal{R}_{\mathcal{SP}}$ of $(\mathcal{SP}, \preccurlyeq_{\mathcal{SP}}, \mathsf{pb}_{\mathcal{SP}})$ is finally a synchronized product of the pb-reverses $\mathcal{R}_S$, $\mathcal{R}_R$, $\mathcal{R}_{\mathcal{FL}}$, $\mathcal{R}_{\mathcal{FL}'}$, and $\mathcal{R}_{\mathcal{CT}}$. Moreover, $\mathcal{R}_{\mathcal{SP}}$ is the induced labeled transition system of a synchronized product $\mathcal{E}_{\mathcal{SP}}$ of $\mathcal{E}_S$, $\mathcal{E}_R$, $\mathcal{E}_{\mathcal{R}_{\mathcal{FL}}}$, $\mathcal{E}_{\mathcal{R}_{\mathcal{FL}'}}$, and $\mathcal{E}_{\mathcal{CT}}$. It is worth noticing that, due to the change of actions' name discussed above, the synchronized constraint $\Sigma'_{\mathcal{R}_{\mathcal{SP}}}$ for $\mathcal{R}_{\mathcal{SP}}$ as well as for $\mathcal{E}_{\mathcal{SP}}$ that is shown in Table 6.2 slightly differs from the one for the synchronized product $\mathcal{SP}$ given in Table 6.1.

Thanks to Lemma 4.1.11 on page 66, both $(\mathcal{R}_{\mathcal{SP}}, \succcurlyeq_{\mathcal{SP}})$ and $(\mathcal{E}_{\mathcal{SP}}, \succcurlyeq_{\mathcal{SP}})$ are well-

Sender	Receiver	FIFO S2R	FIFO R2S	Counter	*Vector name*
$A2A$	ε	$?0$	ε	ε	$A2AS?0$
$A2A$	ε	ε	$!1$	ε	$A2AR!1$
$A2B$	ε	ε	$!1$	ε	$A2BR!1$
$B2B$	ε	$?1$	ε	ε	$B2BS?1$
$B2B$	ε	ε	$!0$	ε	$B2BR!0$
$B2A$	ε	ε	$!0$	ε	$B2AR!0$
ε	$a2a$	ε	$?1$	ε	$a2aR?1$
ε	$a2a$	$!1$	ε	ε	$a2aS!1$
ε	$a2b$	$!1$	ε	$-$	$a2bS!1-$
ε	$b2b$	ε	$?0$	ε	$b2bR?0$
ε	$b2b$	$!0$	ε	ε	$b2bS!0$
ε	$b2a$	$!0$	ε	$-$	$b2aS!0-$

Table 6.2: Synchronization constraint of the synchronized product $\mathcal{R}_{\mathcal{SP}}$.

preordered with reflexive compatibility. Notice that the initial state of $\mathcal{R}_{\mathcal{SP}}$ is $\langle B, b, 10, \varepsilon, 0\rangle$. It follows thus from Lemma 4.1.14 that

$$\begin{aligned}\mathsf{pre}^*_{\mathcal{SP}}(\preccurlyeq_{\mathcal{SP}}(\{\langle B, b, 10, \varepsilon, 0\rangle\})) &= \preccurlyeq_{\mathcal{SP}}(\mathsf{post}^*_{\mathcal{R}_{\mathcal{SP}}}(\{\langle B, b, 10, \varepsilon, 0\rangle\})) \\ &= \preccurlyeq_{\mathcal{SP}}(\mathsf{post}^*_{\mathcal{R}_{\mathcal{SP}}})\end{aligned}$$

Moreover, $(\mathcal{E}_{\mathcal{SP}}, \succcurlyeq_{\mathcal{SP}})$ is converse well-preordered labeled event structure with reflexive (strong) compatibility, the sub-covering question can be answered by using our truncation technique for sub-coverability problem (see Section 4.3.2).

By using the truncating algorithm (Algorithm 5.12 on page 123), we obtain the truncation $\mathcal{T} = (\mathcal{E}_{\mathcal{SP}}, (\succcurlyeq_{\mathcal{SP}} \Cap \trianglerighteq), \mathcal{C}^l_{\mathcal{E}_{\mathcal{SP}}})$ that is illustrated in Figure 6.6.a. Boxes with double frame represent cutoff events, and among them, the ones with dashed frame are reception-errors. Experimental result gives 58 reachable states that are the marking of all configurations in the truncation $\mathcal{T}$. The table in Figure 6.6.b shows all maximal markings w.r.t. the product preorder $\succcurlyeq_{\mathcal{SP}}$ as well as configurations in accordance. Therefore, thanks to Lemma 4.3.9 on page 81, one can deduce that

$$\begin{aligned}\mathsf{pre}^*_{\mathcal{SP}}(\preccurlyeq_{\mathcal{SP}}(\{\langle B, b, 10, \varepsilon, 0\rangle\})) &= \preccurlyeq_{\mathcal{SP}}(\mathsf{post}^*_{\mathcal{R}_{\mathcal{SP}}}) \\ &= \preccurlyeq_{\mathcal{SP}}(\mathcal{M}(\mathcal{C}_{\mathcal{T}}))\end{aligned}$$

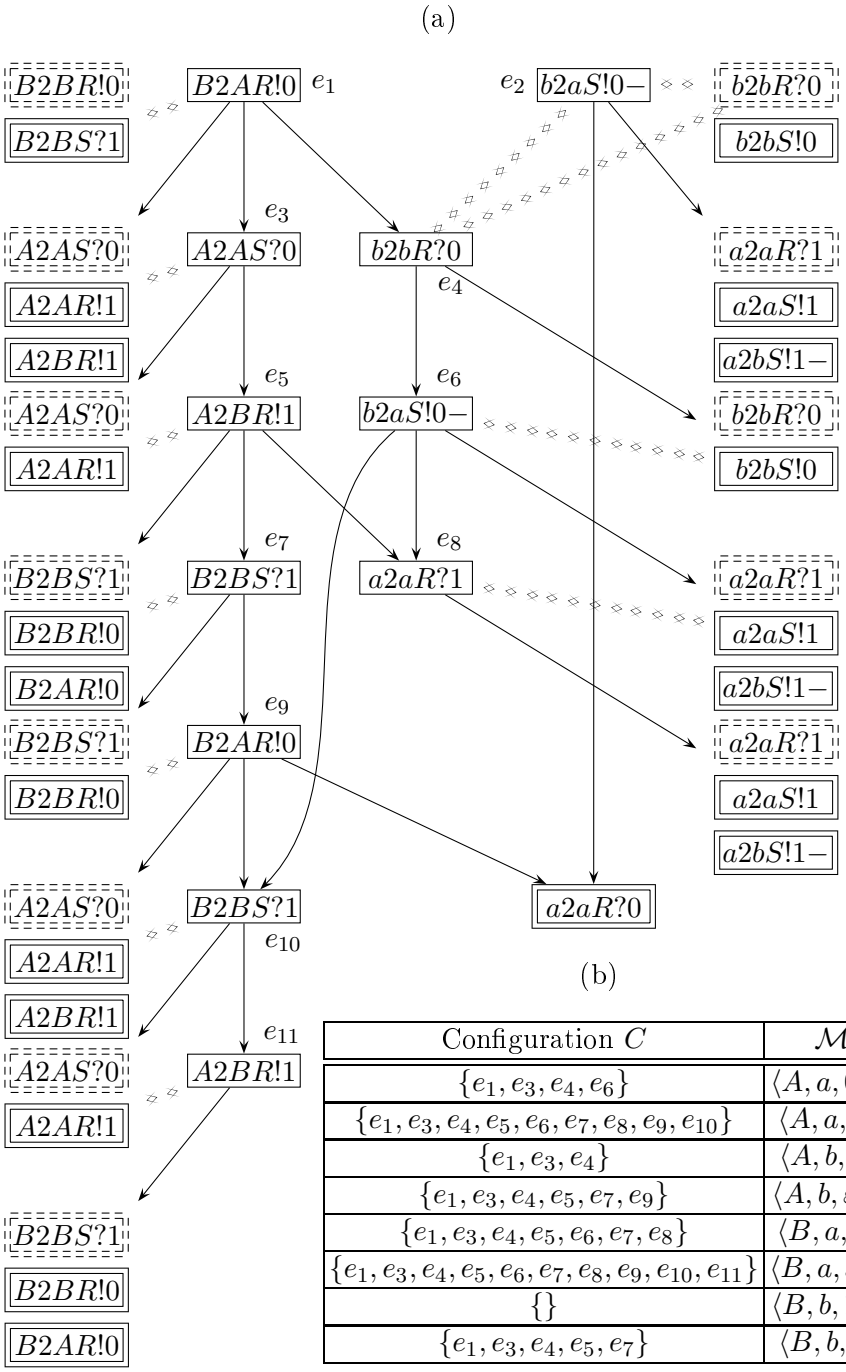
Hence,

$$\begin{aligned}\big(\preccurlyeq_{\mathcal{SP}}(\mathsf{post}^*_{\mathcal{R}_{\mathcal{SP}}})\big) \cap \{\langle A, a, w, w', c\rangle \,/\, w, w' \in M^*, c \in \mathbb{N}\} \\ = (\preccurlyeq_{\mathcal{SP}}(\langle A, a, 01, \varepsilon, 0\rangle)) \cup (\preccurlyeq_{\mathcal{SP}}(\langle A, a, \varepsilon, 0, 0\rangle))\end{aligned}$$

and thus, $\langle A, a, \varepsilon, \varepsilon, 0\rangle \notin \Big(\preccurlyeq_{\mathcal{SP}}(\mathsf{post}^*_{\mathcal{R}_{\mathcal{SP}'}})\Big)$. It means that, in the lossy ABP $\mathcal{SP}$, one can never obtain states $\langle B, b, 10, \varepsilon, n\rangle$, for all $n \in \mathbb{N}$, from the initial state $\langle A, a, \varepsilon, \varepsilon, 0\rangle$.

6.2 The tool Esu

In order to test applicability of the results previously shown, we have developed a model-checker named Esu [esu]. This tool is implemented in *Objective Caml (Ocaml)* and



Configuration C	$\mathcal{M}(C)$
$\{e_1, e_3, e_4, e_6\}$	$\langle A, a, 01, \varepsilon, 0\rangle$
$\{e_1, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$	$\langle A, a, \varepsilon, 0, 0\rangle$
$\{e_1, e_3, e_4\}$	$\langle A, b, 1, \varepsilon, 0\rangle$
$\{e_1, e_3, e_4, e_5, e_7, e_9\}$	$\langle A, b, \varepsilon, 01, 0\rangle$
$\{e_1, e_3, e_4, e_5, e_6, e_7, e_8\}$	$\langle B, a, 0, \varepsilon, 0\rangle$
$\{e_1, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\}$	$\langle B, a, \varepsilon, 10, 0\rangle$
$\{\}$	$\langle B, b, 10, \varepsilon, 0\rangle$
$\{e_1, e_3, e_4, e_5, e_7\}$	$\langle B, b, \varepsilon, 1, 0\rangle$

Figure 6.6: Truncation for sub-coverability problem of $\mathcal{E}_{\mathcal{R}_{\mathrm{SP}}}$ where $\mathcal{M}(\emptyset) = \langle B, b, 10, \varepsilon, 0\rangle$: (a) The truncation, (b) Maximal markings w.r.t. $\succcurlyeq_{\mathrm{SP}}$.

permits the verification of termination, boundedness and quasi-liveness properties for the class of (infinite-state) well-structured systems.

Systems are modeled in Esu as synchronized products of (heterogeneous) components in a hierarchical way: a component itself can be a synchronized product of other components. The semantics of components is given in terms of labeled event structures. Esu has three important modules dedicated to modeling systems, unfolding (synchronized products) and the truncation technique respectively.

Component labeled event structures In fact, in the implementation of our Esu tool, each labeled event structure is represented by an object of a class in which Extend is a method and structure variables are instance variables (see Chapter 5). The construct of a synchronized product can be done on the fly, and of course it is not necessary to construct the components completely in advance but they can be constructed on demand too.

Several standard systems, e.g. counter, queue, are predefined in Esu by simply defining concurrent labeled event structures given in Section 3.3 on page 32. Esu facilitates also extensions by new types of components.

Unfolding synchronized product Due to the constructive definition of the unfolding, a synchronized product can be used as a component in Esu. Hence, we associate synchronized products to a class derived from the base class for labeled event structures. In addition, this class has an instance variable for synchronisation vectors $\mathcal{V}$ (see Section 3.3.4 on page 54). The well-known unfolding technique, more precisely function Extend for synchronization products [ER99], is implemented in this class (see Section 5.3 on page 109). In order to extend a prefix of a synchronized product, the main part of this function Extend is computing new possible combinations of events in component labeled event structures, and consequently, updating $\mathcal{V}$.

Truncation technique This module concerns truncation techniques given in Chapter 4. In fact, Algorithm 5.12 on page 123 is implemented with local cutting contexts in Section 4.3 on page 78. Esu provides not only McMillan's and Esparza's techniques [McM95a, ERV96] for bounded Petri nets but also our techniques [HST07] for termination, boundedness and quasi-liveness of (infinite) well-preordered systems (see Section 4.3.3 on page 83).

Esu has its own file format in order to describe input systems. Some details on this format are given in the next section. In addition, Esu also provides a converter that allows transforming a standard net's file [pep] to Esu's one. As a command-line program, Esu's options give users a versatility control of what verification problem to solve, of what technique to use, and also how the results are reported to users. Thanks to the Graphviz application [gra], Esu permits users to have a graphical representation of the generated prefix.

6.2.1 Modeling Petri nets

Since all experimental results in Section 6.3 are taken for Petri nets, let us detail on how Petri nets are modeled in Esu. As discussed in Section 2.5 and along with this work, we assume that a Petri net is generally a synchronized product of n counters where n is the number of its places. Within Esu, one can associate any labeled event structure given in Section 3.3.2 to each place, and the synchronization constraint corresponds to the set of the Petri net's transitions.

However, in some Petri nets, for instance, the one given in Example 5.4.2 on page 124, the place p_2 may not be represented by neither a counter nor a k-causality process. Because this place concerns the transition b that tests whether p_2 contains a token but firing b does not consume any token on p_2. Therefore, when modeling p_2 (and also p_4) as a counter-like labeled transition system, we need a new action in addition to the increment and the decrement ones. Formally, the place p_2 may be represented by the labeled transition system $\mathcal{P} = (\mathbb{N}, \{+, -, o\}, \rightarrow, 1)$ where $\rightarrow = \{\langle n, +, n+1\rangle, \langle n+1, -, n\rangle, \langle n+1, o, n+1\rangle \,/\, n \in \mathbb{N}\}$.

```
6             // number of places (counters)
BP 1 1        // p1
P1 1          // p2
KP 1 1        // p3
P1 1          // p4
BP 1 0        // p5
BP 1 0        // p6
5             // number of synchronized actions
N 0 - N N N // a
N N - 0 N N // b
- N - N + N // c
- - N N N + // a'
N - N N - + // c'
```

Figure 6.7: An example of Esu's input file

It is worth noticing that this Petri net is 1-safe. As a consequence, one can associate not only some labeled event structure of $\mathcal{P}$ but also any labeled event structure of $\mathcal{P}1 = \mathcal{P}|_{\{0,1\}}$ to the place p_2. One may realize that the actions $+, -, o$ of $\mathcal{P}1$ are pairwise not independent. Therefore, we simply use the labeled event tree of $\mathcal{P}1$ in order to represent the place p_2 as well as the place p_4. The Petri net in Example 5.4.2 on page 124 may be given by the input file shown in Figure 6.7. The number of places, here 6, is given in the first line and places are separately described in the 6 following lines. Such a line starts with a type of some predefined labeled event structure in Esu, and additional parameters come after this type. In this input file,

- '`BP b v`' stands for the b-bounded processes initialized by v, i.e. (b, v)-$\mathcal{BP}$ (see Section 3.3.2);
- '`P1 v`' stands for our labeled event structure for a place that is bounded by 1 and has initially v token. Recall that v is either 0 or 1, and '`P1 1`' corresponds to the labeled event tree of $\mathcal{P}1$ discussed above;
- '`KP k v`' stands for the (k, v)-causality processes, i.e. (k, v)-$\mathcal{CP}$ (see Section 3.3.2).

Although these 6 places are all 1-bounded, one may associate different types to a place and obtain finite prefixes having the same size. But it is not true when working with Petri nets that are not 1-safe. Some examples and comparisons will be shown in Section 6.3.2. Moreover, a good choice for modeling places sometimes avoids or reduces the redundancy in the generated prefix of the synchronized product. This phenomenon is detailed in the next section.

The last 6 lines in the input file (Figure 6.7) give the number of the Petri net's transitions and description of these transitions themselves line by line. Each transition, as usual, intuitively consists of component actions. Notice here that the "do nothing" action ε is represented by the character `N`.

6.2.2 Redundancy reduction

The advantage of using concurrent labeled event structures for components when unfolding is that the synchronized product not only exploits concurrency between components but also the intrinsic concurrency inside each component. As consequence, the constructing prefix is hopefully more compact. However, redundancy in the synchronized product may come from component events that are concurrent and have the same label at the same time. In this case, the generated prefix of the synchronized product is usually much bigger than necessary. This phenomenon is called the *auto-concurrency problem* [KK03].

Let us take an example in order to clarify this problem. Figure 6.8.a illustrates a bounded Petri net that has three places and three transitions. One can simply represents its places p_1, p_2, p_3 by using bounded processes as shown in Figure 6.8.b in the left-to-right order respectively. Hence, the Petri net is represented by the synchronized product, denoted by $\mathcal{E}_{S\mathcal{P}} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, of these bounded processes w.r.t. to the synchronization constraint $\Sigma = \{a, b, c\}$ where $a = \langle -, +, \varepsilon\rangle, b = \langle \varepsilon, +, -\rangle$, and $c = \langle +, -, \varepsilon\rangle$. Figure 6.8.c gives a prefix containing only 10 events of the synchronized product. Let us denote $S = \{s_1, s_2, s_3, s_4, s_5, s'_5, s_6, s'_6, s_7, s_8\}$, this prefix is accordingly represented by $\mathcal{E}_{S\mathcal{P}}|_S$. Recall that a global event in the synchronisation product is nothing but a synchronization of component events, hence, a global one may be illustrated by a closed curve that groups component ones as illustrated in Figure 6.8.b. For instance, the closed curve labeled s_2 means that $\mathcal{V}(s_2) = \langle e_1, f_2, \varepsilon\rangle$; while both global events s'_5, s'_6 have the same synchronization vector, i.e. $\mathcal{V}(s'_5) = \mathcal{V}(s'_6)$, and is represented by the same curve.

The bounded process $(2,0)$-$\mathcal{BP}$ corresponding to the place p_2 consists of two bounded process $(1,0)$-$\mathcal{BP}$ (see Definition 3.3.13 on page 38). Intuitively, one distinguishes tokens on p_2 so that there are concurrent events of the same label in $(2,0)$-$\mathcal{BP}$. For example, f_2 is somehow a copy of f_1 and vice versa. When computing the synchronized product by the unfolding technique, f_1 and f_2 give rises to two different global events labeled $a = \langle -, +, \varepsilon\rangle$ that are respectively s_1 and s_2. However, s_1 is in conflict with s_2 because they correspond to a same event e_1 in the first component (see Definition 3.3.39 on page 54). As a consequence, the empty configuration $\emptyset = (>(s_1)) = (>(s_2))$ has two extensions s_1, s_2, i.e. $\emptyset \vdash s_1$ and $\emptyset \vdash s_2$, that satisfy that $\mathcal{L}(s_1) = \mathcal{L}(s_2)$ and $s_1 \# s_2$. It means that the synchronized product is redundant by Definition 3.2.14 on page 32. Once again, s_2 is a copy of s_1 so that the successors of s_2 are just redundant duplication of the ones of s_1. All configurations containing s_2 as well as successors of s_2 may be removed in the global labeled event structure without loss of information up to isomorphism.

Let us use the same notation of isomorphism in Definition 3.2.7 on page 29 for configurations and events. We say that two configurations C and C' of a labeled event structure $\mathcal{E}$ are isomorphic and write $C \approx C'$ if the two prefixes $\mathcal{E}|_C$ and $\mathcal{E}|_{C'}$ are isomorphic, i.e. $\mathcal{E}|_C \approx \mathcal{E}|_{C'}$. By a same manner, two events e, e' are isomorphic, denoted by $e \approx e'$ if their local configurations are isomorphic, i.e. $\geq(e) \approx \geq(e')$. In the prefix $\mathcal{E}_{S\mathcal{P}}|_S$, we have 5 pairs of isomorphic events: $s_1 \approx s_2$, $s_3 \approx s_4$, $s_5 \approx s_6$, $s_7 \approx s_8$, and $s'_5 \approx s'_6$. As discussed above, s_2 may be intuitively removed from the prefix $\mathcal{E}_{S\mathcal{P}}|_S$ as well as the whole labeled event structure $\mathcal{E}_{S\mathcal{P}}$ and all configurations are still preserved, due to isomorphism, in $\mathcal{E}'_{S\mathcal{P}} = \mathcal{E}_{S\mathcal{P}}|_{E\setminus(\leq(s_2))}$. Formally, for all configurations C of $\mathcal{E}_{S\mathcal{P}}$, there

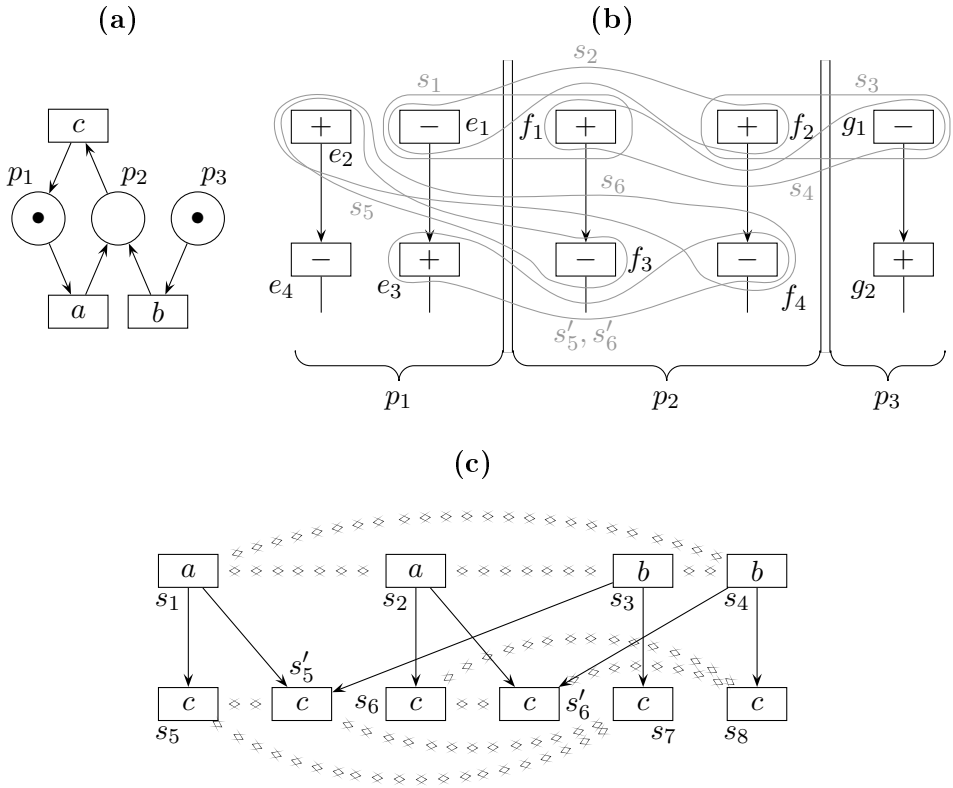


Figure 6.8: Redundancy illustration: (a) a bounded Petri net, (b) bounded processes modeling its three places, and (c) a prefix of the synchronized product of these bounded processes w.r.t. the Petri net's transitions.

exits a configuration C' of $\mathcal{E}'_{\mathcal{SP}}$ such that $C \approx C'$. Therefore, in order to verify decidable problems given in Section 4.3, one intuitively prefers the compact prefix $\mathcal{E}_{\mathcal{SP}}|_{S'}$ where $S' = \{s_1, s_3, s_5, s'_5, s_7\}$ than the prefix $\mathcal{E}_{\mathcal{SP}}|_S$.

Moreover, the prefix $\mathcal{E}_{\mathcal{SP}}|_{S'}$ still contains redundancy because of the extensions s_5 and s'_5 of the configuration $\{s_1, s_3\}$. Recall that the linearisations of events' labels in $\mathcal{E}_{\mathcal{SP}}$ correspond to firing sequences of the induced labeled transition system of $\mathcal{E}_{\mathcal{SP}}$. Consider now the two configurations $\{s_1, s_3, s_5\}$ and $\{s_1, s_3, s'_5\}$. The first one give rise to label linearisations $\mathcal{L}^{\mathcal{W}}(\{s_1.s_3.s_5, s_1.s_5.s_3, s_3.s_1.s_5\}) = \{abc, acb, bac\}$ while the second one corresponds only to label linearisations $\mathcal{L}^{\mathcal{W}}(\{s_1.s_3.s'_5, s_3.s_1.s'_5\}) = \{abc, bac\}$. Therefore, in order to reduce redundancy, one would rather keep s_5 than s'_5 because one will loose the label linearisation acb when removing s_5. The prefix $\mathcal{E}_{\mathcal{SP}}|_{\{s_1,s_3,s_5\}}$ is intuitively more compact than the one $\mathcal{E}_{\mathcal{SP}}|_{\{s_1,s_3,s'_5\}}$. These prefixes differ only on whether the causality between events labeled a and c exists. We say that $\mathcal{E}_{\mathcal{SP}}|_{\{s_1,s_3,s'_5\}}$ is a *sub-linearisation* of $\mathcal{E}_{\mathcal{SP}}|_{\{s_1,s_3,s_5\}}$.

Definition 6.2.1. Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ and $\mathcal{E}' = (E', \leq', \#', \mathcal{L}', \mathcal{M}')$ be two labeled event structures. We say that $\mathcal{E}$ is a *sub-linearisation* of $\mathcal{E}'$ and write $\mathcal{E} \lesssim \mathcal{E}'$ if $\mathcal{E}$ is isomorphic with some labeled event structure $(E', \leq'', \#', \mathcal{L}', \mathcal{M}')$ where the relation $\leq''$ is an extension of the causality $\leq'$, i.e. $(\leq') \subseteq (\leq'')$.

A configuration C is a *sub-linearisation* of another one C', denoted by $C \lesssim C'$, if the C-prefix $\mathcal{E}|_C$ is a sub-linearisation of the C'-prefix $\mathcal{E}|_{C'}$.

Let us return to the idea of our technique for reducing redundancy. That is, given a labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, trying to remove some event as well as

its successors while preserving label linearisations of $\mathcal{E}$. Such an event r is called an *useless event* w.r.t. $\mathcal{E}$. Then, one may continue by removing another useless event r' w.r.t. $\mathcal{E}|_{E\setminus\leq(r)}$, and so on. The final labeled event structure as well as all intermediate ones, denoted by $\mathcal{E}'$, must satisfy that, for all configurations $C \in \mathcal{C}_{\mathcal{E}}$, there exists a configuration $C' \in \mathcal{C}_{\mathcal{E}'}$ such that $C \lesssim C'$. The obtained labeled event structure $\mathcal{E}'$ is much more compact than $\mathcal{E}$, and has possibly no redundancy. In practice, when $\mathcal{E}$ is constructed using the unfolding technique, successors of an useless r event may be avoided by not extending r.

Notice that $\mathcal{C}_{\mathcal{E}|_{E\setminus\leq(r)}} = \mathcal{C}_{\mathcal{E}} \setminus \{C \in \mathcal{C}_{\mathcal{E}} \,/\, r \in C\}$, hence an event r is useless if for all $C \in \mathcal{C}_{\mathcal{E}}$ satisfying $r \in C$, there exists a configuration $C' \in \mathcal{C}_{\mathcal{E}}$ such that $r \notin C'$ and $C \lesssim C'$. This condition is usually guaranteed by the existence of another event $e \in E$ that is, for instance, isomorphic and in conflict with r. In the prefix $\mathcal{E}_{\mathcal{SP}}|_S$ above, one can take $r = s_1, e = s_2$ as an example. Formally,

$$\mathcal{C}_{\mathcal{E}} = \{C \in \mathcal{C}_{\mathcal{E}} \,/\, e, r \notin C\} \cup \{C \in \mathcal{C}_{\mathcal{E}} \,/\, e \in C\} \cup \{C \in \mathcal{C}_{\mathcal{E}} \,/\, r \in C\}$$

It follows from the conflict between e and r that they can not be found in any given configuration. As a consequence, the three subsets above are pairwise disjoint. In order to determine whether r is useless, one needs to verify if configurations in $\{C \in \mathcal{C}_{\mathcal{E}} \,/\, r \in C\}$ are sub-linearisations of configurations in $\{C \in \mathcal{C}_{\mathcal{E}} \,/\, e \in C\}$. We will show that this condition is guaranteed in coherent labeled event structures and one does not have to compute the set of the configurations containing r as well as the ones containing e that are usually infinite.

However, when r is useless due to e, in general, e is also useless due to r. The difficult point here is to define which event to remove. In order to integrate our technique for reducing redundancy into the unfolding technique, we naturally use the total order $\trianglelefteq$ in which events are inserted into or extended in the constructing prefix. This order $\trianglelefteq$ is a linear extension of the causality $\leq$ and is useful to break the symmetry of the notion 'useless'. Let us return back to the example in Figure 6.8. Suppose that s_2 is computed after s_1, i.e. $s_1 \lhd s_2$, one can simply notice that s_2 is redundant and will not extend s_2. And as a consequence, the obtained prefix does not contain successors of s_2. However, it is worth noticing that useless events can not be independently removed. Because, a naive solution such as removing both s_2 and s_3 due the existence of s_1 and s_4 satisfying, for instance $s_1 \lhd s_2$ and $s_4 \lhd s_3$, may result in losing some label linearisations. In such a case, the prefix $\mathcal{E}|_{\{s_1,s_4\}}$ as well as $\mathcal{E}|_{E\setminus(\leq(\{s_2,s_3\}))}$ contains no configuration C such that either the configuration $\{s_1, s_3\}$ or the configuration $\{s_2, s_4\}$ is its sub-linearisation. As a consequence, the label linearisations *ab* and *ba* that are firing sequences of the induced labeled transition system are not preserved. The reason for this 'counter-example' is that, after removing s_2, in the prefix $\mathcal{E}|_{E\setminus(\leq(s_2))}$, s_4 is useless due to s_3 but not in the reverse sense. Therefore, as stated in the following definition, the determination of a useless event not only depends on another event e, but is also based on some E'-prefix containing e. As we will see later, this E'-prefix is the constructing prefix manipulated by the unfolding algorithm and does not contain any other useless event.

Definition 6.2.2 ($\trianglelefteq$-redundance)**.** Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a deterministic labeled event structure $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$, and let $\trianglelefteq$ be a linear extension of the causality $\leq$. Given a downward-closed set of events $E' \subseteq E$, let us denote $\mathcal{E}'$ the E'-prefix of $\mathcal{E}$. An event $r \in E$ is $\trianglelefteq$-*redundant* w.r.t. $\mathcal{E}'$ if there exists another event $e \in E'$ such that

1. $e \lhd r$,
2. $\mathcal{L}(e) = \mathcal{L}(r)$,

3. $e \# r$,
4. $(>(e)) \subseteq (>(r)) \subseteq E'$, and
5. for all $f \in (\triangleright(r) \cap E')$, $f \# e$ implies $f \# r$.

For simplicity, Definition 6.2.2 concerns only deterministic labeled event structures such that their induced labeled transition systems are also deterministic (see Lemma 3.2.5 on page 29). For non-deterministic and well-preordered ones, conditions for the marking function $\mathcal{M}$ are needed. This is a subject of future work.

The advantages of applying $\trianglelefteq$-redundance definition in practice come from its simplicity. As mentioned above, since $\trianglelefteq$ is the insertion order or extending order of events in the constructed prefix, one has to look for r only on the part of the labeled event structure that has already been built, formally represented by the E'-prefix. The second condition is easy to verify while the third and the fifth conditions take only the conflict relation into account. So there is no need to compute global configurations. This is in line with the partial-order idea of the unfolding technique. The fourth condition of $\trianglelefteq$-redundant event does not mean that e and r are isomorphic. This restriction reduces somehow the number of useless events that may be defined as $\trianglelefteq$-redundant in our technique (see Section 6.3.2). We restrict to two particular cases that are when $(>(r)) = (>(e))$ and when $(>(r)) \supset (>(e))$. These two disjoint cases could be found in the example in Figure 6.8. For instance, $e = s_1, r = s_2$ and $(>(s_1)) = (>(s_2)) = \emptyset$; or $e = s_5, r = s'_5$ and $(>(s_5)) = \{s_1\} \subset \{s_1, s_3\} = (>(s'_5))$.

The main idea of $\trianglelefteq$-redundant events is that they are useless. As in the truncation technique (see Chapter 4), they form somehow a frontier between their successors and the other events, called *non-$\trianglelefteq$-redundant* events. By keeping only non-$\trianglelefteq$-redundant events, one obtains a compact prefix that preserves needed information for verification and may be formally defined as follows:

Definition 6.2.3. Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a deterministic labeled event structure and let $\trianglelefteq$ be a linear extension of the causality $\leq$. An E'-prefix $\mathcal{E}'$ of $\mathcal{E}$, where E' is a downward-closed subset of E w.r.t. $\leq$, is called a *prefix without $\trianglelefteq$-redundant event* if

- for all $e \in E'$, e is not $\trianglelefteq$-redundant w.r.t. $\mathcal{E}'$, and
- for all $e \in \mathsf{Min}_{\leq}(E \setminus E')$, e is $\trianglelefteq$-redundant w.r.t. $\mathcal{E}'$.

Lemma 6.2.4. *Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a deterministic labeled event structure and let $\trianglelefteq$ be a linear extension of the causality $\leq$. $\mathcal{E}$ has an unique prefix without $\trianglelefteq$-redundant event.*

Proof. We first prove by contradiction the uniqueness of the prefix without $\trianglelefteq$-redundant event. Suppose that there exists two different prefixes without $\trianglelefteq$-redundant event $\mathcal{E}'$ and $\mathcal{E}''$. Let E' and E'' respectively denote their sets of events. Observe that E' and E'' are downward closed w.r.t. $\leq$, moreover, since these sets are not the same, the set $X = (E' \setminus E'') \cup (E'' \setminus E')$ is not empty. Let r be the minimal event of X w.r.t. the total order $\trianglelefteq$. Without loss of generality, assume that $r \in E'$, and consequently, $r \notin E''$. It follows from the choice of minimal event r that $(>(r))$ is a subset of both E' and E'', and at the same time, $(\triangleright(r)) \cap E' = (\triangleright(r)) \cap E''$. Hence, r is $\trianglelefteq$-redundant w.r.t. $\mathcal{E}''$ because $r \in \mathsf{Min}_{\leq}(E \setminus E'')$, and more precisely, it is due to some event $e \in (\triangleright(r) \cap E'')$. Event r is therefore also $\trianglelefteq$-redundant w.r.t. $\mathcal{E}'$ because $e \in E'$. This contradicts to the fact that $r \in E'$. Therefore, we have $E' = E''$ and the two prefixes $\mathcal{E}'$ and $\mathcal{E}''$ are the same.

Now, we will prove the existence of a prefix of $\mathcal{E}$ without $\trianglelefteq$-redundant event. Let $\mathcal{P}$ be the set of E'-prefixes of E, here $E' \subseteq E$, satisfying that for all events $e \in \mathsf{Min}_{\leq}(E \setminus E')$:

- e is $\trianglelefteq$-redundant w.r.t. the E'-prefix of $\mathcal{E}$ [1], and
- for every $\trianglelefteq$-redundant event $f \in E'$ w.r.t. the E'-prefix of $\mathcal{E}$, $e \lhd f$.

$\mathcal{P}$ is not empty because it contains, of course, $\mathcal{E}$. Notice that the set of all prefixes of $\mathcal{E}$ is partially ordered w.r.t. the inclusion order over their event sets, and moreover, every totally ordered subset of them admits a greatest lower bound. It is straightforward that $\mathcal{P}$ is too. Let us define a function $\mathcal{F}$ from $\mathcal{P}$ to the set of prefixes of $\mathcal{E}$ as follows: for every prefix $\mathcal{E}' \in \mathcal{P}$, let E' denote its event set,

$$\mathcal{F}(\mathcal{E}') = \begin{cases} \mathcal{E}' \text{ if } \mathcal{E}' \text{ contains no } \trianglelefteq\text{-redundant event w.r.t. } \mathcal{E}' \\ \text{the } (E' \setminus (\leq(f)))\text{-prefix of } \mathcal{E}', \text{ where} \\ \qquad f = \mathsf{Min}_{\trianglelefteq}\{r \in E' \,/\, r \text{ is } \trianglelefteq\text{-redundant w.r.t. } \mathcal{E}'\}, \text{ otherwise.} \end{cases}$$

It directly follows from the second property of prefixes in $\mathcal{P}$ and the choice of the minimal $\trianglelefteq$-redundant event f in the definition of $\mathcal{F}$ that $\mathcal{F}(\mathcal{E}') \in \mathcal{P}$ for all $\mathcal{E}' \in \mathcal{P}$, i.e. $\mathcal{F} : \mathcal{P} \to \mathcal{P}$. Moreover, $\mathcal{F}(\mathcal{E}')$ is a prefix of $\mathcal{E}'$ and hence is always smaller than or equal to $\mathcal{E}'$ w.r.t. the inclusion order over event sets. Therefore, $\mathcal{F}$ has a fixed point. It means that there exists a prefix $\mathcal{E}' \in \mathcal{P}$ satisfying $\mathcal{F}(\mathcal{E}') = \mathcal{E}'$. Once again, by definition of $\mathcal{F}$ and the set $\mathcal{P}$, one can deduce that this prefix $\mathcal{E}'$ is a prefix without $\trianglelefteq$-redundant event by Definition 6.2.3. □

Let $NR_{(\mathcal{E},\trianglelefteq)}$ denote the event set of the prefix without $\trianglelefteq$-redundant event of $\mathcal{E}$. This set is downward-closed w.r.t. the causality order $\leq$. The $NR_{(\mathcal{E},\trianglelefteq)}$-prefix of $\mathcal{E}$ may be generated by an algorithm that is slightly different from the unfolding one (Algorithm 5.1 on page 89) as below.

Algorithm 6.1: Unfolding algorithm with redundancy reduction

```
1  begin
2      (Ê, PE) := Init()
3      while PE ≠ ∅ do
4          take an event e in PE
5          if isRedundant(Ê, PE, e) then
6              PE := PE \ {e}
7              Ê := RemoveEvent(Ê, e)
8          else
9              (Ê, PE) := Extend(Ê, PE, e)
10         end if
11     end while
12 end
```

In Algorithm 6.1, there are two additional functions: isRedundant and RemoveEvent. The first one is an implementation of Definition 6.2.2 while the second one simply removes a $\trianglelefteq$-redundant event e from both extension set PE and the constructed prefix $\widehat{\mathcal{E}}$. In this way, $\trianglelefteq$-redundant events are removed and will not be extended. As a consequence, the prefix $\widehat{\mathcal{E}}$ at the end of the main loop (lines 3-11) is exactly the $NR_{(\mathcal{E},\trianglelefteq)}$-prefix of $\mathcal{E}$. In the implementation of this algorithm, we simply define $\trianglelefteq$ as the extending order that depends on the choice of event in line 4. Hence, it is worth noticing that when calling

[1] This corresponds to the second item in Definition 6.2.3.

$\mathsf{isRedundant}(\widehat{\mathcal{E}}, \mathsf{PE}, e)$, e is the maximal event, w.r.t. $\trianglelefteq$, in $(\widehat{E} \setminus \mathsf{PE}) \cup \{e\}$, and at the same time, is the minimal event, w.r.t. $\trianglelefteq$, in PE. So that, for the fifth property of $\trianglelefteq$-redundant event in Definition 6.2.2, we have $(\triangleright(e) \cap E) = (\widehat{E} \setminus \mathsf{PE})$.

Let us give some more details on the function isRedundant in Algorithm 6.2. The set X in line 2 contains all events satisfying the first and fourth properties in Definition 6.2.2. The loop's condition at lines 4 restricts to events e' in X that has a same label as e. It respects the second property. The third and the last ones are handled by the test in line 5. The function terminates and returns true whenever an event e' satisfies this test. Otherwise, it finally returns false (line 9).

Algorithm 6.2: Function isRedundant determines whether e is $\trianglelefteq$-redundant by Definition 6.2.2 where $\trianglelefteq$ is the order of extending events in the unfolding algorithm by default.

```
1  function isRedundant(Ê, PE, e)
2  begin
3      X := {e' ∈ (Ê \ PE) / (≻̂(e)) ⊆ (≻̂(e'))}
4      for each e' ∈ {e' ∈ X / L̂(e') = L̂(e)} do
5          if e' #̂ e and (#̂(e') \ PE) ⊆ (#̂(e) \ PE) then
6              return true
7          end if
8      end for
9      return false
10 end
```

Proposition 6.2.5. *Let $\mathcal{E} = (E, \leq, \#, \mathcal{L}, \mathcal{M})$ be a deterministic, coherent, and finitely-branching labeled event structure and let $\trianglelefteq$ be a linear extension of the causality $\leq$. For every configuration $C \in \mathcal{C}_\mathcal{E}$, there exists a configuration $C' \in \mathcal{C}_\mathcal{E}$ that contains no $\trianglelefteq$-redundant event, i.e. $C' \subseteq NR_{(\mathcal{E},\trianglelefteq)}$, and $\mathcal{M}(C) = \mathcal{M}(C')$.*

Proof. Let us define an order $\trianglelefteq_\mathcal{C}$ on $\mathcal{C}_\mathcal{E}$ by: for all configuration $C, C' \in \mathcal{C}_\mathcal{E}$, let l and l' be respectively the linearisations of C and C' w.r.t. the total order $\trianglelefteq$, $C \trianglelefteq_\mathcal{C} C'$ if l is lexicographically smaller than or equal to l' w.r.t. $\trianglelefteq$. It follows from the important property of lexicographical orders that the well-foundedness and totality of $\trianglelefteq$ are preserved. It means that $\trianglelefteq_\mathcal{C}$ is also a well-founded and total order over $\mathcal{C}_\mathcal{E}$.

Therefore, for every configuration $C \in \mathcal{C}_\mathcal{E}$, the set $\{C' \in \mathcal{C}_\mathcal{E} \,/\, \mathcal{M}(C') = \mathcal{M}(C)\}$ is well-founded and admits a minimal configuration w.r.t. $\trianglelefteq_\mathcal{C}$. We will prove that the minimal configuration C_m w.r.t. $\trianglelefteq_\mathcal{C}$ of the set $\{C' \in \mathcal{C}(\mathcal{E}) \,/\, \mathcal{M}(C') = \mathcal{M}(C)\}$ contains no $\trianglelefteq$-redundant event, so that $C_m \subseteq NR_{(\mathcal{E},\trianglelefteq)}$.

Let us prove this by contradiction, i.e. assume that $C_m \not\subseteq NR_{(\mathcal{E},\trianglelefteq)}$, and let $r = \mathsf{Min}_\trianglelefteq(C_m \setminus NR_{(\mathcal{E},\trianglelefteq)})$. Because $\trianglelefteq$ is a linear extension of the causality $\leq$, for all $e \in (>(r))$, we have that $e \lhd r$, and consequently, $(>(r)) \subseteq NR_{(\mathcal{E},\trianglelefteq)}$. Hence $r \in \mathsf{Min}_\leq(E \setminus NR_{(\mathcal{E},\trianglelefteq)})$ and we get from Definition 6.2.3 that r is a $\trianglelefteq$-redundant event w.r.t. the $NR_{(\mathcal{E},\trianglelefteq)}$-prefix of $\mathcal{E}$ and to some event $e \in NR_{(\mathcal{E},\trianglelefteq)}$. Let us denote $C_- = (\triangleright(r) \cap C_m)$ and $C_+ = (\lhd(r) \cap C_m)$. We have thus C_- and C_+ are disjoint and $C_m = C_- \cup C_+ \cup \{r\}$. Moreover, it holds that $C_- \subseteq NR_{(\mathcal{E},\trianglelefteq)}$.

It follows from the definition of $\trianglelefteq$-redundant event (Definition 6.2.2) that:

1. $e \lhd r$,

2. $\mathcal{L}(e) = \mathcal{L}(r)$.

3. $e \# r$. That implies $e \notin C_m$ and hence $e \notin C_-$.

4. $(>(e)) \subseteq (>(r))$, and consequently, $(>(e)) \subseteq C_-$. Because, on the one hand, of the downward-closed property, w.r.t. the causality $\leq$, of C_m that $(>(r)) \subseteq C_m$, and on the other hand, of the linear extension $\trianglelefteq$ of $\leq$ that $(>(r)) \subseteq (\rhd(r))$.

5. for all $f \in (\rhd(r) \cap NR_{(\mathcal{E},\trianglelefteq)})$, $f \# e$ implies $f \# r$. Observe that $C_- \subseteq (\rhd(r) \cap NR_{(\mathcal{E},\trianglelefteq)})$. Since $(C_- \cup \{r\}) \subseteq C_m$, we get that r is in conflict with no event in C_-. Therefore, e is in conflict with no event $f \in C_-$.

By combining the last three properties above, one obtains that e is an extension event of C_-, i.e. $C_- \vdash e$. More precisely, $(C_- \cup \{r\}) \lesssim (C_- \cup \{e\})$. Due to the restriction that the induced labeled transition system is deterministic, $\mathcal{M}(C_- \cup \{r\}) = \mathcal{M}(C_- \cup \{e\})$. Thanks to Lemma 3.2.10 on page 30, it follows from the coherency of $\mathcal{E}$ that the two suffixes of $\mathcal{E}$ based on the configurations $(C_- \cup \{r\})$ and $(C_- \cup \{e\})$ give rise to the same set of markings. In other words, there exists a configuration $C'_m \in \mathcal{C}_\mathcal{E}$ such that $(C_- \cup \{e\}) \subseteq C'_m$ and $\mathcal{M}(C_m) = \mathcal{M}(C'_m)$.

Since $f \rhd r \rhd e$ for all $f \in C_+$, we have thus $C'_m \lhd_\mathcal{C} C_m$ by definition. This contradicts to the minimality of C_m. Therefore, the assumption above is not true, and consequently C_m contains no $\trianglelefteq$-redundant event. □

A consequence of Proposition 6.2.5 is that the $NR_{(\mathcal{E},\trianglelefteq)}$-prefix of $\mathcal{E}$ preserves information for verifying problems based on coverability (see Section 4.3.2) on $\mathcal{E}$. This technique for reducing redundancy is well adapted to the truncating technique (see Chapter 4) when $\trianglelefteq_\mathcal{C}$ is a linear extension of the adequate order over configurations. One only needs to compute the prefix $\mathcal{E}|_{NR_{(\mathcal{E},\trianglelefteq)}}$ of $\mathcal{E}$ that contains no $\trianglelefteq$-redundant event and truncate it afterward. The final prefix preserves all markings of $\mathcal{E}$. Intuitively, as shown in the proof of Theorem 4.2.14 as well as of Proposition 6.2.5, the key here is that marking of a configuration C is preserved by another configuration $C' \lhd_\mathcal{C} C$ whenever C contains a cutoff event or a $\trianglelefteq$-redundant event. Thanks to the well-foundedness of $\trianglelefteq_\mathcal{C}$ and the adequate order, it gives rise to a configuration in the prefix that contains neither a $\trianglelefteq$-redundant event nor a cutoff event.

Recall that the adequate order used in definition of cut-off events forces that successors of a cut-off event are also cut-off ones. Hence, cut-off events form an upward-closed set w.r.t. the causality. We also aim at giving another definition of $\trianglelefteq$-redundancy based on some improved order $\trianglelefteq$ satisfying the adequate property. In such a case, successors of $\trianglelefteq$-redundant events are also $\trianglelefteq$-redundant events, and that may make our technique for reducing redundancy more efficient. However, the existence of such a total order $\trianglelefteq$, said *total adequate order*, is still an open problem for truncating technique as stated in [ERV96].

It is worth noticing that our technique for reducing redundancy improves substantially the truncating technique. Because in many cases, $\trianglelefteq$-redundant events may not be seen as cut-off events whatever adequate order is used (see Section 4.2.3). However, they may be safely removed when considering the conflict relation in addition as in Definition 6.2.2. For example, as illustrated in Figure 6.8, two isomorphic events s_1 and s_2 can not be cut-off events, but one of these events may be removed because of the $\trianglelefteq$-redundancy. Experimental results may be found in Section 6.3.2.

Let us return to Definition 6.2.2. If one modifies it so that an event r is $\trianglelefteq$-redundant event w.r.t. a configuration $C \subseteq (\rhd(r) \cap NR_{(\mathcal{E},\trianglelefteq)})$ in the place of another event e, then

Proposition 6.2.5 still holds. Notice that the local configuration $\geq(r)$ must be a sub-linearisation of such a configuration C (see Definition 6.2.1) and r must be in conflict with some event in C. This idea coincides with the one of truncating technique used in Pep. In other words, like cutting contexts for cutoff events, $\trianglelefteq$-redundant property of events may based on the purely local cutting context or some arbitrary one (see Section 4.2). However, in Definition 6.2.2, we restrict to a simple case where $C = ((>(r)) \vdash e)$ for some event e satisfying $\mathcal{L}(e) = \mathcal{L}(r)$ and $e \# r$. This avoids to compute the configurations in $\mathcal{E}|_{\rhd(r) \cap NR_{(\mathcal{E},\trianglelefteq)}}$ when determining whether an event r is $\trianglelefteq$-redundant, and respects well the partial-order idea.

Last but not least, there is a challenge to go further by giving some redundant criterion based on the global cutting context. That means a configuration is somehow useless due to another one so that we can remove redundant events while keeping all configurations by means of isomorphism or the sub-linearisation relation (Definition 6.2.1). As seen in the proof of Proposition 6.2.5, when a configuration C contains a $\trianglelefteq$-redundant event r w.r.t. another event e, the configuration $(C_- \cup \{r\}) = (\rhd(r) \cap NR_{(\mathcal{E},\trianglelefteq)}) \cup \{r\}$ is a sub-linearisation of the configuration $(C_- \cup \{e\})$. One may hope that C is a sub-linearisation of another configuration C' such that $(C_- \cup \{e\}) \subseteq C'$ afterward. Formally, if $C \lesssim D$ then for every configuration C' extended from C, i.e. $C \Vdash C'$, there exists a configuration D' extended from D, i.e. $D \Vdash D'$ such that $C' \lesssim D'$ (*). However, it is not true. The reason is that the sub-linearisation relation in Definition 6.2.1 is not preserved in general w.r.t. the extension relation $\Vdash$ (see Section 3.1.2).

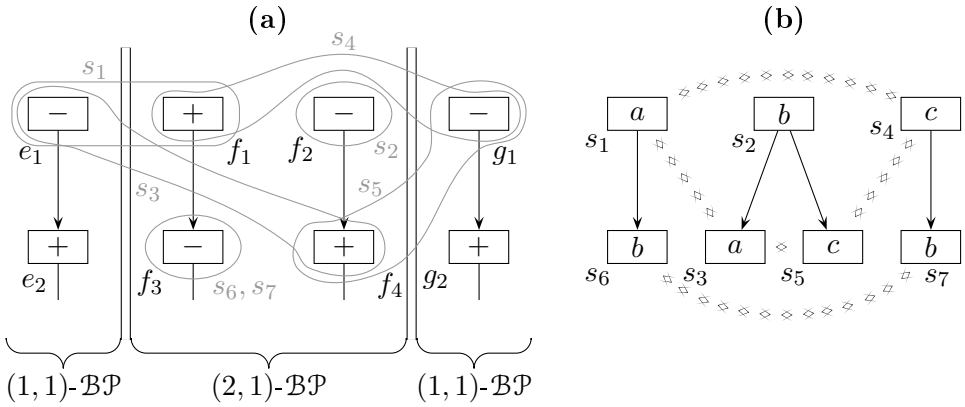


Figure 6.9: Sub-linearisation relation over configurations is not preserved by the extension relation: (a) the three components, and (b) a prefix of the synchronized product w.r.t. the synchronization $\Sigma = \{a, b, c\}$ where $a = \langle -, +, \varepsilon \rangle, b = \langle \varepsilon, -, \varepsilon \rangle, c = \langle \varepsilon, +, - \rangle$.

Figure 6.9 gives an counter-example of the statement (*) above as a case study for our future work. In the prefix of the synchronization product of the three components $(1,1)$-$\mathcal{BP}$, $(2,1)$-$\mathcal{BP}$, and $(1,1)$-$\mathcal{BP}$, the configuration $C = \{s_2, s_3\}$ is a sub-linearisation of the configuration $D = \{s_1, s_2\}$, i.e. $C \lesssim D$. Because there is no causality between s_1 and s_2. Intuitively, the first configuration corresponds only to the label linearisation ba while the second one corresponds to both the label linearisations ab and ba. Consider now the configuration $C' = \{s_2, s_3, s_4\}$ that is extended from the configuration C, i.e. $C \Vdash C'$. This configuration C' gives two label linearisations that are bac and cba. However, one can not find any configuration that contains both s_1 and s_2, and gives at least the same label linearisations as the configuration C' at the same time. Formally, there is no configuration D' such that $D \Vdash D'$, i.e. $\{s_1, s_2\} \subseteq D'$, and $C' \lesssim D'$.

This counter-example intuitively shows the reason that the $NR_{(\mathcal{E},\trianglelefteq)}$-prefix of $\mathcal{E}$ only preserves markings of $\mathcal{E}$ as stated in Proposition 6.2.5, and not its label linearisations, or in other words, not the firing sequences of the induced labeled transitions system $\mathcal{LTS}^{\mathcal{E}}$ of $\mathcal{E}$. Suppose here that $s_1 \lhd s_2 \lhd s_3 \lhd s_4 \lhd s_5 \lhd s_6 \lhd s_7$. When applying our technique, the obtained prefix will not contain the $\trianglelefteq$-redundant event s_3 due to the existence of s_1. In other words, the label linearisation *cba* will not be generated from such a prefix. But this is not a problem for verification of reachability-based properties.

Our technique for reducing redundancy discussed above is implemented in Esu and some experimental results will be shown in Section 6.3.2.

6.3 Experiment results on Petri nets

In order to evaluate the benefits of our approach we have experimented Esu on some well-known examples and compared with two tools for Petri nets: the Pep environment which provides an unfolding tool for bounded Petri nets [GB96, pep], and Tina which analyzes arbitrary Petri nets using structural analysis techniques and forward Karp-Miller reachability analysis [BRV04, tin]. The execution times in our experimental results are obtained on an Intel(R) Pentium(TM) 1.2GHz, with 1GB memory.

6.3.1 1-safe Petri nets

Table 6.3 shows our experimental results as well as the one obtained by using PEP tools [GB96, pep] on various one-safe Petri nets. These benchmark examples are collected by Corbett, McMillan, Melzer, Merkel and Römer, and detailed description can be found in [Kho03, Cor96, MR97]. In the table, the columns S and T respectively refer to the number of places and the number of transitions of the Petri nets; while the columns E and E_{cf} represent the numbers of events and of cutoff-events of the truncation, respectively. The last column named T(s) gives the execution time in seconds. When the truncation may not be computed within 1 minute, we mark the execution time by $-$.

Table 6.3: Experimental results on one-safe Petri nets.

			Pep			Esu		
Problem (size)	S	T	E	E_{cf}	T(s)	E	E_{cf}	T(s)
Cyclic (3)	23	17	23	4	0.00	23	4	0.00
Cyclic (6)	47	35	50	7	0.00	50	7	0.02
Cyclic (9)	71	53	77	10	0.00	77	10	0.06
Cyclic (12)	95	71	104	13	0.00	104	13	0.11
DAC (6)	42	34	53	0	0.00	53	0	0.00
DAC (9)	63	52	95	0	0.00	95	0	0.02
DAC (12)	84	70	146	0	0.00	146	0	0.06
DAC (15)	105	88	205	0	0.01	206	0	0.12
DME (2)	135	98	122	4	0.01	122	4	0.15
DME (3)	202	147	321	9	0.06	321	9	0.62
DME (4)	269	196	652	16	0.18	652	16	2.17
DME (5)	336	245	1145	25	0.51	–	–	–
DP (6)	36	24	96	30	0.00	96	30	0.02
DP (8)	48	32	176	56	0.01	176	56	0.05
DP (10)	60	40	280	90	0.01	280	90	0.12
DP (12)	72	48	408	132	0.02	408	132	0.22

			Pep			Esu		
Problem (size)	S	T	E	E_{cf}	T(s)	E	E_{cf}	T(s)
DPD (4)	36	36	296	81	0.01	296	81	0.11
DPD (5)	45	45	790	211	0.06	790	211	0.58
DPD (6)	54	54	1892	499	0.34	1892	499	3.32
DPD (7)	63	63	4314	1129	3.14	–	–	–
DPFM (2)	7	5	5	2	0.00	5	2	0.00
DPFM (5)	27	41	31	20	0.00	31	20	0.00
DPFM (8)	87	321	209	162	0.00	209	162	0.06
DPH (4)	39	46	336	117	0.01	533	207	0.25
DPH (5)	48	67	1351	547	0.13	2949	1389	5.83
DPH (6)	57	92	7231	3377	6.90	–	–	–
Elevator (1)	63	99	157	59	0.00	157	59	0.07
Elevator (2)	146	299	*15*	*0*	0.00	827	331	1.19
Elevator (3)	327	783	3895	1629	1.21	3895	1629	36.79
Furnace (1)	27	37	326	189	0.21	394	235	0.09
Furnace (2)	40	65	3110	1989	1.21	4980	3331	7.83
Furnace (3)	53	99	20759	13826	28.19	–	–	–
GasNQ (2)	71	85	164	45	0.01	169	46	0.08
GasNQ (3)	143	223	1191	399	0.10	1301	437	3.29
GasQ (1)	28	21	15	2	0.00	21	4	0.00
GasQ (2)	78	97	164	53	0.00	173	54	0.08
GasQ (3)	284	475	1262	486	0.10	1297	490	5.58
GasQ (4)	1428	2705	9853	3986	15.05	–	–	–
Hartstone (25)	127	77	102	1	0.00	102	1	0.07
Hartstone (50)	252	152	202	1	0.01	202	1	0.42
Hartstone (75)	377	227	302	1	0.04	302	1	1.36
Hartstone (100)	502	302	402	1	0.08	402	1	3.09
MMGT (1)	50	58	58	20	0.00	58	20	0.01
MMGT (2)	86	114	643	259	0.03	1178	493	2.00
Over (2)	33	32	35	8	0.00	41	10	0.01
Over (3)	52	53	187	53	0.00	296	81	0.18
Over (4)	71	74	807	243	0.05	1556	495	3.17
Over (5)	90	95	3846	1288	1.89	–	–	–
Ring (3)	39	33	47	11	0.00	47	11	0.01
Ring (5)	65	55	166	36	0.00	167	37	0.08
Ring (7)	91	77	403	79	0.02	403	79	0.32
Ring (9)	117	99	795	137	0.08	795	137	1.31
RW (6)	33	85	397	327	0.00	397	327	0.06
RW (9)	48	181	4627	4106	0.02	4627	4106	2.24
Sentest (25)	104	55	216	40	0.02	223	39	1.23
Sentest (50)	179	80	241	40	0.02	248	39	1.23
Sentest (75)	254	105	266	40	0.02	273	39	1.23
Sentest (100)	329	130	291	40	0.03	298	39	2.15

Since these Petri nets are all one-safe, we model most of them by synchronized products of 1-bounded processes. However, certain examples may not be presented based on bounded processes, and we simply choose the appropriate labeled event tree as discussed in Section 6.2.1. The examples' name are shown in italic in Table 6.3. For

this first implementation, our Esu tool has not much amelioration yet, the computation time is little slow when comparing with Pep. Observe that Pep has integrated some advanced techniques for the unfolding process, for instance, an improved structure of the queue of possible extension as well as an optimized routine for generating possible extensions in the unfolding algorithm. Hence, Pep can achieve significant speed up.

We use the Esparza and Römer's adequate order [ERV96] for determining the truncation. In many cases, Pep and Esu give truncations of the same size and the same number of cutoff events. However, it is worth noticing that the cutting context used in Pep differs from the local cutting context used in Esu. In Pep, a cut-off event is defined based on a configuration that may not be a local one. As a consequence, one can find out more cut-off events and the generated prefix is more compact. Experimental results indicates well this fact. For instance, Esu gives a truncation twice bigger than the one obtained by Pep on example 'Over (3)', and explodes on example 'Over (4)'. We have also observed that the version of Pep used in these experiments does not always produce correct results, for example, in the case of 'Elevator (2)'.

6.3.2 General bounded Petri nets

We have then tested Esu on some parameterized, concurrent and production systems that are modeled by Petri nets. Our case studies consist of

- Central Server Model (CSM) [MBC+95],
- Continuous Transportation (CTS) [MBC+95],
- Flexible Manufacturing System (FMS) [CM97],
- Kanban [CM97],
- Mutual Exclusion
- Multi poll [MC99], and
- Mesh 2x2 [MBC+95].

In Table 6.4, K defines the initial number of resources, i.e. number of tokens in parameterized places of these Petri nets, representing the systems; E (resp. E_{cf}, N, M) denotes the number of events in the truncation (resp. cutoff events, nodes in Tina's reachability tree, markings computed by Tina), and a '–' means that the analysis did not finish within 10 minutes.

As explained in Section 6.2.2, when unfolding Petri nets which are not one-safe, i.e. $K > 1$, the truncation may contains many redundant events due to the auto-concurrency problem. This redundancy does not have too much influence on CSM and Multi Poll because tokens obtained by redundant events will be separately unfolded. In other examples, e.g. FMS, Kanban or Mesh 2x2, since there are combinations of these tokens afterward, the size of constructed unfolding explodes very quickly.

Thanks to the technique for reducing redundancy implemented in Esu, one can observe that truncations computed by Esu are smaller than or equal to the ones computed by Pep, w.r.t. the number of events E. This redundancy is entirely eliminated on the Mutual Exclusion and the Swimming Pool. However, redundancy in the unfolding can not be avoided in other cases, e.g. Mesh and Kanban. It is worth noticing that the results in Table 6.4 are obtained while using the McMillan truncation technique. The Esparza, Römer and Vogler's one is more advantageous only on Mesh 2x2. By combining with our technique for reducing redundancy, for $K = 2$ in Mesh 2x2, Esu gives a truncation containing 2481 events of which 1280 events are cut-off, i.e. $|E| = 2481, |E_{cf}| = 1280$, after 4.58 seconds.

		Pep			Tina			Esu		
Example	K	E	E_{cf}	T(s)	M	N	T(s)	E	E_{cf}	T(s)
CSM	2	75	23	0.00	76	208	0.00	29	9	0.00
CSM	5	180	66	0.00	584	2264	0.00	64	20	0.02
CSM	10	605	231	0.02	3564	16224	0.06	121	37	0.08
CSM	40	8405	3321	5.33	183844	961684	12.13	456	132	5.61
FMS	1	81	19	0.00	120	345	0.00	32	7	0.00
FMS	2	26668	10204	84.35	3444	16311	0.06	585	124	0.54
FMS	3	–	–	–	48590	297382	2.84	–	–	–
Kanban	1	31	9	0.00	160	616	0.00	31	9	0.00
Kanban	2	58824	22946	575.47	4600	28120	0.10	8827	2127	44.83
Mutual Exclusion	5	120	100	0.00	3	4	0.00	4	2	0.00
Mutual Exclusion	10	440	400	0.00	3	4	0.00	4	2	0.00
Mutual Exclusion	40	6560	6400	0.10	3	4	0.00	4	2	0.00
Mesh 2x2	1	48	16	0.00	1881	7776	0.02	48	16	0.01
Mesh 2x2	2	–	–	–	200544	1325472	17.62	18968	11296	132.30
Multi Poll	2	123	48	0.00	11328	75241	0.56	155	48	0.04
Multi Poll	5	354	147	0.00	230664	1728412	30.06	191	48	0.10
Multi Poll	10	1019	432	0.02	–	–	–	211	48	0.22
Multi Poll	40	12359	5292	1.90	–	–	–	331	48	1.32
Swimming Pool	2	388	168	0.00	21	36	0.00	12	2	0.00
Swimming Pool	3	37593	18009	162.61	56	126	0.00	18	3	0.01
Swimming Pool	5	–	–	–	252	756	0.00	30	5	0.02
Swimming Pool	10	–	–	–	3003	12012	0.04	60	10	0.18
Swimming Pool	40	–	–	–	1221759	6516048	189.96	240	40	96.38

Table 6.4: Experimental results on some parameterized Petri nets.

Without the technique for reducing redundancy, the difference between the results of Pep, Tina and Esu comes from the choice of modeling Petri nets' places. Intuitively, when using Tina there is no concurrence between tokens of a same places, or in other words, a place is represented by an event tree. While using Pep, each place corresponds more or less to a K-bounded process. The unfolding of synchronized products of these bounded process, in examples of parameterized Petri nets here, do not really make use of the concurrency in bounded processes, but reversely, commits the auto-concurrency problem. For instance, on the Mutual Exclusion corresponding to a simple Petri net with 4 transitions and 5 places, Pep generates truncations that are approximately K^2 times bigger than necessary. Notice here that Esu uses 1-causality processes, i.e. $(1, v)$-$\mathcal{CP}$ where v is the initial number of tokens, in order to model these parameterized Petri nets.

	2-$\mathcal{CP}$			1-$\mathcal{CP}$			M-$\mathcal{CP}$ like			M-$\mathcal{CP}$ like (*)		
K	E	E_{cf}	T(s)	E	E_{cf}	T(s)	E	E_{cf}	T(s)	E	E_{cf}	T(s)
2	12	2	0.00	12	2	0.00	21	4	0.00	21	4	0.00
3	4136	2855	7.19	18	3	0.00	67	14	0.02	71	14	0.02
4	–	–	–	24	4	0.01	205	43	0.10	214	44	0.10
5	–	–	–	30	5	0.02	616	120	0.71	637	121	0.76
6	–	–	–	36	6	0.02	1872	324	5.90	1932	325	6.46
7	–	–	–	42	7	0.05	5858	892	61.34	6045	901	61.84
8	–	–	–	48	8	0.07	–	–	–	–	–	–

Table 6.5: Experimental results on the Swimming Pool with different choices of components' labeled event structures. Results in the last columns are obtained without using our technique for reducing redundancy, i.e. the truncation may contain $\trianglelefteq$-redundant event(s).

Let us give some details on how the choice of modeling a place is related to the auto-concurrency problem. Table 6.5 shows results on the Swimming Pool while places are represented by the following labeled event structures:

- 2-causality processes (2-$\mathcal{CP}$): each increment event has two direct successors that are increment ones and concurrent; decrement events are pairwise concurrent (see Section 3.3.2).
- 1-causality processes (2-$\mathcal{CP}$): it differs from 2-$\mathcal{CP}$ only on the fact that all increment events are pairwise causal.
- M-$\mathcal{CP}$ like: it is derived from the M-causality process for FIFO-channels where the alphabet M is a singleton (see Section 3.3.3). We have not only that increment events are pairwise causal but also that decrement events are too.

When modeling a place like M-$\mathcal{CP}$, there are few events that are concurrent and labeled by the same label. These events concerns the decrement action that removes initial tokens of such a place. Hence, the unfolding have not much useless events. By comparing the 6 last columns in Table 6.5, one can see that the results obtained with or without our technique for reducing redundancy do not really differ. When using 2-$\mathcal{CP}$, the concurrency between decrement events as well as between increment ones makes the unfolding explode quickly. Our technique for reducing redundancy does not work well in this case.

However, when using 1-$\mathcal{CP}$, redundant events may be completely avoided. The generated truncation has $6 * K$ events where 6 is the number of transition in the Swimming

Pool and K is the number of tokens initially in parameterized places. Although it is not shown in Table 6.5, it is worth noticing that the truncation obtained while using K-bounded process (K-$\mathcal{BP}$ in Section 3.3.2) has the same size $6*K$. Moreover, this truncation intuitively consists of K disjoint sub-structures of which each is the truncation obtained on the Swimming Pool 1, i.e. $K = 1$.

6.3.3 Unbounded Petri nets

We are motivated by a model-checker for infinite systems, but almost all benchmark examples of Petri net are unfortunately bounded. The few unbounded ones are not very suitable due to some advanced type of transitions, e.g. Petri nets with inhibitor arcs or with transfer arcs. Therefore, for experimental purpose, we've created a simple unbounded Petri net which represents a concurrent Producer/Consumer system with n independent production lines and m machines on each line. This example is derived from the one of McMillan [McM95a]. Figure 6.10 illustrates the corresponding Petri net where $n = m = 3$.

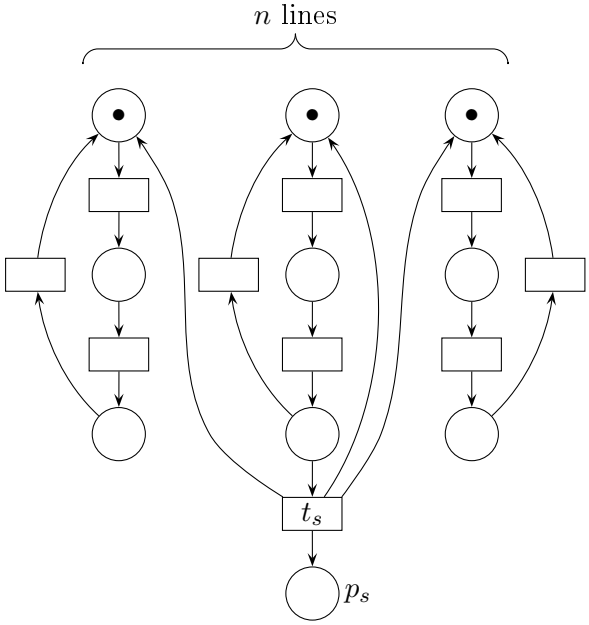


Figure 6.10: A concurrent Producer/Consumer Petri net with $m = 3$ and $n = 3$.

Intuitively, this Petri net consists of an $n \times m$ matrix of places, and another particular place p_s for storing the final product that is combined from the products in n lines. Each place among the n places at the top of n columns (lines), has initially a token on itself. Transitions representing machines allow to move a token either from a place down to the place just below it in the same column, or from a place at the bottom of a column up to the place at the top of the same column. And lastly, there is a transition t_s which allows to, if every places at the bottom of n columns has a token on it, add a new token on the place p_s, and move all tokens at the bottom places of n columns to its top places.

The classical technique for deciding boundedness problem of Petri nets is to compute a Karp-Miller graph. On the example above, the corresponding graph contains many useless interleavings of actions from different production lines. The size of this graph is thus exponential in the size of the example. As shown in Table 6.6, TINA gives

reachability trees that represents m^n markings and have a size of $O(m^n)$. Notice that in the last three cases, verification using TINA can not finish within 10 minutes.

	TINA		ESU		
$m \times n$	T	T(s)	E	E_{cf}	T(s)
5×5	4636	0.02	25	5	0.00
7×5	21396	0.12	35	5	0.01
10×5	115911	1.22	50	5	0.02
5×7	125552	1.16	36	8	0.01
7×7	1094241	14.87	50	8	0.01
10×7	–	–	71	8	0.02
5×10	–	–	46	6	0.01
7×10	–	–	66	6	0.02
10×10	–	–	96	6	0.04

Table 6.6: Experimental results on the Producer/Consumer.

However, ESU resolves the boundedness problem on this Producer/Consumer system while exploiting well its intrinsic concurrency. The prefix generated by ESU is intuitively smaller than or equal to the Petri net representing this system in which there are exactly $|E| = (m-1) * n + 1$ transitions.

Chapter 7

Conclusions

The verification of infinite-state concurrent systems presents two difficult challenges: first dedicated techniques (such as symbolic model checking, abstraction or truncations) must be used to deal with the infinite state space, and then reduction techniques (such as partial-order methods) must exploit the concurrency in the models to fight state-space explosion. In this thesis, we have shown how to combine the unfolding technique, a partial-order method, with analysis techniques for well-structured (infinite-state) systems.

We have presented a general framework for partial-order modeling and analysis of heterogeneous systems. In this approach, systems are modeled as labeled event structures [Win86]. The modelization is no more on the system level (that does not capture concurrency), but rather on a behavioral, branching and non-interleaving level [SNW96]. In labeled event structures, atomic computation steps of the corresponding system are represented by events, and concurrency as well as causality between such events, if exists, are explicitly described. Our labeled event structures for standard systems such as counters and FIFO channels demonstrate that the concurrency may be well captured in this approach.

A reactive system generally consists of several components. Classic models such as synchronized products of labeled transition systems turn out not to be satisfactory when components are concurrent systems. Our solution is modeling them by synchronized products of labeled event structures. The main advantage is that we model not only the concurrency between components but also the intrinsic concurrency inside each of them. Moreover, it permits hierarchical modeling of systems.

On the one hand, at the behavior level, labeled event structures preserve all information about systems in terms of Mazurkiewicz's trace semantics [Maz86], and may be directly used for reasoning about system's properties. On the other hand, since there is no interleaving of concurrent events, their compact size admits efficient verification algorithms. The model-checking concerns first in algorithmically constructing such labeled event structures. We have adapted the unfolding technique [McM95a], initially developed for Petri nets, to labeled event structures. Our algorithms are proved to be correct when constructing component labeled event structures, such as counters and FIFO channels, and allow to efficiently build their synchronized products.

Most of verification problems for infinite-state systems are undecidable. Fortunately, the decidability of interesting properties, for instance termination and boundedness, holds on a subclass of infinite systems having some weak-simulations that are well-preorders. We have introduced well-preordered labeled event structures and shown that decidable results [FS01] may be obtained in this model. In other words, by giving

a definition of a general cutting-context, we have shown that well-preordered labeled event structures admit some finite prefixes that preserve reachability-based properties. Hence, such prefixes may be algorithmically computable, and more interestingly, they are more compact than interleaving ones [Fin91] due to the partial-order approach. We also explain how to obtain standard backward analysis results by using our forward partial-order analysis.

Finally, a prototype implementation, the Esu tool, of our method has been developed. Boundedness, termination, and state covering problems may be checked using Esu. In addition, it has an advanced technique allowing to reduce the auto-concurrency problem that is well-known for Petri nets' unfolding. By using this technique and the truncation technique together, one generally obtains a more compact prefix, and it sometimes produces an "optimal" prefix with just enough events to preserve reachability-based information. The first practical evaluations are very encouraging.

7.1 Future work

The work presented in this thesis can be extended in several ways. We give here a non exhaustive enumeration of possible objectives that, of course, are not really disjoint.

- The first possible extensions should concern the modelization. As discussed in Section 3.3.2 on page 37 and shown in experimental results in Section 6.3.2 on page 151, one needs to choose a value for the parameter k when modeling counters by causality processes. The unfolding algorithm then creates k increasing events when it is necessary. This fact may give rise to harmful auto-concurrency [KK01, KK03] and is different from the original idea of the unfolding technique [McM95a]. One possible solution consists of not only improving our unfolding algorithm but also of making use of our 0-causality process. It certainly demands adapting the truncation technique for synchronized products of labeled event structures so that it does not strictly rely on the finitely-branching property of the components.

  Moreover, we also aim at giving appropriate labeled event structures for standard components other than counters and FIFO channels in order to apply our methods on a larger body of realistic heterogeneous systems.

- Defining the semantics of given systems as labeled event structures and/or designing dedicated unfolding algorithms for those systems is sometimes hard. It requires some prior study on the system's concurrency because the independence between events should be explicitly given. In fact, it is not always possible nor desired to have specific algorithms. Although one may use our event trees containing no concurrency for any component system, it is preferable to give a general algorithm capable of determining independences between events while efficiently constructing the corresponding labeled event structure. Such an algorithm is given in [HST07] allowing to construct a (component) labeled event structure from its induced labeled transition system. The conflicts between events are computed on-the-fly by comparing the markings of their interleavings if they exist. As a result, by applying this algorithm, one obtains corresponding M-causality processes from labeled transition systems modeling FIFO channels over M. However, this algorithm requires modification in order to be applicable to algorithmic construction of synchronized products of labeled event structures.

- Almost all results in this thesis are stated for nondeterministic labeled event structures in which a configuration corresponds to some set of system's states. Although symbolic methods [BCM+92, BW94] are not discussed in this work, we intend to use them in conjunction with our methods. We also plan to consider acceleration techniques [BW94, Sut00], as a tool for truncating (infinite) labeled event structures, hence enforcing the termination of our algorithms while preserving reachability properties.

- Finding abstraction algorithms is a good solution in order to build more compact and concurrent event structures. Structural properties may be used to statically compute over-approximations of the reachability set of a Petri net as shown in [EM00], adapting such results to our framework may be possible. Another big challenge for us is to avoid abstraction algorithms that manipulate system's states as standard abstraction techniques, but rather giving algorithms that compute appropriate over-approximations of system's concurrency. In other words, such algorithms would abstract away causality and conflict information that is irrelevant w.r.t. to a desired property.

- We plan to work on improvement of our unfolding algorithm, and in particular, to deal with the auto-concurrency problem on synchronized products of labeled event structures. Even though our first attempt is encouraging for reachability-based verifications (see Section 6.2.2 on page 141), it turns out not to be entirely satisfactory since the truncation does not preserve Mazurkiewicz's trace semantics.

Bibliography

[AAB99] P. A. Abdulla, A. Annichini, and A. Bouajjani. Symbolic verification of lossy channel systems: Application to the bounded retransmission protocol. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1579 of *LNCS*, pages 208–222. Springer, 1999.

[ABC94] A. Arnold, D. Bégay, and P. Crubillé. *Construction and analysis of transition systems with MEC.* World Scientific Publishing, 1994.

[AČJ00] P. A. Abdulla, K. Čerāns, and B. Jonsson. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1-2):109–127, 2000.

[ACJT96] P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Symposium on Logic in Computer Science (LICS)*, pages 313–321, 1996.

[ACJT00] P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1-2):109–127, 2000.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AIN00] P. A. Abdulla, S. P. Iyer, and A. Nylén. Unfoldings of unbounded Petri nets. In *Computer Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 495–507. Springer, 2000.

[AJ93] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Symposium on Logic in Computer Science (LICS)*, pages 160–170. IEEE Computer Society, 1993.

[AJ94] P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 820 of *LNCS*, pages 316–327. Springer, 1994.

[AJ96] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[AN82] A. Arnold and M. Nivat. Comportements de processus. In *Colloque AFCET "Les mathématiques de l'Informatique"*, pages 35–68, 1982.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Masson, 1992.

[BBF+01] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. McKenzie, and P. Schnoebelen. *Systems and Software Verification: model-checking techniques and tools.* Springer, 2001.

[BCK04] P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: An unfolding-based approach. In *International Conference on Concurrency Theory*, volume 3170 of *LNCS*, pages 83–98. Springer, 2004.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BHFJ03] A. Benveniste, S. Haar, E. Fabre, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. In *International Conference on Concurrency Theory*, volume 2761 of *LNCS*, pages 1–26. Springer, 2003.

[BHK06] P. Baldan, S. Haar, and B. König. Distributed unfolding of Petri nets. In *Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 3921 of *LNCS*, pages 126–141. Springer, 2006.

[BHR06] P. Bouyer, S. Haddad, and P.-A. Reynier. Timed unfoldings for networks of timed automata. In *Automated Technology for Verification and Analysis (ATVA)*, volume 4218 of *LNCS*, pages 292–306. Springer, 2006.

[BM99] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *LNCS*, pages 323–333. Springer, 1999.

[Boc78] G. V. Bochmann. Finite state description of communication protocols. *Computer Networks (and ISDN Systems)*, 2:361–372, 1978.

[BRV04] B. Berthomieu, P.O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14), 2004.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[BSW69] K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Computer Aided Verification (CAV)*, volume 818 of *LNCS*, pages 55–67. Springer, 1994.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CCJ06] F. Cassez, T. Chatain, and C. Jard. Symbolic unfoldings for networks of timed automata. In *Automated Technology for Verification and Analysis (ATVA)*, volume 4218 of *LNCS*, pages 307–321. Springer, 2006.

[CE81] E. M. Clark and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[CF97] G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable (extended abstract). In *Computer Aided Verification (CAV)*, volume 1254 of *LNCS*, pages 304–315. Springer, 1997.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[CGP00] J.-M. Couvreur, S. Grivet, and D. Poitrenaud. Designing a LTL model-checker based on unfolding graphs. In *International Conference on Applications and Theory of Petri Nets (ICATPN)*, pages 123–145, 2000.

[CGP01] J.-M. Couvreur, S. Grivet, and D. Poitrenaud. Unfolding of products of symmetrical Petri nets. In *International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 2075 of *LNCS*, pages 121–143. Springer, 2001.

[CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *International Conference on Concurrency Theory*, volume 1664 of *LNCS*, pages 242–257. Springer, 1999.

[CJ04] T. Chatain and C. Jard. Symbolic diagnosis of partially observable concurrent systems. In *Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE)*, volume 3235 of *LNCS*, pages 326–342. Springer, 2004.

[CJ06] T. Chatain and C. Jard. Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In *International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 4024 of *LNCS*, pages 125–145. Springer, 2006.

[CJEF96] E. M. Clarke, S. Jha, R. Enders, and T. Filkorn. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[CLM89] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *Symposium on Logic in Computer Science (LICS)*, pages 353–362, 1989.

[CM97] G. Ciardo and A. S. Miner. Storage alternatives for large structured state spaces. In *Computer Performance Evaluation*, volume 1245 of *LNCS*, pages 44–57. Springer, 1997.

[Cor96] J.C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering*, 22(3), 1996.

[DJN04] J. Desel, G. Juhás, and C. Neumair. Finite unfoldings of unbounded Petri nets. In *International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 3099 of *LNCS*, pages 157–174. Springer, 2004.

[DJS99] C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of reset P/T nets. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of *LNCS*, pages 301–310. Springer, 1999.

[EC82] E. A. Emerson and E. M. Clark. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Symposium on Logic in Computer Science (LICS)*, pages 352–359, 1999.

[EH00] J. Esparza and K. Heljanko. A new unfolding approach to LTL model checking. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *LNCS*, pages 475–486. Springer, 2000.

[EH01] J. Esparza and K. Heljanko. Implementing LTL model checking with net unfoldings. In *International SPIN Workshop*, volume 2057 of *LNCS*, pages 37–56. Springer, 2001.

[EM00] J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, 16(2), 2000.

[ER99] J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *International Conference on Concurrency Theory*, volume 1664 of *LNCS*, pages 2–20. Springer, 1999.

[ERV96] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1055 of *LNCS*, pages 87–106. Springer, 1996.

[ERV02] J. Esparza, S. Römer, and W. Vogler. An improvement of mcmillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.

[ES96] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.

[esu] Esu. http://www.labri.fr/~tran/esu/.

[FGM+92] J. C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A toolbox for the verification of LOTOS programs. In *International Conference on Software Engineering (ICSE)*, pages 246–259, 1992.

[Fin87] A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 267 of *LNCS*, pages 499–508. Springer, 1987.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.

[Fin91] A. Finkel. The minimal coverability graph for Petri nets. In *Applications and Theory of Petri Nets*, volume 674 of *LNCS*, pages 210–243. Springer, 1991.

[Fin94] A. Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, 1994.

[FS00a] A. Finkel and G. Sutre. An algorithm constructing the semilinear Post* for 2-Dim Reset/Transfer VASS. In *Mathematical Foundations of Computer Science (MFCS)*, volume 1893 of *LNCS*, pages 353–362. Springer, 2000.

[FS00b] A. Finkel and G. Sutre. Decidability of reachability problems for classes of two counters automata. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *LNCS*, pages 346–357. Springer, 2000.

[FS01] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

[FS02] H. Fleischhack and C. Stehno. Computing a finite prefix of a time Petri net. In *International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 2360 of *LNCS*, pages 163–181. Springer, 2002.

[GB96] B. Grahlmann and E. Best. PEP - more than a Petri net tool. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1055 of *LNCS*, pages 397–401. Springer, 1996.

[GHP92] P. Godefroid, G. J. Holzmann, and D. Pirottin. State-space caching revisited. In *Computer Aided Verification (CAV)*, volume 663 of *LNCS*, pages 178–191. Springer, 1992.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Computer Aided Verification (CAV)*, pages 176–185, 1990.

[gra] GRAPHVIZ - *A graph visualization software*. http://www.graphviz.org/.

[GS97] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Computer Aided Verification (CAV)*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[GW91] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Computer Aided Verification (CAV)*, volume 575 of *LNCS*, pages 332–342. Springer, 1991.

[Haa99] S. Haar. On occurrence net semantics of Petri nets. Research Report 3718, INRIA Lorraine, 1999.

[HCF+02] F. Herbreteau, F. Cassez, A. Finkel, O. Roux, and G. Sutre. Verification of embedded reactive fiffo systems. In *Latin American Theoretical INformatics (LATIN)*, volume 2286 of *LNCS*, pages 400–414. Springer, 2002.

[Hel99] K. Heljanko. Deadlock and reachability checking with finite complete prefixes. Technical Report A56, Laboratory for Theoretical Computer Science, HUT, Espoo, Finland, 1999.

[HKK02] K. Heljanko, V. Khomenko, and M. Koutny. Parallelization of the Petri net unfolding algorithm. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 2280 of *LNCS*, pages 371–385. Springer, 2002.

[HKT96] P. Hoogers, H. Kleijn, and P. Thiagarajan. An event structure semantics for general Petri nets. *Theoretical Computer Science*, 153(1-2):129–170, 1996.

[Hol97] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[HST07] F. Herbreteau, G. Sutre, and T-Q. Tran. Unfolding concurrent well-structured transition systems. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 706–720. Springer, 2007.

[Iba78] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.

[ISD+02] O. H. Ibarra, J. Su, Z. Dang, T. Bultan, and R. A. Kemmerer. Counter machines and verification problems. *Theoretical Computer Science*, 289(1):165–189, 2002.

[JP93] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. *Information and Computation*, 107(2):272–302, 1993.

[Kho03] V. Khomenko. *Model Checking based on Prefixes of Petri Net Unfoldings*. PhD thesis, University of Newcastle upon Tyne, 2003.

[KK01] V. Khomenko and M. Koutny. Towards an efficient algorithm for unfolding Petri nets. In *International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 366–380. Springer, 2001.

[KK03] V. Khomenko and M. Koutny. Branching processes of high-level Petri nets. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 2619 of *LNCS*, pages 458–472. Springer, 2003.

[KK05] B. König and V. Kozioura. AUGUR - a tool for the analysis of graph transformation systems. *Bulletin of the EATCS*, 87:126–137, 2005.

[KKV03] V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of Petri net unfoldings. *Acta Informatica*, 40(2):95–118, 2003.

[KKY04] V. Khomenko, M. Koutny, and A. Yakovlev. Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental SAT. In *International Conference on Application of Concurrency to System Design (ACSD)*, pages 16–25. IEEE Computer Society, 2004.

[KM69] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[Kos82] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *ACM Symposium on Theory of Computing*, pages 267–281, 1982.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[LB99] R. Langerak and E. Brinksma. A complete finite prefix for process algebra. In *Computer Aided Verification (CAV)*, volume 1633 of *LNCS*, pages 184–195. Springer, 1999.

[LI05] Y. Lei and S. P. Iyer. An approach to unfolding asynchronous communication protocols. In *Formal Methods*, volume 3582 of *LNCS*, pages 334–349. Springer, 2005.

[LS02] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theoretical Computer Science*, 274(1-2):89–115, 2002.

[May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[Maz86] A. W. Mazurkiewicz. Trace theory. In *Advances in Petri Nets*, volume 255 of *LNCS*, pages 279–324. Springer, 1986.

[MBC+95] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons Ltd (Import), 1995.

[MC99] A. S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 1639 of *LNCS*, pages 6–25. Springer, 1999.

[McM95a] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[McM95b] K. L. McMillan. Trace theoretic verification of asynchronous circuits using unfoldings. In *Computer Aided Verification (CAV)*, volume 939 of *LNCS*, pages 180–195. Springer, 1995.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 481–489, 1971.

[MR97] S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Computer Aided Verification (CAV)*, volume 1254 of *LNCS*, pages 352–363. Springer, 1997.

[MRE96] S. Melzer, S. Römer, and J. Esparza. Verification using PEP. In *Algebraic Methodology and Software Technology (AMAST)*, volume 1101 of *LNCS*, pages 591–594. Springer, 1996.

[NPW80] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science*, 13(1):85–108, 1980.

[Pel94] D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Computer Aided Verification (CAV)*, volume 818 of *LNCS*, pages 377–390. Springer, 1994.

[pep] PEP. http://theoretica.informatik.uni-oldenburg.de/~pep/.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Univ. Bonn, 1962. Schriften des Instituts für Instrumentelle Mathematik.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science (FOCS)*, pages 46–57, 1977.

[Pra86] V. R. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer, 1982.

[Rei85] W. Reisig. Petri nets with individual tokens. *Theoretical Computer Science*, 41:185–213, 1985.

[San04] D. Sangiorgi. Bisimulation: From the origins to today. In *Symposium on Logic in Computer Science (LICS)*, pages 298–302, 2004.

[San07] D. Sangiorgi. On the origins of bisimulation, coinduction, and fixed points. Research Report 24, University of Bologna, 2007.

[SG90] G. Shurek and O. Grumberg. The modular framework of computer-aided verification. In *Computer Aided Verification (CAV)*, volume 531 of *LNCS*, pages 214–223. Springer, 1990.

[SK04] C. Schröter and V. Khomenko. Parallel LTL-X model checking of high-level Petri nets based on unfoldings. In *Computer Aided Verification (CAV)*, volume 3114 of *LNCS*, pages 109–121. Springer, 2004.

[SNW96] V. Sassone, M. Nielsen, and G. Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170(1-2):297–348, 1996.

[SSE03] C. Schröter, S. Schwoon, and J. Esparza. The model-checking kit. In *International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 2679 of *LNCS*, pages 463–472. Springer, 2003.

[Sta89] E. W. Stark. Connections between a concrete and an abstract model of concurrent systems. In *Mathematical Foundations of Programming Semantics*, volume 442 of *LNCS*, pages 53–79. Springer, 1989.

[Sut00] G. Sutre. *Abstraction et accélération de systèmes infinis*. PhD thesis, ENS de Cachan, 2000.

[SY96] A. Semenov and A. Yakovlev. Verification of asynchronous circuits using time Petri net unfolding. In *ACM/IEEE Design Automation Conference*, pages 59–62, 1996.

[tin] TINA. http://www.laas.fr/tina/.

[Val89] A. Valmari. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, volume 483 of *LNCS*, pages 491–515. Springer, 1989.

[Val90] A. Valmari. A stubborn attack on state explosion. In *Computer Aided Verification (CAV)*, volume 531 of *LNCS*, pages 156–165. Springer, 1990.

[VSY98] W. Vogler, A. L. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *International Conference on Concurrency Theory*, volume 1466 of *LNCS*, pages 501–516. Springer, 1998.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS)*, pages 332–344, 1986.

[WG93] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *International Conference on Concurrency Theory*, volume 715 of *LNCS*, pages 233–246. Springer, 1993.

[Win82] G. Winskel. Event structure semantics for CCS and related languages. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 140 of *LNCS*, pages 561–576. Springer, 1982.

[Win86] G. Winskel. Event structures. In *Advances in Petri Nets*, volume 255 of *LNCS*, pages 325–392. Springer, 1986.

Index