A State-oriented, Partial-order Model and Logic for Distributed Systems Verification

Vasumathi K. Narayanan

A Thesis

in

The Department
of
Computer Science

Presented in Partial Fulfilment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

April 1997

0-612-40312-2

ABSTRACT

A State-oriented, Partial-order Model and Logic for Distributed Systems Verification

Vasumathi K. Narayanan, Ph. D
Concordia University, 1997

A theory of state-oriented, partially-ordered model named, *Communicating Minimal prefix machines* (CMpms) that represent a fixed set of processes, is presented. Each of these Mpms is a possibly *infinite, State Transition System*. Communication among a set of Mpms is by synchronization. Enriched by a global, *causal dependency relation* among the Mpm-states that is *partial*, the disjoint union of CMpms comprise a *sum machine*. It is shown that the set of all global states of the *product machine* of CMpms is obtainable dynamically from the local Mpm-states of the sum machine using the *monotonicity* property, linking the product machine's global states and the sum machine's local ones. In this sense, the product-machine is generated *virtually*, simulated by the sum-machine.

More interestingly, it is shown that from every given set of Communicating Finite state machines (CFsms), a truncated (finite) version of a set of CMpms and so a sum machine can be generated in a recursive functional manner. The set of global states of the product machine of CMpms *surjectively map onto* that of CFsms. Consequently, it is possible to generate all the global states of the latter using the local states of the sum machine composed by a corresponding set of CMpms. The *sum machine* models *true causality* and hence *true sequence*, *true concurrency and true choice* among local states, as exhibited by the original input CFsms specification without enumerating all the *runs* of the system nor all the *nondeterministic interleavings* of each run as opposed to the *product machine*.

A *Spatial, temporal logic* or *space-time logic* named CML (Computational Mpms Logic), is proposed that combines the conventional *branching time* feature with what is proposed as *branching space* feature: the latter corresponds to *concurrency* just as the former to *conflicts*, as exhibited by the input specification. CML therefore introduces operators to reason about properties of *interleavings* within each run, *orthogonal* to the *branching time*

operators, to reason about *runs*. The logic turns out to be more expressive than the ones in vogue for specifying properties of concurrent systems. In addition, CML coupled with the *sum machine* model, enables the implementation of a *deterministic model checker* algorithm to verify the properties of a given CFsm system that is free of *exponential complexity* caused by the *enumeration* of runs as well as that of *all interleavings* within each run.

Acknowledgments

I thank both my parents who were my very first teachers in life. I thank them both, my uncle, aunt and grand parents for all the education and support they had provided while fostering me. I take this opportunity to think in gratitude, of all my teachers who influenced my thinking, formally or informally, through out my life which led me to take up this research.

It is said that, any accomplishment of work, however small, is *ninety nine percent, inspiration* and *one percent, perspiration*. I attribute the entire credit of the former to my teacher, mentor and supervisor, Prof. H.F.Li. I thank him for all his inspiring, deeply self-absorbed flow of lectures during our discussions. I thank him for accepting the proposal of this research despite the fact that it was contrary to his belief. I greatly appreciate his confidence in still accepting and funding the project, supervising and refereeing the work at all stages, providing me with very useful and challenging feedback and criticisms at every stage, despite my non-resident status. They went a long way in raising my standard of thinking, with a better grip of the subject and improving the presentation after every communication with him. I thank him for all the useful pointers to the references. I am grateful to him for his patient scrutiny and corrections of all iterations of the thesis draft. It makes me very lucky indeed to have associated with him for a decade as his student.

I like to thank all my instructors at Concordia University, who enriched my knowledge: I thank Dr. P.Grogono, my first teacher at Concordia, who apart from his lectures, gave me a chance to work on a survey of *Paradigms of Concurrent systems* as a starting point of this research. I thank Dr. B.Sarikaya for reinforcing my interest on synchronous communication systems, and throwing light on many related languages; the language LOTOS in particular, in the context of protocol validation and concurrent system verification. I could gather very useful survey material from his course. I thank Dr. W. Atwood for giving me a chance to compare various orthogonal paradigms in concurrent systems and conclude that the state-oriented one is as workable as event-oriented paradigm, which has been more popular than the former in the last decade. This sowed the seed for this thesis.

I thank Dr. M.Okada for introducing me to formal and mathematical logic. I thank Prof. V.S.Alagar for giving me a chance to work on a LOTOS based project in his course which enriched my interest in distributed systems in the context of *specifications* and *Abstract data types*. I thank Dr. D.K. Probst for giving me a chance to explore Distributed Systems in the context of *Operating Systems Design*. I thank Dr. C. Lam, Dr. T.D. Bui and Dr. J. Opatrny for letting me audit their fundamental courses on Computer Science, which were to my benefit. I thank Dr. Li again for directing me to all the above instructors and their courses, relevant to my research work and its foundation.

I thank Dr. D.K.Probst again for hearing my preliminary presentation and for his encouraging comment that there is something positive in the work which with simplified presentation would make a meaningful one. It gave me a lot of impetus in expediting matters, and in revising the manuscript.

I take this opportunity to thank Dr. Alagar again. I thank him for his advice and useful pointers to references on Temporal logic particularly while writing the thesis proposal. I am grateful to him for his wise, compassionate and philosophical advice whenever we met. I am grateful to him for patiently hearing and comprehending my presentations despite their inadequate organization, for recognizing the positive aspects and making useful suggestions, all of which contributed to the beginning of the thesis write-up. I am very thankful to him for his patient reading and corrections that helped me refine the presentation of the final write-up.

I thank the authors of the prime references made through out this thesis and also the rest, which provided me a great deal of motivation, guidance and as benchmarks for comparison in my research. Apart from the facilities and services of the libraries at my *Almamater*, Concordia University at Montreal, I thank also the library facilities and services of all the following Institutions and Organizations: McGill University and University of Montreal at Montreal, Bell Northern Research, Carlton University and National Research Council at Ottawa and Stanford University at California. Though not all the information I accessed, the reminiscences of my reading experience at each one of these elite facilities and their picturesque surroundings are indelibly imprinted in my memory. I can never forget the unfailing searches at NRC library, Ottawa, a sophisticated facility I have ever come across.

My sincere gratitude to the thesis examiners that gave me their comments and modifications after the defence, which were very useful to me in making this revision. I only wish, I had a chance to respond to most if not all of them, during the defence.

I thank Dr. R.Jayakumar for his patient listening of my presentations. I thank Dr. T.Radhakrishnan, Dr. P. Goyal and Dr. B. Desai for their wishes; the rest of the staff and secretaries of Concordia University, in particular Ms. Stephanie Roberts, Ms. Halina Monkiewicz, Ms. Angie De Benedictis and Ms. Terry Czernienko for their kind services, and the entire management including the Dean's office for all the support and facilities provided to me as a student.

My sincere apologies for all the errors that may remain uncorrected in the manuscript, typographical or otherwise.

I thank all my family for their support. I am thankful to my husband Hari, for all his support and incredible patience with me especially when I was adjusting to life in Canada. I am thankful for his wise suggestions and advice whenever I needed them. I am grateful to him for providing the resources, all the documentation facilities and help; for being very tolerant to my domestic slips during this commitment. I thank both my children, Sumitra and Spatika, both of whose births during the course of this work gave me more momentum and energy towards this research, than ever before, despite my parenting occupation. I apologize to them for being an absent-minded parent on innumerable occasions. I thank them all for having thrived patiently through my illness.

I thank my older brother Rathnam, for all his clinical support during my critical illness in the course of this work and my younger brother Sridhar, and sister Jayashree, for all their continuous moral support and encouragement. I thank all those well-wishers who are not specifically mentioned here, from the circle of extended family and friends, for their support and concern during this research and my sickness. I thank my parents again for going through the anxiety with me, especially my mother for substituting me in my domestic responsibilities for more than a year during the course of this research and its completion. I thank her for teaching me great axioms of parenting, which is a more challenging profession than anything else in my conclusion after this research.

I thank the Almighty for giving me the strength and perseverance to undertake and complete this research in its current form amidst much of culture shock, climatic shock and ill-health.

Dedication

I dedicate this work to my dear grandmother, who reached the heavenly abode at the end of the year '96, at the age of ninety six, who was ever a symbol of patience, hard work and compassion.

Table of Contents

List of Figures

List of Tables

List of Special Symbols

Symbols used in the Computational Model (CMPms and CFsms)

Notation	**Meaning**
F_i	A Finite state machine, Fsm_i
s_{fi}	A state of F_i
S_{fi}	Set of states of F_i
R_{fi}	Local reachability relation of F_i
Π	Cross Product
ΠF	Product machine of F_i, $i = 1..n$
s_f	A state of ΠF
	i.e, Global-state of $\{F_i, i=1..n\}$
M_i	A Minimal prefix machine, Mpm_i
s_{mi}	A state of Mpm, M_i
S_{mi}	Set of states of M_i
R_{mi}	Local reachability relation of M_i
R^+_{mi}	Transitive closure of R_{mi}
R^*_{mi}	Kleene closure of R_{mi}
$sync_{in}$	synchronous input partnership relation
$sync_{out}$	synchronous output partnership relation
$\leq$	causality (dependency-order) among Mpm-states/vectors
$=$	Equality (*identity* or *similtaneity*) of Mpm-states or vectors
Σ	Disjoint union of sets
$\cup$	Set union
$\cap$	Set intersection
ΣM	Sum machine of M_i, $i=1..n$
seq	Sequence relation among Mpm-states
$conf_i$	Local conflict relation among local Mpm-states of S_{mi}

Notation	Meaning
conf	Global conflict relation among ΣS_{mi}
co	Concurrency relation among non-local Mpm-states of ΣS_{mi}
$Mp_i(s_{mi})$	Minimal prefix of s_{mi}
$C_i(s_{mi})$	Local configuration of s_{mi}
C	General configuration
Fsv(C)	Final state vector of C i.e., a reachable Mpm-state vector
P_i	A path of state-tree of M_i
ΠM	Product machine of M_i, i =1..n
s_m	A state of ΠM
	i.e., Global-state of $\{M_i, i=1..n\}$
	i.e, Reachable Mpm-state vector of ΣM
R_m	Reachability relation of ΠM
Πr	A run of ΠM
ΠI_r	An interleaving of Πr
Πr_f	A run of ΠF
ΠIr_f	An interleaving of Πr_f
$\subseteq$	Subset of
$\in$	Member of
$\notin$	Not a member of
$<>$	Not equals

Symbols used in the logic (CML)

Notation	Meaning
$CML_{\Pi F}$	CML with respect to the *product-machine* ΠF.
$CML_{\Pi M}$	CML with respect to the *product-machine* ΠF.
$CML^*_{\Sigma M}$	CML with respect to the *extended sum-machine*, $\Sigma^* M$.
ap_{fi}	atomic proposition of s_{fi}, i.e., $p_{fi}(s_{fi})$

Notation	Meaning
ap_{mi}	*atomic proposition* of s_{mi}, i.e., $p_{mi}(s_{mi})$
^	Logical *Negation* operator
$\wedge$	Logical *And* operator
$\vee$	Logical *or*
=>	implication operator
<=>	equivalence operator
$\forall$	Universal operator
$\exists$	Existential operator
X	Next state modal operator
$\underline{X}$	Previous state modal operator
F	Future state modal operator
$\underline{F}$	Past state modal operator
G	Operator for: Always in future
$\underline{G}$	Operator for: Always in past
g *until* h	predicate g remains true *until* h does
g *since* h	g is true ever *since* h was true
A_r	Universal run operator
E_r	Existential run operator
A_{Ir}	Universal interleaving operator
E_{Ir}	Existential interleaving operator
g *pos-wait-for* h	g *possibly waits for* h
g *must-co-wait* h	g and h *necessarily wait for each other*

Chapter 1
Introduction

1.1 Goals of Research

The goals of this research are to:

- Develop a model of a *finite distributed*[1] *system of processes* to express *causality, sequence, choice*[2] and *concurrency* among the entities, states in particular, of the component processes in their *true* forms of occurrence, faithfully as represented by the *input specification.*
- Specify a wide range of *safety* and *liveness* properties of such a system formally with the aid of *logic*, without foregoing the above characteristics of the system, *concurrency* and *choice* in particular, represented by the model of the system; thereby attempting to fill the void in the table cited by Reisig in his foundational survey table of [2].
- Develop the verification algorithms to verify the above properties as efficiently as possible, in particular without the *enumeration of all interleavings* and the resulting *state-space explosion* and exponential complexity incurred by the traditional state-transition systems.

We assume an input specification of a fixed set of *n communicating sequential processes* represented as a set of *n Communicating Finite state machines* (CFsms).

1.2 Verification of Finite Concurrent Systems

Proof of correctness of concurrent/distributed systems is non-trivial unlike that of sequential systems since there is no master process or global clock controlling the system. The component processes run according to their own local clocks.

Fortunately, the entire set of properties of a concurrent system can be grouped into two categories, viz., *safety* and *liveness* and proving the correctness of the system amounts to

---

[1] In this work, we treat the phrases, *distributed system* and *concurrent system*, equivalently.

[2] The terms, *true choice* and *conflict* are synonymous throughout the sequel.

verification of these properties. This is the relevance of properties in the verification of concurrent systems.

A *safety* property is generally described as: *negation* of undesirable phenomena in a system and a *liveness* property as: *assertion* of desirable ones happening in the system. *Deadlock freedom* and *system-invariance* properties are the typical examples of safety properties; *properties* specified with *inevitability* and *eventuality* are among a variety of interesting liveness properties.

1.2.1 Model-Checking

Model-checking is a popular methodology for verifying the properties of concurrent systems. It consists in determining if the *safety* and *liveness* properties of the system expressed by a *specification tool* are indeed represented by the *computational model* of the system. Since these properties stem from the *basic characteristics* of the system, it is important that the underlying computational model represents these characteristics faithfully, as exhibited by the system.

A *rich* computational model of a system is one, which reflects the above mentioned primitive *characteristics* of the system viz., *sequence, choice, causality* and *concurrency* among its entities (states and events), in a faithful fashion as originally exhibited by the system.

A rich or a powerful *specification tool* is one which should be able to express the above characteristics by suitable means.

1.3 Computational Models of Concurrent Systems

We focus on *finite, concurrent system* verification and there are many ways to model them. Depending on the primary entities modeled in the system, we have mainly two classes of models, viz., *state-oriented* ones and *event-oriented* models. In the state-oriented systems, many of the system entities are defined primarily with respect to the *states* of the system often accompanied by explicit representation of states. In the latter models on the other hand, they are defined with respect to the *events* that occur in the system. *State-transition systems* come under the first category. Petrinets[7], can be put under this category albeit indirectly, as will be elaborated. The *event-structure* based models [15], [18] fall into the second. Each have their merits and demerits as described below.

1.3.1 State-Oriented Paradigm

The state-transition system is a classical model. Traditionally, the concurrent execution of a given finite system of n communicating sequential processes, each of which is modeled as a *Finite state machine* is represented as a single state-transition system, conventionally referred to as the *product machine*.

The main advantage of a *product machine* is that all the possible *states* of the system are *explicitly represented* and since the properties (*safety* and *liveness*) of a system are associated with its states, it is always beneficial to ascertain all the states of a system in order to verify its properties. But the two main drawbacks of this paradigm are:

(i) The product machine loses track of all the *basic characteristic* relations among the local entities of component processes, in particular, *concurrency* and *conflicts* among them (referred to as *true concurrency* and *true choice* respectively), as exhibited by the input system of processes. It simulates concurrency among *unrelated* states and events by arbitrarily and *artificially* ordering them (and thus creating a *total-order* among them) in all possible combinations of *non-deterministic choices* popularly referred to as *non-deterministic interleaving* of events and states. These artificial, nondeterministic choices are treated as if they are the true choices of the system thus corrupting the purity of both *(true) concurrency* and (true) choice, exhibited originally by the system. Models of the state-oriented paradigm are therefore referred to as, *totally-ordered models* and synonymously, *interleaved models* [1] as well as *system models* [53].

(ii) As a consequence of the above artificial representation of *concurrency* and consequent artificial growth of *sequence* and *choice*, there is an exponential (in the number of component processes) number of *reachable* global-states of the system in the product-machine, which is commonly referred to as the *state-space explosion* problem.

1.3.2 Event-Oriented Paradigm

The models in this category consider *events* as their primary entities, as mentioned. While relating the events according to their order of *occurrences* (often called *causality* or the *dependency-order*), events that are *unrelated* (by causality) are represented as they are. Since this is the case with almost all of the models in this paradigm[18], [15], these are referred to as *partially-ordered* models and also known as *behaviour models* [53].

The main advantage of these models is that by capturing the causality among events, they retain the *true concurrency* of the given system as originally exhibited by it, without ordering them artificially, as by their peers of state-oriented paradigm. As a result, there is no state-space explosion in these models. But these models have their own draw-backs as follows:

(i) The event-oriented models do not represent the global-states of the system explicitly and so many characteristics or attributes of the system are defined with respect to the *events* of the system. This is not conducive to the *verification of the properties* of a system because, the *properties* are the attributes of the *states* of the *system*, irrespective of the paradigm. Consequently, the checking of any reasonable property is not quite direct or straightforward in these models. Not all interesting properties can be expressed without global states.

(ii) The event-oriented models are not based on *operational semantics* or in other words, there is no direct connection to *automata theory* in this paradigm. Consequently, there is no *finite acceptor* for the *prime event-structures* [34]. So, implementing the model-checker is hard, however rich the model and the corresponding specification tool may be.

1.3.2.1 Petrinet Models

Petrinet models, lie at the cusp of the state-based and event-oriented paradigms. With their *places* and *transitions* (or alternatively, with their *conditions* and *events)* defined, *global-states* are represented by the vector of *places/conditions* holding the *tokens* at any time. By doing the *reachability analysis*, this model can be transformed to a global *state-transition system* just like the product machine discussed in the last subsection. So, they come under the category of *system models*. But with the places/conditions and transitions/events represented as they are, the *causality* among the events are captured and *true concurrency* expressed. In this sense, they also belong to the category of event-oriented models.

There is a modeling drawback of Petrinet models as explained below:

(i) There is a *flow relation* defined in Petrinet model as:

$(S \times T) \cup (T \times S)$, where:

S is the set of *places* similar to local states of Fsms,

T is the set of *transitions* similar to events.

The *flow-relation* is such that it does not in general capture the *occurrence order* i.e., the *causal relation* among its places. Only places that are in *sequential order* are related, even though their causal dependency is observable physically through the flow of *tokens*. The transitions corresponding to events alone are modeled according to their *causality*. Thus we see that the mathematical mapping of what is observable as the mechanics of the system is *incomplete* or *not faithful*. What is supposedly defined by the *flow-relation* as *causality*, actually represents *sequentiality* as far as the places are concerned, thus corrupting both the relations (causal and sequential).

(ii) This model is also capable of representing a dynamically varying number of processes due to their *birth* and *death* during the course of the system's behaviour. A given process may spawn its own child processes. Multiple processes can merge with a parent process. This is clear from the variable number of tokens appearing from place to place and the fact that they are not necessarily conserved after the occurrence of a transition by the ones before the occurrence of that transition.

When such a powerful model is chosen to represent the system of a fixed set of processes, the *locality* information, that is, the *identity* of the individual components, is lost from their joint system behaviour. As will be shown, preserving the locality of the components in the composition and thereby their respective entities such as states (analogous to places), *propositions* that qualify these states and *conflicts* among them is imminent for a *deterministic verification*, which is free of the exponential complexity.

1.4 Logic in System Verification

The area of logic is an alternative that has strong support from a large segment of software engineering community. The application of logic in verification is contributed by the efficient search of the entire space of possible behaviours. More than the characteristic of infallibility, popularly attributed to it, what logic accomplishes is the *efficient search of* *combinatorially large or even infinite state spaces*, for all the known types of bugs in a practical amount of time[35][1]. No methodology comes near the efficacy of logic in that role, particularly in the case of infinite search spaces where mathematical induction permits seeking out in finite time every nook and corner that may hide a known type of bug, to quote the referred article above.

To further quote the above reference,

"logic works best when understood as a discipline for manipulating not just symbols, (*proof theory*) but also facts about some world (*model theory*). To the latter end, one develops a mathematical model of that world, and evaluates the *soundness* of the proof system relative to the model. The model must be *faithful* to the world, yet simple enough to permit the soundness of the logic to be assessed.

One weakness of logic is that it can not guarantee the recognizability of bugs of a kind not anticipated by the *axioms* of the logical system. For this and other reasons, logic should be viewed as just one player on a team whose overall goal is improved reliability. Logic has proved a valuable player in this role, fully justifying its continued support and growth".

1.4.1 Computational Model & Temporal Logic

Temporal logic is a suitable and commonly used specification tool for concurrent/distributed systems. This is because, the component processes run according to their own *local time scales*, which have to be somehow integrated and consolidated to arrive at the global properties of the system. Temporal logic is a tool with which, one can express the *past* as well as *future* modalities apart from the *present* to formulate and prove the properties (and thus verify them) of concurrent/distributed systems [4], [24] with the above mentioned characteristic.

Not all computational models are *rich* enough, and similarly the logics. Depending on whether or not *true choice* is represented, we have *branching* or *linear time* logics. Depending on whether or not *concurrency* among the elements of the system (states and events) is represented, we have logics specified over *partial* or *total order* structures.

---

[1] The actual quotation is from the preface of the entire volume of the reference [35].

In the linear-time category, *conflicts* (that are *true choices*) exhibited by the specification of the system are hidden, i.e., not represented by the underlying model of the system, supported by the logic. Instead, *sets of independent, disjoint execution sequences* of the system are handled by the logical formulae. Each execution sequence forms a single *continuum* in the time scale.

While in the case of branching-time logics, all the different execution sequences are connected into a *tree* so that the *conflicts* exhibited by the specification are not hidden and are available for reasoning.

In practice, quite a few of the linear time logics support *partially-ordered* models of the event-oriented paradigm[4][21]; similarly, the branching-time ones support the *totally-ordered* models in the state-oriented paradigm [1]. The combination of both the *partial-order* feature and *branching-time* semantics seems to be a rarity. The reasoning of this observation is elaborated in what follows.

1.4.1.1 Taxonomy of Computational Models and Temporal Logics

The following figures tabulate the taxonomy of the models of concurrent systems and the corresponding temporal logics in the respective platforms, reproduced from [2] for emphasis here.

Fig. 1 Taxonomy of models of concurrent systems

		What is a Single Run?	
		A sequence of events ordered in time.	A partial order of events ordered by causality.
How are runs grouped together, representing a system's behaviour?	A set of detached runs	A set of sequences	A set of partial orders.
	A branched structure indicating conflicts	A tree	An event structure with conflicts

Fig. 2 Taxonomy of different classes of Temporal Logics

	Interleaved execution sequences	Partially ordered, causality based runs.
Detached runs	Linear time temporal logic	Pinter,Wolper [11] Katz[12], Lodaya[21] Reisig [4]
A branched structure of runs with conflicts	CTL* etc.	*F(B)[2], CTL-X*[40] **<Theme of our research>**

An important characteristic and the trend observed from the taxonomy illustrated at both the tables above, upon which our research is essentially centered around, are that:

- *True concurrency* is represented in the *linear time temporal logics* but not *conflicts.* On the other hand, *choices* (not necessarily the *true choices*) are represented in the *branching time logics* but not *true concurrency.*

1.4.1.2 Theme of Our Research

As shown in the bottom, right corner of the table of Fig. 2 above, there have been attempts to propose a suitable logic reported in [2] called *F(B)* and most recently in [40] of CTL-X (CTL without the *next-time operator*) with a [1]*model/structure* that represents both the above highlighted aspects of *true concurrency* and *true choice*. *F(B)* is based on *Occurrence nets* whose process semantics does not support conflicts/true choices directly at the same basic computational level as *sequence/causality* and *concurrency* that are complements of each other. This was mentioned in a previous section. CTL-X obviously does not cover the *next-time* temporal modality in addition to being exponentially complex in general, when it comes to implementation of a verifier of properties with it, model-theoretically.

[1] Logics are often associated with *models and structures* in their semantic domain as with languages in syntactic one.

Richer the computational model and the logic, the more difficult it is to implement a model-checker, let alone the tractability of the checking. Though a theoretical possibility, the computational model combined with the temporal logic of [2] are not implementable in concrete form.

It is our claim that the above lacunae stems from the inherent drawbacks of the Petrinet model and its derivatives due to:

- The incomplete representation of causality/flow-relation explained already,
- Conflicts not being represented at the basic level of processes which is related to the above issue, and
- Highly general assumption of dynamically varying number of processes.

We claim further, that with the restricted assumption of a fixed set of processes in the models of *state-oriented paradigm* and with a refined notion of *causal dependency-order* among the *states* (specifically, the state entries) of component processes, it is indeed possible to make them *partially-ordered* as well, without explicitly generating the *totally-ordered product machine*.

Proposed as *'sum machine'*, this partially-ordered composition generates all the reachable global-states of the product machine dynamically, using only the set of (local) states of component machines that are associated with a minimal set of global states. In this sense, the two machines are equivalent, with the product machine generated *virtually* by the sum machine that is free of the state-space explosion of the former. An extension of the temporal logic which is a *spatial, temporal logic* to specify and verify the properties of the sum-machine and so of the product-machine of a given input of CFsms, including both the characteristics of *true concurrency* and *conflict* relations is proposed. This is the underlying theme of this research.

1.4.2 Theorem Prover Versus Model-checker

Theorem prover is the traditional approach to verification using temporal logic. In this approach, the axioms and inference rules of the logic are used as a deductive system. The proof that a design of the system meets it specification is constructed manually using the above mentioned axioms and inference rules. This task of constructing the proof is laborious and a great deal of work is required to organize the proof. Even the simplest logics are inherently complex with this approach [1].

In the case of finite concurrent systems, the literature shows that the proof construction from axioms and inference rules is unnecessary. Instead, the *model theoretic* approach aims to determine algorithmically whether the system meets its specification expressed as a set of temporal logic formulae. For instance, a model checker algorithm for CTL, a branching time temporal logic, is a pioneering work [1],[5] that has a great deal of influence on this work, particularly in the extension of the temporal logic CTL and the associated model-checker algorithms.

1.5 Summary of the Desirable Needs of a Model-checking Method

- A model-checker must be supported by a classical computational model that faithfully represents all the characteristics of the given specification, especially the three basic relations of *sequence, choice and concurrency* and *causality* among states as well as events.
- It must be supported by a formal logical language (propositional or higher-order) that can express all the properties, covering both safety and liveness properties of the system specification to be checked.
- The checking must be algorithmic as opposed to heuristics and as tractable as possible.

1.5.1 The Drawbacks of Currently Existing Popular Methodology

Owing to the fact that the traditional model-checker based on *Kripke structure* supported by the logic CTL [1] has an exponential complexity in the worst case, which is attributed to the *state-space explosion* of its *total-order model*, there has been a family of methods called *Partial-order Reduction methods* [40], [44], [9], [10], [13], [17] evolved in the last decade.

Even though these methods are quite successful in practice, they are not based on a classical model in the following sense: though called PO based, they choose a representative interleaving among all that are otherwise possible in a total-order model with the assumption that if a property is true for one, it is also true for all. This assumption is valid for safety properties but not for more versatile liveness ones in general. Because of the above simplified representation of total-order view, there is no *partial-order based, branching time logic* supporting these methods in a classical sense.

All these methods are aimed at constructing a *reduced state-graph*, based on exploring for each visited state only a *subset* of the enabled operations, so that only some of the successors of that state are expanded. Hence these methods are called *set-methods*. In these methods, as reported most recently in [40], finding such subsets (called *optimal ample set method*) is in general NP-complete and any implementation of it must use heuristics.

1.5.2 The Motivation and Proposed Work

The drawbacks mentioned in the last section above, make an important motivation to develop a model-checker that is based on a classical partial-order model supported by formal logic (to express both liveness and safety properties), as well as *algorithmic* which is efficient at that, as much as possible.

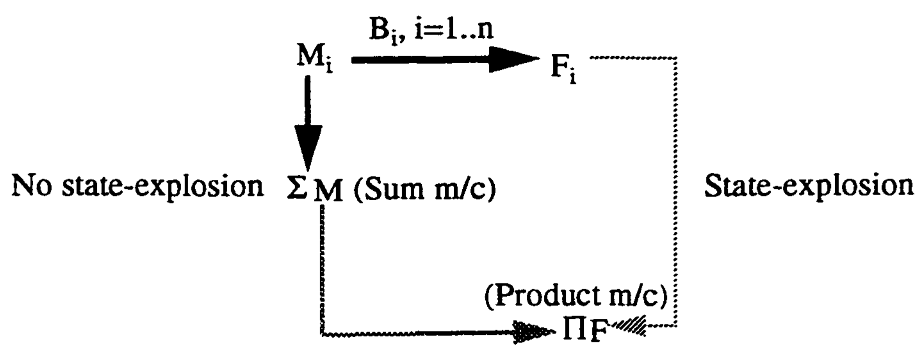
In this work, we propose a *partial-order model* in the *state-oriented paradigm* that alleviates the drawbacks of the product machine as well as those of the net models of the *event-based paradigm*. We also propose a *branching-space* (to cover the PO-semantics), *branching-time* logic to express the properties and a model-checker to verify if the properties expressed as the formulae of this logic are satisfied by the model.

Precisely, Chapter-2 defines and develops the theory of CMpm (Communicating Minimal Prefix Machines) system, from a set of *state-transition systems* called *Minimal Prefix Machines* (Mpms), M_i,i=1..n, each of which is a *deterministic*, possibly *infinite*, machine. The *sum machine* ΣM of the CMpm system is defined as a *disjoint union* of the component machines M_i, i=1..n, based on a global *dependency-order* representing *causality* among the state entries which is a *partial*-order (PO). We also define the traditional *product composition* ΠM of M_i, i=1..n. The causality among local Mpm-states is extended to define the notions of configurations and their Final-state vectors.

We assume a given *input specification* of a set of communicating Fsms, F_i,i=1..n each of which is finite and possibly non-deterministic, constituting a CFsm system.

We define a *set of n functions* that map entities of M_i,i=1..n to those of F_i,i=1..n respectively, using which we construct ΣM (and so M_i,i=1..n) corresponding to the given input, F_i, i=1..n. We also define a *surjective mapping* from ΠM *onto* ΠF the latter being the product machine of F_i,i=1..n. The reachable state vectors of the *extended sum machine* Σ^{*}M *correspond one-to-one*, to the global states of ΠM. Composing the above two mappings, we deduce the *surjection* from the reachable state vectors of ΣM *onto* those of ΠF. This is the important result of this research, as the sum machine ΣM *does not enumerate all the possible runs and their non-deterministic interleavings* as opposed to the product machine ΠF, but can generate all of them or their properties at the time of verification, dynamically. Following figure illustrates the mapping between M_i and F_i and shows the path of state-explosion in the absence of sum machine on the right side, and that of no-state-explosion with sum machine introduced on the left in order to construct ΠF from F_i,i=1..n.

Fig. 3 Pictorial representation of Role of CMpms in alleviating state explosion of CFsms



Finite model of CMpms with *cut-off points* is defined. Finite, *deterministic model* of CMpm system is proved *equivalent* to *non-deterministic model* of CFsm system using the functions B_i, i=1..n.

Chapter-3 introduces the proposed *branching space-time (i.e., branching-space* and *branching-time*) temporal logic CML (*Computational Mpms Logic*). CML is defined as a *monadic, third-order logic* with three equivalent versions. Two of them, $CML_{\Pi F}$ and $CML_{\Pi M}$ are based on *total order* models ΠM and ΠF respectively. The third version is an *extended partial-order* version $CML^{*}_{\Sigma M}$ based on the extension of ΣM with configurations and the state-vectors, that is equivalent to the former two versions. *Safety, liveness* properties can be expressed unambiguously at ease with all the three modalities of *past*, *present* and *future*. CML incorporates the newly introduced *branching space* aspect of *concurrency* as well as the conventional branching time aspect of *conflicts*. The improved expressiveness due to the former is explained.

Chapter-4 explains the model-checker algorithms to verify the properties of the input CFsm system expressed in $CML_{\Pi F}$. These formulae are transformed to corresponding $CML_{\Pi M}$ formulae over ΠM which in turn are viewed as $CML^{*}_{\Sigma M}$ ones, upon a correspondingly generated sum machine. The complexity of the model-checker involves the generation of ΣM and its *distributed, nested depth-first search of multiple Mpm-trees* for the verification of properties. The *deterministic algorithm* of the model-checker directly follows from the *functional* definition of notions such as *Minimal prefixes* and *configurations* as well as the *functional* definition of M_i *onto* F_i, for all i =1..n. Exponential complexity due to enumeration of all the *runs (maximal configurations)* and *interleavings* of every run is shown to be alleviated.

Chapter-5 presents the summary of the work, comparison with some of the related work, followed by the conclusions and scope of future work.

Chapter 2

Communicating Minimal prefix machines (CMpms)

The CMpms model proposed in this chapter is state-oriented but has the advantages of the traditional event-oriented models such as Petrinets. In other words, it alleviates the demerits of both the paradigms (state and event based) that are the impediments of an efficient verification system whose needs were outlined in Chapter-1on *introduction*.

We assume an input specification of a set of *n communicating sequential processes* represented as a corresponding set of *n Communicating Finite state machines* (CFsms). We transform this set of CFsms into a set of *n communicating concurrent machines* called *Minimal Prefix Machines* (CMpms) for the reason that will be clear in the sequel. In what follows, first of all the set of CMpms will be introduced as a set of *n* state-transition systems representing a set of *n* communicating processes.

We show that a set of n CMpms constitute a *partially-ordered, sum machine* as contrasted to their *totally-ordered, product machine* that is traditional. After their formal definition of CMpms and the sum machine, and a discussion of their salient features and properties through Lemmas and Theorems, CMpms with respect to a set of *n given input* CFsms will be introduced thus completing the development of the model from the given input specification.

2.1 Formal Definition of Communicating Minimal prefix machines (CMpms)

Minimal Prefix machines are developed to model *communicating processes* that progress *concurrently* and so each of them is defined as an element of a set, each representing a process, that communicates with the other elements of the set by *synchronization*. The following definition and Definition 2.2 that follows subsequently constitute the formal definition of CMpms.

Definition 2.1 An Mpm denoted M_i, $i=1..n$, is a *state-transition system* that is *possibly infinite* with *constraints* as follows:

$M_i = (S_{mi}, E_{mi}, R_{tmi}, s_{0mi})$ where,

S_{mi} is the set of *countable Mpm-states, possibly infinite*.

E_{mi} is the set of *countable, possibly infinite* events,

$R_{mi} \subseteq (S_{mi} \times S_{mi})$ is the binary *reachability relation* among the Mpm-states such that:

the *inverse relation* of R_{mi} denoted R_{mi}^{-1} is *constrained to be* a *function*, referred to as the *predecessor function*, defined for all states except s_{0mi}, the *initial-state*.

R_{mi} contains precisely one element for each $e_{mi} \in E_{mi}$ such that:

$L_{mi} : R_{mi} \dashrightarrow E_{mi}$ is a *bijection*, referred to as the *labeling function* which assigns an event *uniquely* to every element of R_{mi};

R_{tmi} is a *ternary transition relation*, derived from R_{mi} and L_{mi}:

$R_{tmi} \subseteq (S_{mi} \times E_{mi} \times S_{mi})$ such that,

For every transition $r_{tmi} = (s_{mi}, e_{mi}, s'_{mi}) \in R_{tmi}$,

$L_{mi}\,(s_{mi}, s'_{mi}) = e_{mi}$.

s_{mi} is said to be the *input state* and s'_{mi}, the *output state* of e_{mi}.

The *inverse* of L_{mi} is often referred to as the *I/O function* denoted: IO_{mi}.

$s_{0mi} \in S_{mi}$ is the *initial state* of M_i.

All events are considered *atomic* in the sense that they are executed *instantaneously*.

Since L_{mi} is a a *bijection*, and from the definition of *predecessor function* as the inverse of the *reachability relation* R_{mi}, it follows that the state-graph of an Mpm is restricted to be a *tree* that is free of *cycles*. As long as an event is ready to be executed from its input state, a *unique output state* may be produced.

In the case of non-terminating systems, as there is always some ready event to be executed from a given input state, there is an indefinite growth of an Mpm representing a non-terminating process and hence has an infinite state space and events.

Having defined Mpms, we need to define their communication aspect, as follows:

2.1.1 Communication among Mpms, to define CMpms

The Mpms communicate by *synchronizing* on certain common events among them. The above definition of Mpms along with the following one(s) constitute the formal definition of CMpms.

$sync_{in}$, $sync_{out}$ are each defined as *symmetric, binary* relations :

<u>Definition 2.2</u> $sync_{in}$, $sync_{out} \subseteq (S_{mi} \times S_{mj})$ $i, j = 1..n$, $(i \neq j)$ such that:

$(s_{mi}, s_{mj}) \in sync_{in}$ and $(s'_{mi}, s'_{mj}) \in sync_{out}$ iff :

$(s_{mi}, e_m, s'_{mi}) \in R_{tmi}$ and

$(s_{mj}, e_m, s'_{mj}) \in R_{tmj}$, $e_m \in E_{mi}$ and $e_m \in E_{mj}$. Then,

e_m is called a *synchronous event*,

s_{mi}, s_{mj} are *synchronous input* states and,

s'_{mi}, s'_{mj} are *synchronous output* states.

(s_{mi}, e_m, s'_{mi}), (s_{mj}, e_m, s'_{mj}) are called the synchronous transitions, that are referred to as *partner transitions* of each other. Similarly, s_{mi} and s_{mj} are *partner input states* of each other and s'_{mi}, s'_{mj} are *partner output states*.

In general, more than two Mpms may contain a given synchronous event. In that case, *every pair* of corresponding synchronous input states are related by $sync_{in}$ and every pair of corresponding output states by $sync_{out}$ relation.

Together, $sync_{in}$, $sync_{out}$ form *sync*:

$sync_{in}$ U $sync_{out}$ = sync

$sync_{in}$ relates states that are *exited simultaneously* at some instant of time before the synchronous event but they are *entered independently* of each other. On the other hand, $sync_{out}$ relates every pair of states that are *entered simultaneously* after the common synchronous event, but they are *exited independently* of each other.

Through out the sequel, *<u>the order in which the states are entered</u>* , i.e., their *entry order* is the one that is emphasized and captured rather than their *exit order*; the exit order among states is the same as the entry order of their respective *successors* i.e, the *output states* of the *I/O function* and so is redundant. This is because, the events are assumed to be atomic and so take place instantaneously. For instance, the *simultaneous exit* of synchronous input states is captured by the *simultaneous entry* of the corresponding synchronous output states. In this sense, $sync_{out}$ relation is emphasized more than $sync_{in}$ relation, in the sequel. This is the reason why only the synchronous output states entered together alone are glued together as illustrated in Fig.B. This point of contact is referred to as a *synchronization point* that represents the *simultaneous entry* of partner output states.

2.1.1.1 Sync_{out} /Simultaneity Relation is Equality

Sync_{out} relation , by virtue of relating two or more distinct states of multiple processes that are entered simultaneously, captures the *equality* subset of a *global, partial, causality order*, to be defined and is instrumental in defining the latter from the first principles. This simultaneity relation is not tractable if we were to order the events as opposed to states; for in the case of the events being related, there is no simultaneity among *distinct events* as a synchronous event is *identical* in all the participating processes and there is no way of telling a synchronous event apart from an asynchronous event. On the other hand, the output states of a synchronous event are *distinct and different* and so there is a scope to represent their *simultaneous entry*, if they were to be chosen as *primary entities* in the ordering.

Consequently, lot of modeling advantages follow in the case of ordering the state entries by keeping their *equality/simultaneity order* as a basis for many conceptual notions to be derived.

2.1.1.2 Initial Mpm-states are Simultaneous

Switching on or booting of a given system is considered as a *special start condition* when all the initial states s_{0mi},i=1..n of the Mpms are entered simultaneously at the same time, after the special synchronous event *init*, synchronizing all n Mpms. This idea is not only intuitive but also facilitates the mathematical treatment of the theory of Mpms to be put forth in the sequel.

So, we assume without contradicting any other ideas of the theory, that there is a transition denoted: $(Null, init, s_{m0i}) \in R_{tmi}$, of all the Mpms M_i, i=1..n respectively, where:

$(s_{0mi}\ sync_{out}\ s_{0mj})$, $\forall$ i , j= 1..n, i<>j.

Example 2.1 Fig. C of *Appendix*[1] shows a set of three state-trees of Mpms M_1, M_2 and M_3.

In this example,

---

[1] Fig.A, Fig. B, Fig. C and Fig.D are placed in *Appendix* for easy reference, as they are constantly referred to from many different sections of different chapters. Wherever Fig.C causes confusion, its equivalent representation Fig. D is referred to and vice versa. In fact, Fig.B, Fig.C and Fig.D are three different, equivalent representations of a CMPm system.

$R_{m1} = \{(a_0, b_0), (b_0,c_0), (d_0, a_1), (d_0,a_2)...\}$

The inverse of this relation viz., the *predecessor function* is easily verified to be a function (many-to-one) from the state-tree of M_1, as every state has its *unique predecessor.*

$R_{tm1} = \{...,(b_0, A_0, c_0), (c_0, C_0, d_0),...\}$

$L_{m1}(b_0, c_0) = A_0$, $L_{m1}(c_0, d_0) = C_0$

$IO_{m1}(A_0) = (b_0, c_0)$, $IO_{m1}(C_0) = (c_0, d_0)$.

The function L_{m1} is easily checked to be a bijection as every event is associated with a unique element of R_{m1}.

Example 2.2 Fig. B shows the set of three communicating Mpms, M_1, M_2 and M_3. The *synchronous output states* of different Mpms that are *partner states* of each other are shown glued together. The transitions of the three individual *state-trees* are drawn in three different styles to tell them apart. Though each of them is *infinite*, they are shown in a *truncated* form as will be explained in a future section.

$sync_{in} = \{(b_0, q_0), (q_0, b_0), (c_0, t_0), (d_0, v_0), (d_0, v_1), (v_1, g_1), (g_1, d_0), (u_0, z_0),...\}$

$sync_{out} = \{(a_0,p_0), (p_0, a_0), (a_0, x_0), (p_0, x_0), (c_0, s_0), (d_0,u_0), (r_0,h_0),...\}$

Note that $sync_{in}$, $sync_{out}$ are *symmetric* as mentioned in the definition.

The simultaneous entry of c_0 and s_0 for instance, that are tied together represents a *synchronization point*.

$sync = sync_{in}\ U\ sync_{out} = \{(b_0,q_0), (a_0,p_0),....\}$

We say that: $(b_0\ sync_{in}\ q_0)$, $(a_0\ sync_{out}\ p_0)$ etc.

Thus every pair of the *initial* Mpm-states s_{0mi}, $i=1..n$ are related by $sync_{out}$ and they together form the initial *synchronization point* after their simultaneous entry after *init*, as discussed.

2.1.1.3 Causality, the Global Dependency-order among Mpm-states

We formally link the n Mpms by the $sync_{out}$ relation, which represents *simultaneity* by relating the states entered at the same time after executing the synchronous event. We perform the following union and the reflexive, transitive closure to create the *global depen-*

dency-order among the states of all Mpms, which is in general a *partial-order* that is *reflexive* and *transitive*.

The global (intra and inter) dependency-order often referred to as *causality* is defined as:

Definition 2.3

Causal Dependency-order $(\leq) \subseteq (S_{mi} \times S_{mj})$

$:= (R_{m1} \; U \, R_{m2} \ldots U \, R_{mn} \; U \, sync_{out})^{+}$

where the *superscript* '+' stands for the *reflexive* , *transitive closure* of the union of the *binary relations* $R_{mi}, i=1..n$ *and* $sync_{out}$.

The subset $(sync_{out} \; U \; id)$ of the above closure represents the *equality* relation '=' ;

The *id* function (from reflexive closure) is added for manipulative convenience in order to extend $\leq$ to relate the state-vectors to be introduced at a later section.

The definitions of $\leq$ and = relations can be extended to define < and > relations as well:

< is defined as the difference between $\leq$ and = relations:

$< := \leq \; - \;$ '=' and,

> as the *inverse* of < relation:

$> := (<)^{-1}$

The binary causal-order that forms a *derived, global, partial-order* (PO) among Mpm-states is the basis of the *'sum machine'*, to be defined shortly in the sequel.

2.1.1.4 Significance of *State Order* Versus *Event Order*

This issue was touched upon in a previous section. The causal dependency-order or causality that is global and partial is defined and centered around Mpm-states rather than the events being related as the entities of the PO, as in many models of the literature.

Many models of event-structure assume $\leq$ as *granted*, while in our case it is a *derived* notion. It is derived from the *equality/simultaneity* of output states that follows every synchronous event. In other words, it is derived from the *physical communication mechanism* of the *concrete domain* rather than a granted notion in the *abstract domain*.

When a synchronous event takes place, it is different from an asynchronous one in the sense that every participating process executes a replica of the synchronous event. But this information that might prove vital from modeling point of view, is not recorded at all in the case of models relating only the occurrences of events' execution. On the other hand, when we relate the states by their entry order, we are in a position to account for and model the simultaneity of two or more *distinct states*, one from every participating process, that hold right after the synchronous event.

Since states directly carry certain *propositions* which become true as soon as they hold, relating the states and hence their predicates carries a lot of value in terms of : the modeling capability, development of the logics and in algorithmic application.We prove these in the course of development of the rest of the theory in the current and subsequent chapters.

It is to be noted that in the sum-machine, there are as many events as there are states (excepting the set of roots of the Mpm-trees) and since every *state-entry* is followed by *an event-occurrence*, the same partial-order of causality among states can be extended to define that among events as well, and in this sense both the entities viz., states and events are exact *duals* of each other and completely accounted for. But, for our current application viz., *verification of properties*, we only require the PO *among states* as explained in the last two paragraphs.

Example 2.3 For illustration, let us consider Fig. B again for the following sample of elements in each of the above relations:

$\leq$:= $\{(a_0, p_0), (p_0, x_0), (x_0, a_0), (a_0, b_0), (b_0, c_0), (c_0, s_0), (c_0, t_0),(c_0, d_0),$

$(x_0, y_0), (q_0, d_0),(b_0, s_0),(b_0, u_0),........\}$

$=$:= $\{(a_0, p_0), (p_0, a_0), (p_0, x_0), (x_0, a_0), (c_0, s_0), (s_0, c_0), (t_0, z_0),$

$(d_0, u_0),(a_0, a_0), (b_0, b_0), (y_0, y_0), (v_0, g_0),....\}$

$sync_{out}$ is *symmetric* relation and so is the *equality* relation $=$.

$< := \leq - =$

$:= \{(a_0, b_0), (c_0, d_0),(x_0, y_0),(b_0, c_0),(q_0, s_0),(q_0, d_0),$

$(b_0, s_0),(b_0, u_0),........\}$

$> := <^{-1} := \{(b_0, a_0), (d_0, c_0),(y_0, x_0),(c_0, b_0),(s_0, q_0),(d_0,q_0),$

$(s_0,b_0),(u_0,b_0),........\}$

The states that are *later* in the order of $\leq$ are referred to as the *descendents* of the states that are *earlier* in that order, which are the *ancestors*. The immediately following descendent is the *successor* and the preceding ancestor, the *predecessor*. This terminology is adopted with respect to the local transition relations R_{mi},i=1..n as well as the ones to be defined among vectors. The states that are *equal* are either *identical* or *distinct partner output states* of each other.

2.2 Certain Theoretical Basics of Mpms

The $\leq$ relation defines a *'necessarily-entered-before-or-together'* relation, that forms the back bone for most of the notions to be developed in CMpms theory. Certain basics of the theory are in order:

All the *n* Mpms have their own respective *clocks* controlling the speeds of execution of their local events. In every Mpm, all events are *atomic* in the sense that their execution is instantaneous, when ready. We also refer to the execution of an event as the *happening* of an event. When an event happens in a given Mpm, its *locus of control* transits from its current state to the next state; we say that the current state is *exited* and the next state is *reached* or *entered*, synonymously. When we say that '*the state holds*', it means that the locus of control of the given Mpm resides at that state, from the instant it was entered till the time it is exited when an event of the next state transition takes place.

All the *n* Mpms are tied together i.e., dependent on each other through *synchronization* or the *causal dependency-order* $\leq$ derived therefrom. Even though the *n* loci of control are essentially dependent on their respective local clocks, these loci are also dependent on each other since they *must wait for each other* at the synchronization points, viz., the synchronous events and states as per the relation $\leq$. Since the *equality* relation (=) comes from synchronous states entering together, it is appropriate to refer to $\leq$ as: '*necessarily entered before or together*' relation.

Even though there are *n* different *clocks* controlling n different *local time* scales/loci of control, there is only one *global, 'real time'* scale, onto which events of all the above *n* local time scales may be *projected* when the system of *n* Mpms execute or *run* actually. Depending on the *relative speeds* of execution of the *n* local clocks, there could be many *possible projections* onto the *real time scale* during a concurrent execution of n Mpms. Of course all these possible projections must obey the dependency-order, $\leq$. We refer to each of these possible projections as an *interleaved observation* or simply an *interleaving* within a given *execution/run* of n Mpms. These concepts will be formally presented towards the end of this chapter.

In summary, with the causal dependency-order (inter and intra) respected by all the n loci of control of Mpms, there could be many different possible orders of execution of their states and events that are not dependent on each other and so unrelated by $\leq$, in a given run of n Mpms in *real time*. Informally, the *looser* the dependency-order $\leq$ (i.e., the lower its cardinality), the looser is the *coupling* among Mpms, the more will be the number of independent (asynchronous) states and events and the larger will be the number of possibilities of ordering them, which are in other words the number of *projections/interleavings* mentioned in the last paragraph. Similarly, the more the number of synchronization points, the *tighter* the dependency-order (the higher its cardinality), the tighter the coupling, and the fewer are the number of such projections.

2.2.1 Primary versus Secondary Mpms

The individual Mpms are capable of executing independently of other Mpms except during synchronization events. But during their generation and application for verifying the system properties (as will be discussed in Chapter-4), they are traversed one after the other except when blocked by synchronization requirements.

In other words, the Mpms are *simulated* for practical purposes, in such a way that at a given time, only one of them is traversed exhaustively and the others are allowed to make only a restricted progress as much as necessary to satisfy the synchronization requirements of the former. The formerly mentioned Mpm is referred to as the *primary Mpm* and the latter, as the *secondary Mpms*.

The above simulation is needed in order not to lose track of any of the reachable state vectors or the global-states of Mpms. At any given time, the state of the primary Mpm represents the *present* and those of the secondary ones the *past or the present* with respect to the former state. This and the concepts of the previous section will be formalized in the following sections.

By using the causal dependency-order $\leq$, we can define the following composition of the set of CMpms, M_i,i=1..n referred to as the *sum machine*, denoted as ΣM:

Definition 2.4 : $\Sigma M := \Sigma_{i=1..n} M_i := (\Sigma S_{mi}, \Sigma E_i, \leq , \Sigma s_{0mi})$, where

Σ denotes the *disjoint-union* from i=1..n of every entity.

The composition of the dependency-order $\leq$ is not a rigid one since by taking away *sync*$_{out}$ component from its constituents, we get back ΣR_{mi},i=1..n and so the set of Mpms M_i, i =1..n. Thus the sum machine consists of the *disjoint union* of *Mpms*, with all their *partner output states* tied together according to synchronization requirements. The only difference between the *sum-machine* ΣM and the set of CMpms M_i,i=1..n is the *enriched global causality* $\leq$ of the former composed from R_{mi},i=1..n and $sync_{out}$ relations of the latter. Hence the notation ΣM denotes the sum machine, emphasizing the *disjoint-union*[1] of its component machines.

The sum machine is a *partially-ordered machine/state-transition system* by virtue of its *partial* dependency-order $\leq$ among the (Mpm-)states.

Example 2.4 Fig. B of *Appendix* illustrates the *sum machine* ΣM of M_1, M_2 and M_3 introduced earlier in which,

$s_{0m1} := a_0$, $s_{0m2} := p_0$ and $s_{0m3} := x_0$, all three of which are glued together to represent the *initial synchronization point*, considered to be entered *simultaneously* after the special synchronous event, *init*;

$(c_0\ sync_{out}\ s_0)$, $(t_0\ sync_{out}\ z_0)$, $(d_0\ sync_{out}\ u_0)$, $(r_0\ sync_{out}\ h_0)$, $(v_0\ sync_{out}\ g_0)$,

$(s_1\ sync_{out}\ x_3)$, $(a_2\ sync_{out}\ p_2\ sync_{out}\ x_2)$, $(a_1\ sync_{out}\ p_1\ sync_{out}\ x_1)$ are other pairs and 3-tuples that represent *simultaneous entries* of partner output states and hence the synchronization points that are used to form the global dependency-order $\leq$ in the sum machine, $\Sigma_{i=1...3} M_i$.

---

[1] *Notation convention* :

Throughout the sequel, the symbol Σ continues to denote the '*disjoint union*' of sets.

2.4 Sequence, Conflict and Concurrency in ΣM

Sequence, *conflict* and *concurrency* are the three fundamental binary relations in addition to *causality*, among the local states of the *n communicating processes* in any concurrent system in general, which in particular correspond to the states of *n* Mpms in this context. We define these as binary relations in terms of the local reachability relations R_{mi}, $i=1..n$ and the global *causal dependency-order*, $\leq$. Sequence and conflict originate among local states and are inherited by the non-local ones through the dependency-order. Concurrency is basically a global relation since it relates only non-local states.

2.4.1 Sequence in ΣM

Sequence is a *stronger* relation than the *causal* dependency-order $\leq$. The latter relates two states when one is *entered* before (or together with) the other. Whereas, the two states are in *sequence* only when the one entered before also has to *exit* in order to enable the *entry* of the other.

Definition 2.5 : $seq \subseteq (S_{mi} \times S_{mj})$ i, j = 1..n.

$(s_{mi}\ seq\ s_{mj})$ iff:

$\exists s'_{mi}: (s_{mi} R^{+}_{mi} s'_{mi}) \wedge (s'_{mi} \leq s_{mj})$ where R^{+}_{mi} is the transitive closure [6] of R_{mi}, representing the *reachability relation* among states within an Mpm-tree.

seq is an irreflexive and transitive relation. It is a global relation which locally degenerates to the reachability relation R^{+}_{mi} , since < reduces to R^{+}_{mi} within a given Mpm M_i.

2.4.1.1 Sequentiality Versus Causality among Mpm-states

Both the relations *sequence* and *causality* are derived global relations of *synchrony/simultaneity* i.e., the *equality* relation and *local reachability* relations: sequence extends the local reachability relation globally through equality. It is the converse in the case of causality: it extends equality globally through local reachability relations.

To define sequence, we first start from $(s_{mi} R^{+}_{mi}\ s'_{mi})$. Then, this relation is extended by relating s_{mi} to all the states that are related to s'_{mi} through equality or in general causality itself. In this sense, sequence is a stronger relation than causality.

To define causality, we first start from an element of *equality* relation not in *id* say, ($s_{mi} = s_{mj}$). Then this is extended by local reachability relations relating s_{mi} and s_{mj} respectively with their local descendents. The exact definition of causal dependency-order is given previously.

The basic relations in both the cases are equality and local reachability relations.

(s_{mi} *seq* s_{mj}) implies that s_{mi} should be *exited* to allow the entry of s_{mj}. In other words, at least one transition of R_{mi} is involved in reaching s_{mj} from s_{mi}. On the other hand, ($s_{mi} < s_{mj}$) not necessarily means (s_{mi} *seq* s_{mj}), though it could well be so. It only implies s_{mi} should be *entered* before the entry of s_{mj} (a transition is not necessarily involved requiring s_{mi} to be exited before the entry of s_{mj}). This leads us to draw the following conclusion.

Example 2.5 Referring to Fig. B again, it is true that (c_0 *seq* d_0) but not (s_0 *seq* d_0). This is because, as explained before, there is no transition taking place between s_0 and d_0 as opposed to c_0 and d_0 in which case, unless c_0 is exited d_0 can not be entered; s_0 only inherits the causality between c_0 and d_0 from its equality with c_0 i.e., $c_0 = s_0$. Therefore ($s_0 < d_0$) is true and not (s_0 seq d_0) as the two states s_0 and d_0 can *co-exist* in M_1 and M_2 respectively.

2.4.2 Conflicts in ΣM

Conflict, to be formally defined below, means *true choice*, as exhibited by the given concurrent system. The true choice/conflict is contrary to the *non-deterministic choice* arising out of the *different execution orders* of the component Mpms in this context, of the system. Therefore nondeterministic choices are *artificially* imposed.

We define a *local conflict* relation as an *irreflexive* and *symmetric* relation:

Definition 2.6 (s_{mi1} $conf_i$ s_{mi2}) iff:

$\wedge(s_{mi1} R^*_{mi} s_{mi2} \vee s_{mi2} R^*_{mi} s_{mi1})$ where, $\wedge$ is the *complement* operator and R^*_{mi} , the *Kleene closure* [6] of R_{mi}.

Local conflict is thus the complement of the local sequential/reachability relation.

A *global conflict* relation is derived from the above local definition:

$$conf \subseteq (S_{mi} \times S_{mj})\ i, j = 1..n$$

Two states s_{mi}, s_{mj} of M_i, M_j respectively are in conflict denoted:

(s_{mi} *conf* s_{mj}), which is deduced from the following *equivalence*, referred to as *conflict-inheritance* property:

2.4.2.1 Global Conflicts are propagated Local Conflicts

Following is an important property that defines global conflicts as the ones originating locally within Mpms and propagated globally among non-local states through *causality*. This property implies that by maintaining the local conflicts alone of a given Mpm-state and a minimal set of those that are causally dependent on it, all the non-local ones in conflict with it can be deduced without *enumerating* all of them.

This property is exploited in verification by scanning only the 'local neighborhood' of Mpm-states, as will be explained in Chapter-4.

Property 2.1 (s_{mi} $conf_i$ s'_{mi}) <=> (s_{mk} *conf* s_{mj}), $\forall$ s_{mk}, s_{mj}: $s_{mi} \leq s_{mk}$, $s'_{mi} \leq s_{mj}$, where $i,j,k = 1..n$.

All conflicts thus originate locally within an Mpm and are propagated globally through synchronization embedded in the dependency-order, $\leq$. For instance, all the states that are in local conflict with a synchronous output state are also in global conflict with the latter's partner output states.

Example 2.6 From Fig. B again,

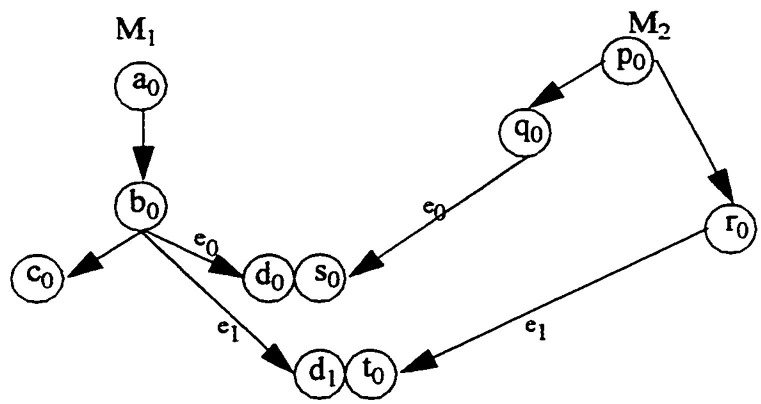
(t_0 *conf* x_4) follows from (z_0 $conf_3$ x_4).

2.4.2.2 Induced Local Conflicts

It is also the case that local conflicts in one Mpm are inherited/manifested as local conflicts of another Mpm. A *synchronous input* state s_{mi_in} can be a *partner state* of two different synchronous input states that are in local conflict, belonging to another Mpm. In this case, two different pairs of synchronous transitions result, both with s_{mi_in} as input state.

Example 2.7 Let us consider Fig. 4 below. It shows two Mpms M_1, M_2 in which state b_0 of M_1 is a *synchronous input state* with two different *partner input states* q_0 and r_0 synchronizing respectively on the synchronous events e_0, e_1. The corresponding synchronous output states of e_0, e_1 respectively of M_1 are d_0 and d_1 which are in local conflict. This is essentially the manifestation of the local conflict between q_0 and r_0 of M_2. This is a case of induced local conflict.

Fig. 4 Example for *induced local conflicts*



The induced local conflicts of Mpms will be associated with *non-deterministic synchronization of true choices* of input Fsms in a future section, which may in general lead to exponential enumeration of local conflicts, due to this non-determinism.

All the elements of the *disjoint union* of the local conflict relations $\Sigma conf_i$, i=1..n are the ones explicitly represented in ΣM, and the derived ones of *conf* relation are implicit.

It is to be noted that the relation *conf* like *seq*, disallows the related states to be holding simultaneously i.e, at the same time; the relation *seq* ensures that one state holds only in the *past* of the other, while *conf* is stricter than that: it means that one state can not even hold in the *past* or in the *future* of the other, and vice versa.

2.4.2.3 Backward Conflicts in ΣM

Definition 2.7 Two states s_{mj}, s_{mk} are in *backward conflict* iff:

$\exists s_{mi}$: $(s_{mj}\ conf\ s_{mk}) \wedge (s_{mj}\ seq\ s_{mi}) \wedge (s_{mk}\ seq\ s_{mi})$.

Backward conflict Lemma:

Lemma 2.1 There are no backward-conflicts in ΣM

Proof: (By contradiction)

Let $\exists s_{mi}$: (s_{mj} *conf* s_{mk}) $\wedge$ (s_{mj} *seq* s_{mi}) $\wedge$ (s_{mk} *seq* s_{mi}) where,

(s_{mj} *conf* s_{mk}) => (s_{mj} $conf_j$ s'_{mj}) $\wedge$ ($s'_{mj} \leq s_{mk}$).

=> $\exists s_{mi}$: (s_{mj} $conf_j$ s'_{mj}) $\wedge$ (s_{mj} *seq* s_{mi}) $\wedge$ (s'_{mj} *seq* s_{mi})

=> $\exists s_{mi}$: (s_{mj} $conf_j$ s'_{mj}) $\wedge$ (s_{mj} R^+_{mj} s''_{mj}) $\wedge$ (s'_{mj} R^+_{mj} s''_{mj}) such that: $s''_{mj} \leq s_{mi}$

=> s''_{mj} has more than one predecessor, a contradiction of the definition of state-tree of an Mpm at Definition 2.1.

■

The above lemma, Lemma 2.1, can be proved alternatively as follows:

Locally within the state-tree of an Mpm, backward conflicts are avoided by the state-tree formation. In the definition of backward conflicts, the relations *seq*, *conf* are ΣR_{mi}, $\Sigma conf_i$ respectively extended globally through *simultaneity* (of synchronous output states) . So, what we need to ensure is that, the above extension does not introduce backward conflicts. In other words, we need the following property.

2.4.2.4 Uniqueness of (Synchronous) Partner Transitions

Property 2.2 A synchronous output state is associated with a unique synchronous transition and thus has a unique set of (possibly a singleton) *partner states*.

Proof: (By contradiction)

Let s_{miout} be a synchronous output state that has two sets of partners $\{s_{mjout}\}$ and $\{s'_{mjout}\}$ after a synchronous event e_m and e'_m respectively.

=> (s_{miin} e_m s_{miout}), (s'_{miin} e'_m s_{miout}) are the two synchronous transitions with output state s_{miout} with partner transitions (s_{mjin} e_m s_{mjout}), (s'_{mjin} e'_m s'_{mjout}) respectively.

=> if $s_{miin} <> s'_{miin}$, it is a contradiction of unique predecessor of s_{miout} according to the *predecessor function* of an Mpm. When $s_{miin} = s'_{miin}$, it is a contradiction of the definition of *I/O function* as *bijection*, since both e_m, e'_m are mapped to (s_{miin}, s_{miout}).

Therefore, s_{miout} must be associated with a unique transition.

■

Conceptually, by restricting every synchronous output state to have a *unique* set of partner states (by associating a unique transition with it), we make sure that the extension of $conf_i$ to *conf* and R^+_{mi} to *seq* by means of $\leq$ in the definition of *backward conflicts*, ensures their absence globally (in ΣM) as well as within an Mpm, since backward-conflicts violate the *tree* formation of the state-graph of an Mpm.

Essentially, by avoiding backward conflicts, we associate a *unique past* (by the unique predecessor property applied cumulatively) and a *unique present* (by the unique set of partner states in the case of synchronous output state, along with the *unique past* of each of them that comes with it) with every state.

Disallowing backward conflicts makes the modeling of *sequence, conflict* and *concurrency* all at the same computational level and hence easier, and gives rise to algebraic or manipulative convenience of its entities without taking away the expressiveness of a given specification. This will be made progressively clearer in the course of development of the model.

Property 2.3 Two Mpm-states s_{mi} and s_{mj} can not be related by *conflict* and *causality* relations at the same time. i.e., (s_{mi} *conf* s_{mj}) is in contradiction with ($s_{mi} \leq s_{mj}$).

Proof: This property follows from the absence of backward-conflicts.

Let us assume, both (s_{mi} *conf* s_{mj}) and ($s_{mi} \leq s_{mj}$) are true.

(s_{mi} *conf* s_{mj}) => $(s_{mi}\ \mathrm{conf}_i\ s'_{mi}) \wedge (s'_{mi} \leq s_{mj}) \vee (s_{mj}\ \mathrm{conf}_j\ s'_{mj}) \wedge (s'_{mj} \leq s_{mi})$...(i)

Let us assume, $(s_{mi}\ \mathrm{conf}_i\ s'_{mi}) \wedge (s'_{mi} \leq s_{mj})$ is the case in (i) above.

=> $(s_{mi} \leq s_{mj}) \wedge (s'_{mi} \leq s_{mj})$ where $(s_{mi}\ conf_i\ s'_{mi})$

=> (a)Both s_{mi} and s'_{mi} have a *same synchronous partner state* s'_{mj} such that: (s'_{mj} *seq* s_{mj}) or,

(b) both s_{mi} and s'_{mi} have a common descendent s''_{mi} such that $s''_{mi} \leq s_{mj}$.

(a) contradicts Property 2.2 and both (a) and (b) imply the presence of backward conflicts, a contradiction of Lemma 2.1.

Similarly, the disjunctive case $(s_{mj}\ \mathrm{conf}_j\ s'_{mj}) \wedge (s'_{mj} \leq s_{mi})$ can be assumed and the proof is similar.

▩

2.4.3 Concurrency in ΣM

The binary concurrency relation *co* among disjoint Mpms is defined as follows:

Definition 2.8 $co \subseteq (S_{mi} \times S_{mj})$ *where* $(i \neq j)$

(s_{mi} *co* s_{mj}) iff s_{mi}, s_{mj} are *unrelated* by *seq* or *conf*.

The *co* relation is therefore the *complement* of (*seq* U *conf*). This makes sense conceptually as well because, both *seq* and *conf* imply that the related states *cannot co-exist* at the same time, while concurrency does the opposite, or the complementary condition.

In other words, the union (*seq* U *conf* U *co*) is a *total* binary relation, relating states of ΣS_{mi}, $i=1..n$, that is irreflexive.

2.4.3.1 Concurrency is not 'Unorder'

It is to be noted that concurrency is defined not as a *complement* of the *causal order* but as that of *sequence* and *conflict*. By so doing, we make room for both the relations *co* and $\leq$ to *possibly co-exist among the same pairs of states*. This indeed will be the case because, the non-local states related by *equality* are automatically in causal order; and since they are neither in *sequence* nor in *conflict*, must be concurrent as well.

The advantage of the above idea is two-fold:

(i) The fact that *concurrency* is defined independent of *causality* is exploited in *labelling* every Mpm-state with a *concurrent state-vector* (whose states are all pairwise concurrent) which has at least n pairs of states related by causality as well. Thus the partial order $\leq$ among Mpm-states becomes a *labelled PO*, with each state having, details of which will be elaborated in a sequel section. The labeled information will be exploited in the verification algorithm of Chapter-4.

(ii) Because concurrency is the complement of the union of sequence and conflict, all the three basic relations are included in the '*universe*' or the same level of execution, (as opposed to the case of concurrency being the complement of causality whence conflict is pushed out of the process semantics), a much sought after goal in modeling concurrent systems.

Example 2.8 For instance, we consider states related by *co* relation from Fig. B of *Appendix*:

(c_0 *co* s_0), (d_0 *co* s_0).

The *co* relation is *irreflexive* and *symmetric* but not necessarily transitive. Two sequential states from a given Mpm could both be concurrent with a third state of a different Mpm.

For instance, back in Fig. B, (c_0 *co* s_0) and (s_0 *co* d_0) but (c_0 *seq* d_0).

All the three relations viz., *seq*, *conf* and *co* manifest globally and their union is a *total* relation among all the Mpm-states, through the underlying global, dependency-order $\leq$, that is *partial*.

2.4.3.2 The Paradox of Concurrency in ΣM

ΣM is a state-oriented model, and so all the entities are primarily defined with respect to its (Mpm-) states. Concurrency is no exception.

Two (or more) states of different Mpms are said to be concurrent if it is *possible* that they both *may hold* at some point of time in their respective Mpms. Informally, they co-exist at the same time.

The *paradox* stems from the following two orthogonal views of concurrency:

- When two or more transitions of different Mpms can take place *independently*, *asynchronous* of each other, then the corresponding output states are said to be concurrent, since they *may co-exist* (it is noted that it is not a *must*) i.e., they may hold at the same time.
- On the contrary, when different Mpms participate in a common *synchronous event*, there is such a *strong dependence* among them that the common event have to be executed *simultaneously* and the corresponding synchronous output states are *entered simultaneously* after the common synchronous event and they *must co-exist*. These output states are concurrent as well.

Therefore concurrency is attributed to both *independence/asynchrony* and *dependence/synchrony* of states (and events) of different Mpms at once. This is the manifestation of *the* above mentioned *paradox*. The logical explanation is that: concurrency is first *originated by/sourced out of synchrony* or strong dependence as *simultaneous* synchronous output states and then become prolific or multiply by *asynchrony* among Mpms. Therefore, it makes more sense to represent strong concurrency (than not, as in many models) along with concurrency; as after all, the former seems to be the *basis* of the latter and not *vice versa*.

The *sync* relation represents *'strong-concurrency'* since the two related states *must hold* at their respective Mpms at some point of time, *before* or *after* the happening of the synchronous event concerned, as the case may be.

2.4.3.3 Synchronization: the Controlling Medium of Concurrency and Causality

Both concurrency and causality are triggered and controlled by the synchronization that is followed by the simultaneity of Mpm-states in the following sense:

The synchronization points source the *'threads' of concurrency*, as many as the number of participating processes which later 'grow' or progress asynchronously of one another, sustaining the concurrency among their local states till the next synchronization point, when the processes are forced to *wait for each other;* this is followed by the growth of threads again, as above. At every synchronization point, the processes participating are *controlled* or *regulated* to wait for one another and after the synchronization, the respective threads of the processes are set to progress asynchronously. In this sense, synchronization is said to be the *source* of simultaneity (and hence causality) and the controlling medium/agent of concurrency as explained above.

The important point to note here is that unlike in event-structures of many *behaviour models* where *concurrency* and *causality* are complementary, in our model both are not disjoint as both are controlled by the simultaneity relation (due to synchronization). There is no *analogous* simultaneity relation for events since synchronous events of participating processes are *identical* and hence do not convey any more relational information than asynchronous ones, as already mentioned. Concurrency is viewed as *unorder*, complement to the causal order, $\leq$ in these event-based models.

The above fact has an important consequence: The process based semantics of event based models rule out conflicts from a process due to the fact that concurrency and causality (which are complementary) solely make up the *universe* by their union.

The crux of the thesis lies in the fact that causality and concurrency are not complementary. Causality is first derived based on synchronization and local reachability relations of the concurrent automata (CMpms) and then applied to derive all the three relations viz., *sequence*, *conflict* and *concurrency* in the same basic and global level of computation of the sum machine.

Example 2.9 For instance from Fig. B,

(d_0 $sync_{out}$ u_0) is a synchronization point which represents the origin of concurrency by way of its synchrony. The respective local descendents of the two states d_0 and u_0 that are reached asynchronously after local transitions according to R_{m1} and R_{m2} respectively are concurrent as well.

Thus, (d_0 $sync_{out}$ u_0) => (d_0 *co* u_0)

($d_0 = u_0$) and (u_0 R_{m2} v_1) => ($d_0 < v_1$), from the equality of $sync_{out}$ and the definition of <.......(i)

(d_0 *co* u_0) and (u_0 R_{m2} v_1) => (d_0 *co* v_1), from the *asynchrony* of v_1 with respect to d_0.........(ii)

From (i) and (ii) as a consequence of above *paradox*, we have both the results:

($d_0 < v_1$) and (d_0 *co* v_1)

The important result below (as explained in a previous section) follows which is claimed as the *crux* of the entire work:

- *Concurrent states are not necessarily unrelated by causality* ($\leq$), although it is true that the states unrelated by $\leq$ are concurrent, assuming that they are not in conflict.

The above result will be stated and proved formally at a later section. The added advantage of explicit *sync* relation (in comparison with the other *partial-order* models in which *sync* is transparent) is that, *concurrency* and *strong concurrency* are distinguished. The former is *possible* concurrent holding of states of different Mpms and the latter is *necessary* concurrent holding of the states since they either are *entered* together or *exited* together. Because *sync* relation is a subset of both *causality* and *co* relations, these two relations *co* and $\leq$ have *non-null* intersection (with the intersection containing elements in addition to those of $sync_{out}$ relation due to the transitivity of $\leq$ relation. An example was shown already). This result has an important application in the following:

If a global-state is reachable in one interleaving and if its local components *wait for each other*, then the global-state is reachable by all interleavings thus avoiding the enumeration of all the *non-deterministic interleavings* and the consequent state explosion. The *possible* and *necessary* holding of concurrent states enable us to define the interleaving operator and deduce the property of all interleavings from one. This feature will be expanded in a future section and subsequent chapters as well.

2.4.3.4 Concurrent transitions

Two transitions (possibly synchronous) $r_{tmi} = (s_{mi}, e_{mi}, s'_{mi})$ and $r_{tmj} = (s_{mj}, e_{mj}, s'_{mj})$: $i <> j$, $r_{tmi} \in R_{tmi}$, $r_{tmj} \in R_{tmj}$ are said to be *concurrent* iff:

$(s_{mi}\ co\ s_{mj})$ and $(s'_{mi}\ co\ s'_{mj})$.

When r_{tmi} and r_{tmj} are synchronous, partner transitions, $e_{mi} = e_{mj}$ and they take place simultaneously. The input and output partner states are related by the stronger *sync* relation than *co*.

2.5 The Product machine ΠM

We define the following conventional composition of M_i, i=1..n, to give rise to the familiar '*product machine*' as:

Definition 2.9 $\Pi M := \Pi_{i=1..n} M_i := (S_m, E_m, R_{tm}, s_{0m})$ where Π denotes the product of the n components.

$S_m \subseteq (S_{m1} \times S_{m2} \times \ldots \times S_{mn}), E_m = \cup E_{mi}, i=1..n$

$s_{0m} = (s_{0m1}, s_{0m2}, \ldots, s_{0mn})$ and

the *transition relation* R_{tm} is defined as follows:

$\forall\ s_m, s'_m \in S_m, e_m \in E_m, (s_m, e_m, s'_m) \in R_m$ iff :

$\exists\ i: (s_{mi}, e_m, s'_{mi}) \in R_{tmi}$ where $s_{mi}, s'_{mi} \in S_{mi}, e_m \in E_{mi}$

$\wedge$

$\forall j <> i$:

$(s_{mj}, e_m, s'_{mj}) \in R_{tmj}$, if $(s'_{mi}\ sync_{out}\ s'_{mj})$ and $e_m \in E_{mj}$

$s_{mj} = s'_{mj}$, otherwise.

The *reachability relation* R_m (used more often than R_{tm}) similar to R_{mi} $i=1..n$ is defined as:

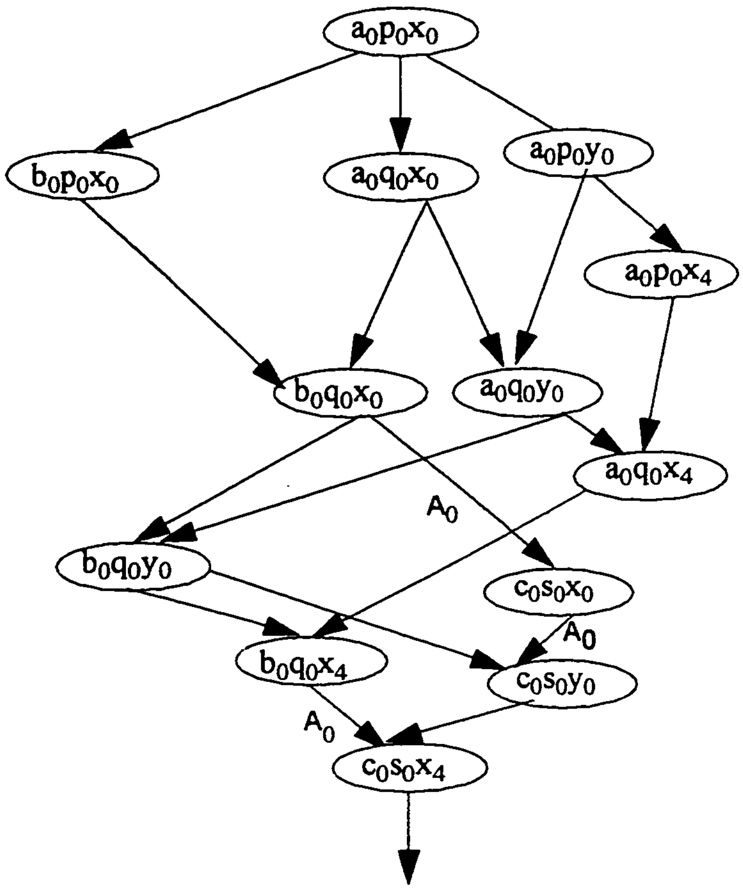
$(s_m, s'_m) \in R_m$, iff $(s_m, e_m, s'_m) \in R_{tm}$.

Global-state:

Since the states $s_m \in S_m$ of ΠM are composed of the Mpm-states, as reachable vectors, they are referred to as *global-states*, as opposed to Mpm-states that are local to Mpms.

Example 2.10 The product composition of M_1, M_2, M_3 illustrated in Fig. D of *Appendix* is shown below:

Fig. 5 The product machine ΠM of M_1, M_2 and M_3 of Fig. D



The initial state of the product machine is: $(a_0p_0x_0)$.

States are related by R_m as for instance:

$((a_0p_0x_0)\ R_m\ (a_0q_0x_0))$, $((b_0q_0x_0), A_0, (c_0s_0x_0)) \in R_{tm}$, etc.

2.5.1 Sequence and Choice in ΠM

Sequence and choice are the two relations used to compare every pair of global-states of ΠM.

2.5.1.1 Sequence in ΠM

The sequence relation seq_g is defined as the transitive closure of the reachability relation R_m:

Definition 2.10 $seq_g = R_m^+$.

If (s_m R_m s'_m), s_m is said to be the *predecessor* of s'_m which is the *successor* of s_m.

If (s_m seq_g s'_m) , a *sequence* of states from s_m to s'_m is said to be a *path* of ΠM, with s_m and s'_m as the *initial* and *final* states of the path respectively. s'_m, the final state of the path is said to be the *descendent* of s_m, and s_m is said to be the *ancestor* of s'_m.

There could be more than one path from a given initial state to a final state.

For example from Fig. 5 above, both the following paths:

(($a_0p_0x_0$), ($a_0q_0x_0$), ($b_0q_0x_0$), ($c_0s_0x_0$), ($c_0s_0y_0$)) and,

(($a_0p_0x_0$), ($b_0p_0x_0$), ($b_0q_0x_0$), ($b_0q_0y_0$), ($c_0s_0y_0$)) have

($a_0p_0x_0$) and ($c_0s_0y_0$) as their *initial* and *final* states respectively.

2.5.1.2 Choice in ΠM

Choice is a complementary concept of sequence in ΠM. It is a binary, symmetric and irreflexive relation among states. If two states are not related by seq_g, then they are related by *choice* relation.

Definition 2.11 : (s_m *choice* s'_m) iff ^(s_m seq_g s'_m ∨ s'_m seq_g s_m) where ^ denotes the complement operator.

2.6 Analysis of ΠM with respect to ΣM

Even though ΠM is the product machine composed in a traditional manner, each of its states is formed as a vector of Mpm-states. So, the pre-existing *structure* of Mpm-states, in particular, the *sequence, choice* and *concurrency* relations among them defined with respect to ΣM cannot but be carried on to ΠM as well.

We proceed further to analyse ΠM with respect to ΣM and see if more light can be thrown on the former thereby.

2.6.1 Global-state Theorem

Theorem 2.1 Every pair of the *n* components of a state of ΠM are related by the concurrency relation *co*, and vice versa.

i.e, $s_m \in S_m$ of ΠM <=> (s_{mi} *co* s_{mj}) $\forall$ i, j =1..n, i <> j.

Proof:

=>part:

Any two Mpm-states have to be related by either *seq* or *conf* or *co* by the property of their union being a *total* relation.

Let (s_{mi} *seq* s_{mj}). It means that s_{mj} can not be entered unless and until s_{mi} is exited; which in turn means that they can not *hold simultaneously* (at the same time) and in other words, they cannot co-exist to form a state of ΠM.

By the same token, (s_{mi} *conf* s_{mj}) can not be true either.

Thus, (s_{mi} *co* s_{mj}) $\forall$ i, j =1..n, i <> j.

<=part:

Since *co* is the complement of (*seq* U *conf*), s_{mi} and s_{mj} could hold simultaneously which means all *n* components can form a reachable vector or a state of ΠM.

■

2.6.2 Choices & Conflicts in ΠM

We can extend the conflicts of ΣM to ΠM and view them on the latter as will be defined in what follows. It will be shown that *conflicts* carried over to ΠM from ΣM are blended with another category of *choices* unique to ΠM in an indistinguishable manner from the latter. Therefore conflicts are *transparent* to ΠM.

2.6.2.1 Conflicts in ΠM

Definition 2.12 Two states s_m, s'_m are in conflict denoted (s_m $conf_g$ s'_m) iff:

(s_{mi} *conf* s'_{mj}) as defined in ΣM, where s_{mi}, s_{mj} are some components of s_m, s'_m respectively.

Since conflict represents the true choice exhibited by the specification, one would expect the conflict relation to correspond to the choice defined in the last section. But we see that it does not, as illustrated by the following example:

The two states (b_0 q_0 x_0) and (a_0 q_0 y_0) are *not in conflict* since none of their respective components are; but they are related by *choice* according to the definition of *choice* relation in ΠM as defined in Section 2.5.1.2. This means that there are choices in ΠM other than the ones due to conflicts among Mpm-states. We call these extraneous ones as *non-deterministic choices* just the way they are traditionally termed, and they originate from the simulation of *concurrency* among Mpm-states by *non-deterministically interleaving* them *sequentially* in all possible arbitrary orders.

Example 2.11 For example, from state (a_0,p_0,x_0) in Fig. 1 above, (b_0,q_0,x_0) can be reached through (a_0,q_0,x_0) or (b_0,p_0,x_0) by executing the asynchronous transitions (a_0, e_1, b_0) of M_1 and (p_0, e_2, q_0) of M_2 (e_1, e_2 are respective local events not labelled in the figure) one after the other *sequentially* in *either order*. The states (a_0,q_0,x_0) and (b_0,p_0,x_0) are in *(non-deterministic) choice*.

The above phenomenon of replacing the concurrent transitions of the Mpms is referred to as *nondeterministic interleaving* and the corresponding paths of ΠM as *non-deterministic choice paths*. The issue of interleavings will be formally handled in a future section.

2.6.2.2 Non-deterministic choices in ΠM

Definition 2.13 Two states s_m, s'_m are related by *non-deterministic choice* if they are neither in conflict nor in sequence:

(s_m $choice_{\text{non-det}}$ s'_m) iff:

$\wedge((s_m\ conf_g\ s'_m) \vee (s_m\ seq_g\ s'_m) \vee (s'_m\ seq_g\ s_m))$

Since the choice comes from *conflicts* or *non-deterministic choices*,

Definition 2.14 Two states s_m, s'_m of ΠM are related by *choice* denoted:

(s_m *choice* s'_m) iff:

$(s_m\ conf_g\ s'_m) \vee (s_m\ choice_{\text{non-det}}\ s'_m)$.

The union ($seq_g \cup$ *choice*) *is a total relation* among all the states of ΠM.

Both the relations seq_g and *choice* of ΠM are now *larger* than the ones that might have modeled the *true sequence* and *true choice* (conflict) relations respectively, as exhibited by the specification and by ΣM. In the process, *concurrency* is hidden as well from the product model, as reflected by the complementary nature of seq_g and *choice* relations to each other.

Thus among the relations *seq*, *conf* and *co* of ΣM, *co* disappears in ΠM. In its place, due to *nondeterministic interleaving*, every transition of ΣR_{mi} of ΣM appears *multiple* number of times (provably exponential as will be formalized in a later section) in ΠM one each from a set of as many global-states (as the above number), to make up R_m.

2.7 Extended Sum machine, $\Sigma^* M$

The theory presented thus far of the sum machine will be referred to as the *basic sum machine*, in which the local Mpm-states are the central entities. In what follows, we extend the presented notions of *causality* and the other relational structure among the local Mpm-states to those of *extended sum machine* denoted Σ^*M, in which the *state-vectors* or equivalently the *global-states* of Mpms will be the central entities of interest.

When there is no confusion, we skip the adjectives viz., *basic* and *extended* and their two distinct denotations of the sum machine, and let the context of reference identify one or the other.

Minimal prefixes and *Configurations* are the two important extended notions of the dependency-order $\leq$ again, forming vectors/sets of Mpm-states according to some criteria. These extended notions constitute the foundation of the (extended) sum-machine Σ^*M and its component Mpms that enable their applicability for an efficient verification of the properties of the concurrent/distributed system to be discussed in the sequel.

2.7.1 Minimal Prefix, Mp

An Mpm-state is a *unique instance* of an Fsm-state. This instance is not arbitrary, but carries a meaning. Abstractly, it inherits a *unique past*; as a culmination/extremity of which, there is a unique vector of Mpm-states associated with it, called its Minimal prefix.

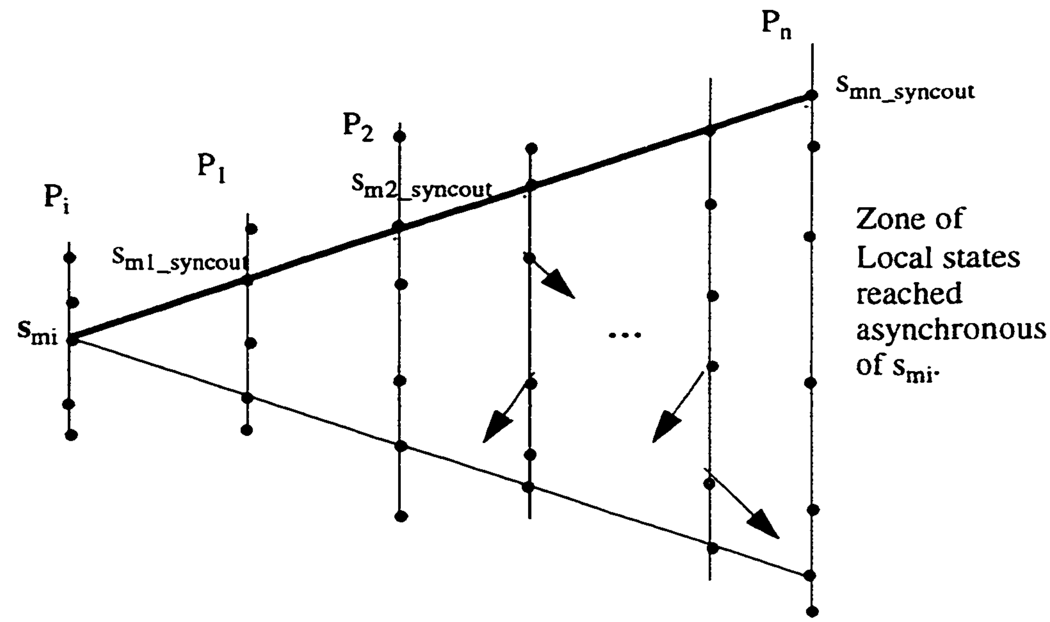
The conceptual definition of a Minimal prefix is made below. This definition precedes the more mathematical ones of Lemma 2.5 and Corollary 2.1 and Corollary 2.2, in order to emphasize the basic idea, *a priori*.

Definition 2.15 The Minimal prefix of an Mpm-state is an *n-state vector*, one from each Mpm M_i, $i = 1..n$ which should *necessarily* and *sufficiently* be *reached* in order to *guarantee* the *entry* of the given state, in accordance with the *causal dependency*.

$Mp_i(s_{mi}) := (s_{m1\_syncout}, s_{m2\_syncout}, \ldots, s_{mi}, \ldots, s_{mn\_syncout})$ as shown in Fig. 6 where the significance of the state labels $s_{m1\_syncout}$ etc. will be explained shortly. The *necessity* condition of the definition guarantees the *minimality* of all the states that must precede the entry of the given state s_{mi} and the *sufficiency* condition of the definition guarantees the *maximality* of the specific *vector components* reachable among all those states that must precede s_{mi}.

As will be formally proved, the non-local components of an Mp vector turn out to be synchronous output states. *A Minimal prefix thus forms a pairwise concurrent, reachable vector and at the same time formed out of causal dependency. Mp is a direct exploitation of the fact that concurrency and causality co-exist among states and are not complementary to each other.*

Fig. 6 Minimal-prefix , A Representative Vector of Global-states



The above figure represents a set of n paths ΣP_i,i=1..n of n Mpm-trees and a couple of state vectors having s_{mi} as their i^{th} component. The vector on top formed by ($s_{m1\_syncout}$, s_{m2_synout}, ...,s_{mi},..., $s_{mn\_syncout}$) is the Mp vector of s_{mi}. The zone of Mpm-states between the two vectors with s_{mi} component represent the ones reachable *asynchronous of* s_{mi}, from the individual non-local components of Mp in the respective Mpms. The *synchronous* components of Mp could have progressed up to the respective components of the bottom vector with s_{mi} within the zone (formed by the top and bottom vectors) still enabling the entry of s_{mi}. Mp can be considered as a *representative of all the asynchronous global states formed by the various possible combinations of these local states*. It is this *combinatorial* possibility that gives rise to exponential enumeration of global-states due to *non-deterministic interleaving* of the product machine.

If we can somehow avoid all the above asynchronous combinations of local-states to form asynchronous global-states and instead generate them with Mp vector alone as their *representative* on need-basis, as and when the occasion demands, then we would be avoiding their exponential enumeration. This will be the pursuit of the sequel.

Mp of a given state of an Mpm thus represents '*minimal globality*' not only by virtue of generating itself, but also of potentially generating all the global-states of the product machine. This will be made clearer in one of the following sections.

Example 2.12 The Mp of the state d_0 in Fig. B is given by:

$Mp(d_0) = (d_0, u_0, z_0)$, the 3-state vector. The reasoning behind the definition of Mp with respect to this example is as follows:

In order for d_0 to be entered, M_1 must trivially be at state d_0. Since d_0 synchronizes with u_0, u_0 must be *necessarily* entered to allow the entry of d_0. By transitivity in $\leq$, z_0 in M_3 must have also been entered. We say that M_3 has to be minimally at z_0 ; the term *minimal* is used because, M_3 could have exited z_0 and entered g_1 for instance, independent of M_1 and M_2, but it is *sufficient* that M_3 is at z_0 to *guarantee* the entry of u_0 and in turn d_0.

The condition of *sufficiency* is needed here to include only that *vector*, whose components are the maximal/largest among the respective Mpm's states (*local maximum*) ordered by R_{mi} are necessarily entered. In this example again, the states s_0 , y_0 of M_2 and M_3 respectively must be necessarily entered to guarantee the entry of d_0, but it is not sufficient until u_0 and z_0 are entered as well which are the respective local maximum (in the order $\leq$) of M_2, M_3 to guarantee d_0's entry.

2.7.2 Global-state Corollary

Corollary 2.1 Every Minimal prefix is a state of ΠM.

This is the corollary of the Global-state Theorem stated at Theorem 2.1. This follows since every Mp is a *reachable* vector of Mpm-states, every pair of which are *concurrent* to each other, i.e., related by the relation *co* since they can neither be in sequence (due to the *maximality* criterion of Mp) nor in conflict.

■

2.7.3 Minimal prefix and Synchronization

2.7.3.1 Mp Lemma

Lemma 2.2 All the non-local components in the Minimal Prefix of every state of an Mpm M_i are *synchronous output* states.

Proof: The proof follows from the fact that the non-local Mpms must progress *sufficiently* and *minimally* in order to enable the local Mpm to enter a given state which can only be due to *synchronization* requirement.

The *necessity, sufficiency* conditions of the definition of Mp of say, s_{mi} =>

(i) Non-local components *must* reach a specific, *maximal state* s_{mj}, possibly after a sequence of transitions in the order R^+_{mj} , j<>i. This state must be a *synchronous output* state which is directly a partner state of the given state s_{mi} or that of a synchronous output state s_{mk} of another Mpm M_k , k<>j, k<>i which should progress further on in order to be a partner state of s_{mi} .

(ii) The *sufficiency* condition of the definition of Mp of s_{mi} =>

Asynchronous progress of M_j, j<>i beyond the above *maximal synchronous output states* is unnecessary since that will not contribute to the progress of M_i to reach s_{mi} either through direct or indirect partnerships.

Hence the result.

■

It is thus noted that Mp covers both a *minimum* and a *maximum* criteria by its necessity and sufficiency conditions respectively.

Synchronization is the hurdle or a stumbling block that keeps an Mpm (the *primary* one), from progressing to all its future states independently of other Mpms (the *secondary* ones). In order to enable the given Mpm to cross the mentioned hurdle, the partner Mpm(s) participating in the synchronization have to possibly go through a sequence of asynchronous states and then synchronize with the former. This process may call for certain more synchronizations in a *recursive* manner. This will be reflected in the *generation algorithms* of CMpms (with respect to given CFsms). After the synchronization in each case, it is *immaterial* whether the respective partners continue asynchronously to progress or not, and if so how far, so long as the *necessary* hurdles for the Mpm in question before reaching a given state have been *sufficiently* crossed.

The above is stated as the *necessary* and *sufficiency* condition in the definition of an Mp.

2.7.3.2 Minimal Prefix as a *one-to-one* Function

Lemma 2.3 Minimal prefix can be expressed as a *one-to-one* function with its *domain* as the states of an Mpm and the *range* as the respective Minimal prefix vectors of those states.

In other words, Mp_i, i=1..n denote the n *one-to-one* functions, that express the Minimal prefixes of the states of the n Mpms respectively.

Proof: (by contradiction)

Essentially the proof follows from the absence of *backward-conflicts* in ΣM.

For a state to have more than one Minimal prefix, by its definition,

(i) the state is reached by two different *paths* of the given Mpm.

Or,

(ii) Either the state or one of its *ancestors* (with respect to $\leq$) is a synchronous output state with more than one set of partner states.

(i) => contradiction of the restriction of the state-graph of an Mpm to be a *tree*.

(ii) => contradiction of the property of *uniqueness of partnership* in synchronous output states as proved in Section 2.4.2.4.

Thus, Mp_i is a *function* within Mpm M_i, i=1..n.

$Mp_i : S_{mi} \dashrightarrow (S_{m1} \times S_{m2} \times \ldots S_{mi} \times \ldots \times S_{mn})$

$Mp_i(s_{mi}) = (s_{m1\_sout}, s_{m2\_sout}, \ldots, s_{mi}, \ldots s_{mn\_sout})$ where,

the i^{th} component of the Mp of the state s_{mi} is that state itself and the rest are synchronous output states, either partners of each other or having their own partners in the *past* of s_{mi} according to Mp Lemma. When s_{mi} itself is a *synchronous output state*, it shares an identical Mp-vector with all its partner states. That is why Mp_i is only a *local function*, within the domain of local Mpm-states of a given Mpm, M_i.

Mp_i is a *one-to-one* function; it follows trivially because, as *every state is unique* in an Mpm's state-tree, at least the i^{th} components of the two Mp-vectors corresponding to two distinct states have to be different, even if the other (n-1) components are possibly the same.

Hence the result.

■

Example 2.13 It is noted from Fig. B of *Appendix*,

$Mp_1(d_0) = Mp_2(u_0)$, since (d_0 *sync*$_{out}$ u_0), where u_0 is in M_2.

$Mp_1(v_1) = (v_1, t_0, z_0)$ where v_1 is the output state of an asynchronous transition. Except v_1, both the other components are synchronous output states in the past of v_1 in the dependency-order $\leq$.

2.7.4 Minimal Prefixes and Labelled Partial Order

The bijective association of Mpm-states and the Minimal Prefix vectors make the *partial order* $\leq$ relating the Mpm-states, a *labelled* one. This *labelled PO* of the sum machine is linked to the operational semantics of the product machine which helps to generate all the latter's global states. This will be demonstrated by some of the theorems of the sequel.

The Mp labels will be used dynamically during verification while *branching in space* from one Mpm-tree to the other to do the local search of the needed Mpm-trees, without losing track of the continuity in time. The Mp label of an Mpm-state is the '*minimal encoding*' of the non-local states that are causally dependent on the given Mpm-state. *Local conflicts* together with these *labels* keep track of the *global conflicts*, as will be demonstrated. This property is exploited in model-checking to be discussed as well in Chapter-4.

2.7.5 Local Configuration, C(s_{mi})

The notions of *local configuration* and of *general configuration* that follows the former are important to establish the link between the (Mpm-) states of ΣM and the global states of ΠM. They are also necessary to correlate the conventional notion of *runs* and *interleavings* of concurrent systems and formally define them with respect to our context. This link is vital for the expressiveness of a specification and the strategy adopted in the implementation of its verification and the complexity incurred.

Definition 2.16 The *Local configuration* of an Mpm-state s_{mi} is defined as the *upward closure* of that state, which is:

The set of states : $\{s_{mk} \mid s_{mk} \leq s_{mi}, \forall k=1..n\}$.

The phrase *upward closure* is the logical synonym for *Local Configuration* and is chosen with respect to the state-trees of Mpms which grow downward from their respective initial-states situated up in their roots.

The upward closure and so a local configuration is the set of all states that are to be necessarily reached in order to enable the entry of the state in question. This view establishes the link between the *Minimal prefix* of a state and its *Local Configuration*. Each of the *n* components of the former corresponds to the local maximum of the states ordered by R_{mi},i=1..n among the member states of the latter. This notion will be formally presented at a later section.

Property 2.4 No two states in the *upward closure* (local configuration) of a state can be in conflict.

Proof: (by contradiction)

Assume (s_{mi} *conf* s_{mj}) where,

s_{mi}, s_{mj} are members of the *upward closure* of s_{mk}.

=> (s_{mi} *conf* s_{mj}) $\wedge$ ($s_{mj} \leq s_{mk}$) from the above assumption and definition of upward closure.

=> (s_{mi} *conf* s_{mk}) by the *conflict-inheritance* property of Section 2.4.2.

=> contradiction of ($s_{mi} \leq s_{mk}$), from Property 2.3.

Hence the result.

▩

Local configuration represents the *unique global past-and-present* associated with a given Mpm-state in entirety , tracing back upward to the initial states of the system.

For the same reason as argued for Mp, local configuration is also *unique* for every state within a given Mpm. However, two states of two different Mpms may inherit the same *past and present* due to synchronization. In particular, two or more *synchronous output states* that are *partner states* share the same local configuration and Mp-vectors.

Consequently, we can formally define C_i, the local-configurations of the states of individual Mpms M_i, i = 1..n, as the following set of *one-to-one* functions:

C_i : $Smi \rightarrow Smset$ where *Smset* is the *powerset* of S_m, i =1..n.

$C_i(s_{mi}) = \{s_{mj} \mid s_{mj} \leq s_{mi}\}$, for j = 1..n, for every i=1..n.

Example 2.14 From Fig. B of *Appendix*, $C_1(d_0) = \{d_0, c_0, b_0, a_0, u_0, t_0, s_0, q_0, p_0, z_0, y_0, x_0\}$ and nothing else. $C_1(d_0)$ refers to the local-configuration of the state d_0 in Mpm M_1. No other local configuration of M_1 maps to the same set as above. But $C_1(d_0)$ is also equal to $C_2(u_0)$ since d_0 and u_0 are the *synchronous output* states reached simultaneously after a common synchronous event.

2.7.6 General Configuration C

Definition 2.17 A general configuration C, or a configuration in short, of the extended sum-machine Σ^*M is a subset of states of ΣM i.e., $C \subseteq \Sigma S_{mi}$, satisfying the following two constraints:

(i) *Upward closed* : If a state is present in a configuration, so does its local configuration.

(ii) *conflict-free*: No two states within the same configuration are in conflict with each other.

Every local configuration of s_{mi} by its definition satisfies both the above constraints as follows:

The upward closure i.e., local configuration of s_{mi} also contains the local configuration of every other member by its definition. No two states of an upward closure can be in conflict from Property 2.4. Thus every *local configuration* is a special case of a *general configuration*.

The *initial configuration* C_0 of Σ^*M consists of just the set of all initial Mpm-states, which also constitute the Mp of every one of those states, as there is no *past* beyond these :

$C_0 := \{s_{m01}, s_{m02}, \ldots, s_{m0n}\} := Mp_i(s_{m0i})$, i=1..n

Example 2.15

$C_0 := \{a_0, p_0, x_0\} := Mp\,(a_0) := Mp\,(p_0) := Mp(x_0)$ in Fig. B.

$C := \{d_0, c_0, b_0, a_0, v_1, u_0, t_0, s_0, q_0, p_0, g_1, z_0, y_0, x_0\}$ is a general configuration which is not a local one of any specific state.

By introducing the above notion of configurations in Mpms, we have the important advantage of expressing any configuration as a *set union* of local configurations, a result which is directly used in *deterministic model-checking* and proving its complexity claim as elaborated in Chapter-4.

2.7.6.1 Path

Definition 2.18 A path P_i is a sequence (*ordered set*) of k states, $k \geq 1$ from the state-tree of Mpm M_i, $i \leq n$ with an *initial state* and a *final state*, where:

Each state in the path is a *successor* state of the previous state in the Mpm-tree, except the initial state.

For example, $P_i := \{s_{m1i}, s_{m2i},...,s_{mki}\}$[1] is a path where:

s_{m1i} is the *initial state* of P_i,

s_{mki} is its *final-state* denoted as: $fs(P_i)$ and

k is the length of the path P_i which is the number of states in the path.

s_{m2i} is a successor state of s_{m1i} and so on.

k =1 => initial state is the same as final state of a path.

Example 2.16 The path $P_2 = \{p_0, q_0, s_0, t_0, r_0\}$ of M_2 in Fig. D has its initial-state as p_0 and final-state as r_0 with its length k = 5.

The paths are thus defined local to individual Mpms.

Definition 2.19 Two paths P_i, P_j are *conflict-free* iff:

No two states of P_i, P_j are in conflict.

i.e., $\forall\, s_{mi} \in P_i, \forall s_{mj} \in P_j,$ ^(s_{mi} *conf* s_{mj}).

Conflict-freeness can be extended to more than two paths as well.

A set of *n* paths are said to be conflict-free if every pair of those *n* paths is conflict-free.

2.7.6.2 Disjointness Theorem

Theorem 2.2 Every configuration is a disjoint union of exactly [2]*n unique, conflict free paths* P_i, i=1..n with *initial-states* s_{0mi} respectively from the *n* state-trees of M_i, i=1..n.

---

[1] Since a sequence is an ordered *set*, without loss of generality, a *path* is treated as a *set* itself for manipulative convenience in the sequel.

[2] *n* is italicized wherever possible to make it stand out in the running material which is the same as its non-italicized occurrences in other places.

i.e., $C = \Sigma_{i=1..n}P_i$ where,

P_i, P_j are *conflict-free* $\forall i, j = 1..n, i \lt\gt j$.

Proof: From the first constraint of the definition of the configuration, viz., the *upward closure*, every configuration contains the initial states of all the Mpms, s_{0mi}, i=1..n.

The second constraint of the definition of a configuration, viz., *conflict-freeness*, allows just one single path with s_{0mi} as its initial state from each M_i i=1..n. Since the state-tree of M_i is rooted at s_{0mi}, for two paths with s_{0mi} as their initial-state to be conflict-free, one must be a proper subset of the other. Otherwise they must be in conflict, by the definition of $conf_i$ relation in Definition 2.6, since the very successors of s_{0mi} are in conflict.

By the same constraint of a configuration, viz., conflict-freeness, no pair of the n respective paths, one from every state-tree of M_i, i=1..n can be in conflict with each other.

Only one *unique* path from each state-tree can account for the subset of states of C that belong to M_i. Any other path will alter the configuration to something other than C.

Therefore, any configuration is a disjoint union of n *unique* paths with initial states s_{0mi}, one each from every M_i, i=1..n that are conflict-free.

Hence the result.

■

Example 2.17

The configuration $C := \{d_0, c_0, b_0, a_0, v_1, u_0, t_0, s_0, q_0, p_0, g_1, z_0, y_0, x_0\}$ from Fig. B can be viewed as:

$C = \Sigma_{i=1..3}P_i$ where:

$P_1 := \{a_0, b_0, c_0, d_0\}$,

$P_2 := \{p_0, q_0, s_0, t_0, u_0, v_1\}$,

$P_3 := \{x_0, y_0, z_0, g_1\}$.

Local and general configurations with their corresponding Mp and Fsv respectively are illustrated in Figure 7 on page 52.

The above view of a configuration leads to the following notion.

2.7.6.3 Disjointness Theorem and Labelled PO

A *labelled* PO is an *abstract* entity and the set of n paths of n Mpm-trees is a *concrete* one, since it associates with the operational semantics of the n concurrent automata (CMpms). A configuration being a set of Mpm-states related by the causal dependency-order $\leq$ (which is *partial*), each labelled by or mapped to its *unique Minimal prefix vector* is a *labelled partial-order,* that is conflict-free. So, we see that disjointness theorem bridges the gap between the entities of abstract and concrete domains respectively, by extracting *conflict-free, labelled PO structures* that are *configurations* out of a *collection of concurrent automata* (CMpms) or the so-called *sum machine*.

The advantage of the fusion above is two-fold :

(i) Because of the connection to automata and their operational semantics, the problems of expressing many interesting liveness properties as well as safety ones and verifying them can be more easily solved in the concrete domain, as will be demonstrated in next two chapters.

(ii) It lays the foundation for resolving many open questions in the *classical formal language theory,* some of the details of which will be discussed at a later section and the rest are left for the future work.

2.7.6.4 Final State Vector of a Configuration

Every configuration has an associated *final-state-vector.*

Definition 2.20 If $C = \Sigma_{i=1..n} P_i$, its final-state-vector is defined as:

$Fsv(C) = (fs(P_1), fs(P_2),...fs(P_n))$ where $fs(P_i)$ is the final-state of

P_i, i=1..n.

The i^{th} component of Fsv(C) is denoted as.

: $Fsv_i(C)$ which is $fs(P_i)$, i=1..n, in the sequel.

Example 2.18 The Final state vector of the configuration, $C := \{d_0, c_0, b_0, a_0, v_1, u_0, t_0, s_0, q_0, p_0, g_1, z_0, y_0, x_0\}$ is given by:

$Fsv(C) := (d_0, v_1, g_1)$ which follows easily from the set of paths of C illustrated in the last subsection.

Lemma 2.4 The Minimal prefix of a state s_{mi}, given by $Mp_i(s_{mi})$ is the same as the *Final state vector* of the Local configuration of s_{mi} denoted $C_i(s_{mi})$.

Proof: $C_i(s_{mi}) = \{s_{mj} \mid s_{mj} \leq s_{mi}\}$, for j = 1..n , for every i=1..n.

Let $C_i(s_{mi}) = \Sigma_{\ i=1..n} P_i$ by applying the disjointness theorem, Theorem 2.2.

Let $Mp_i(s_{mi}) = (s_{m1}, s_{m2}, ..., s_{mn})$. Then,

s_{mi} is the *local maximum* (largest) in the order of R_{mi} among the states *necessarily entered* in order to enable the entry of s_{mi}, by the definition of Minimal prefix, Example 2.12.

=> $s_{mi} = fs(P_i)$, i=1..n.

=> $s_{mi} = Fsv_i(C_i(s_{mi}))$, i=1..n.

=> $Mp_i(s_{mi}) = Fsv(C_i(s_{mi}))$.

∎

2.7.6.5 Fsv Lemma

Lemma 2.5 There is a *one-to-one mapping* between configurations and their *final state vectors.*

Proof:

Since there can only be one *final-state* of a path, there can be only one final-state-vector of a configuration. Also, for two different configurations to be distinct, they must have their respective Fsvs differing in at least one component; for if they do not, they become one and the same, by the *upward closed* constraint of a configuration or the *disjointness theorem*.

Therefore Fsv is a *one-to-one* function from the set of all possible configurations *Cset*, to a set of *Mpm-state-vectors* of ΣM:

Fsv: $Cset \rightarrow (S_{m1} \times S_{m2} \times ... \times S_{mn})$

∎

Since the above is an one-to-one function, we can define the set of *general configurations* as its *inverse function* as follows:

C: $S_m \rightarrow Cset$ where $S_m \subseteq (S_{m1} \times S_{m2} \times ... \times S_{mn})$ is the set of all reachable Mpm-state vectors of ΣM. This function is both *one-to-one* and *onto* obviously but Fsv is not an *onto* function since only a subset of the vectors of the cross-product are reachable vectors.

Corollary 2.2 The Minimal-prefix of the state s_{mi}, $Mp_i(s_{mi})$ is the *Final state vector* of its local configuration, $C_i(s_{mi})$.

Property 2.5 $\wedge(fs(P_i)\ seq\ fs(P_j))$, $\forall i,j = 1..n$ where $\Sigma_{i=1..n} P_i$ is a configuration.

Proof: (By contradiction)

Let us assume $(fs(P_i)\ seq\ fs(P_j))$ for some i and j, i<>j. Then,

$\exists s_{mi} \in S_{mi}: fs(P_i)\ R_{mi}\ s_{mi}$ such that $s_{mi} \le fs(P_j)$ (By the definition of *seq* in Definition 2.5)

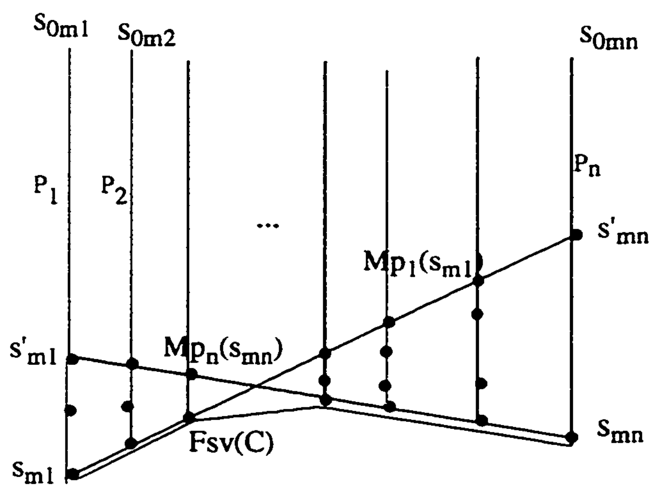
The above => $s_{mi} \in C$ since $fs(P_j) \in C$ (By the *upward closed* constraint of C in Definition 2.17);

$s_{mi} \in C$ is a contradiction since $fs(P_i)$ is the largest in the order of R_{mi} among all the members of C from the definition of C and P_i,i=1..n. Hence the result.

▩

Figure below illustrates a set of n conflict-free paths that grow into local and general configurations. The vectors denoted by s_{m1}-s'_{mn} = $Mp_1(s_{m1})$ and s'_{m1}-s_{mn}=$Mp_n(s_{mn})$ are the final state vectors of local configurations and their union forms a general configuration whose final state vector is s_{m1}-s_{mn}=Fsv(C), as shown in the figure.

Fig. 7 Local and General Configurations as sets of n conflict-free paths



2.7.6.6 Minimal prefix and Concurrency

Minimal prefix is a vector, each component of which should precede or synchronize with the associated (Mpm-)state, to enable that state to be reached. Therefore, the very entry of a state automatically implies the entry of other components of its Mp-vector in its *past*, or *possibly the present*.

Concurrency means two states *holding simultaneously* at some point of time, whether or not they are actually *entered simultaneously*. It was defined as the complement of the union of *sequence* and *conflict*. When there are no conflicts, (i.e., the *conf* relation is *Null*) *co* is the complement of *seq*. It was also mentioned that concurrency is originated by *simultaneity* of synchronous output states and multiplies due to *asynchrony*. So, given an (Mpm-)state s_{mi}, it may be interesting to consider, how many states can be reached *asynchronous/independent* of s_{mi}, i.e., *concurrent* to it, before becoming *sequential* to it.

The above gives an idea of the *degree of concurrency* i.e., to what extent the related *partner states*, originated as *strongly concurrent* ones in *simultaneity*, can progress asynchronous of each other with their local descendents (in the order R_{mi}) still remaining concurrent. In this sense, the *co* relation is given a different perspective with respect to Mp, orthogonal to its relationship with *conf*. We assert the following:

2.7.6.7 Concurrency Lemma

Lemma 2.6 $(s_{mi}\ co\ s_{mj}) \Rightarrow (s_{mj} \geq Mp_i(s_{mi})(j)) \wedge (s_{mi} \geq Mp_j(s_{mj})(i))$ where,

$Mp_i(s_{mi})(j)$ denotes the j^{th} component of $Mp_i(s_{mi})$.

Proof:

The very entry of s_{mi} implies the *past* or *present* entry of every component of $Mp_i(s_{mi})$, by the definition of a Minimal prefix. Therefore, in order to *hold simultaneously* with s_{mi}, s_{mj} must at least be equal to the corresponding j^{th} component of $Mp_i(s_{mi})$ i.e., $Mp_i(s_{mi})(j)$, or one of its *local descendents* in the order R^*_{mj} (and hence $\geq$; it is to be noted that $\geq$ degenerates to R^*_{mj} among states local to M_j). Likewise, the symmetric argument for s_{mj} holds good.

■

Both the conjuncts in the lemma above represent the sequence of (possibly none) asynchronous transitions of M_i and M_j respectively with respect to one another. This lemma defines the interval/span of states which can be entered asynchronously with respect to another state still remaining concurrent to it, (i.e., related by *co*) thus defining the *degree of asynchrony* and so *of concurrency*.

2.7.6.8 Fsv Theorem

Theorem 2.3 Every pair of states in a Final state vector (of a configuration) are related by *co* relation. i.e.,

If $C = \Sigma_{i=1..n} P_i$ then $fs(P_i)$ *co* $fs(P_j)$ $\forall$ i, j =1.. n, i<>j.

Conversely, if there is a vector of Mpm-states such that (s_{m1} *co* s_{m2} *co*...*co* s_{mn}), then the components form the final-states of the paths whose disjoint union is a configuration.

Proof:

The first part:

^($fs(P_i)$ *conf* $fs(P_j)$) : By conflict-freeness constraint of C from Definition 2.17.

^($fs(P_i)$ *seq* $fs(P_j)$) : By Property 2.5.

Therefore, ($fs(P_i)$ *co* $fs(P_j)$) for all i,j $\in$ n : By the definition of *co* in Definition 2.8.

of the three relations, *seq*, *conf* and *co*, as defined among all the states of ΣM.

The second part: (by contradiction)

Let us consider the *n* paths with the respective initial states as s_{0mi} and final state as s_{mi}, ΣP_i. We have to show that ΣP_i satisfies both constraints of a configuration:

(i) ΣP_i is *conflict-free* by the following contradiction:

If any two states belonging to two different paths P_i, P_j are in conflict, by the *conflict inheritance property* of Property 2.1, (s_{mi} *conf* s_{mj}) would be the deduction, which contradicts (s_{mi} *co* s_{mj}) since *co* and *conf* are complementary. Thus the n paths are conflict-free.

(ii) ΣP_i is *upward-closed* as follows:

(s_{m1} *co* s_{m2} *co*...*co* s_{mn}) => $s_{mi} > Mp_j(s_{mj})(i)$, $\forall$ j <>i by *concurrency Lemma* at Lemma 2.6

=> $s_{mi} = fs(P_i)$, i=1..n and $C_i(s_{mi}) \subseteq \Sigma P_i$

Therefore, C= ΣP_i forms a configuration.

Hence the theorem.

■

2.7.6.9 Continuations of Configurations in ΣM

Definition 2.21 If two configurations C and C' are related such that $C \subseteq C'$ then, C' is said to be a *continuation* of C and Fsv(C') is said to be a *descendent* of Fsv(C). C is said to be an *ancestor* of C' and also Fsv(C) an *ancestor* of Fsv(C'). We say that the continuation C' is *reachable* from C as well as the vector Fsv(C') from Fsv(C).

As a special case of the above, suppose a continuation C' can be reached from C by a single transition ($s_{mi}\ e_{mi}\ s'_{mi}$) of R_{tmi} such that $C' - C = \{ s'_{mi} \}$. Then, C' is said to be the *successor* of C.

2.7.6.10 Conflict between Configurations

Definition 2.22 Two configurations C and C' are said to be in conflict iff:

$\exists s_{mi} \in C$, $\exists s'_{mi} \in C'$ such that: (s_{mi} $conf_i$ s'_{mi}).

2.7.7 Configurability

The theorem to follow suggests a method of checking if the union of two given configurations give rise to a third configuration, containing the two. It does so essentially by checking if the individual components of the corresponding Fsvs are reachable from one another when not identical.

The theorem is applied to check if the union of the local configuration of an Mpm-state s_{mi} and a given, possibly general configuration C gives rise to a *continuation* of C. If it does, we say that $C_i(s_{mi})$ or s_{mi} *configures with* C. This process is referred to as the *configurability checking*. Illustration with an example will be given in the context of application of this theorem in Chapter-4.

2.7.7.1 Configurability Theorem

Theorem 2.4 The result of the set union of two configurations C, C' is a third configuration C" *if and only if* every corresponding components of their respective Final-state-vectors viz., Fsv(C) and Fsv(C') are *reachable* from one another (if not identical), in their respective Mpm-trees.

i.e., C" = (C U C') is a configuration <=>

(i) $Fsv_i(C)\ R^*_{mi}\ Fsv_i(C')$,

Or,

(ii) $Fsv_i(C')\ R^*_{mi}\ Fsv_i(C)$, for all i=1..n,

such that: in case (i), $Fsv_i(C'') = Fsv_i(C)$,

and, in case (ii), $Fsv_i(C'') = Fsv_i(C)$.

Proof:

=> part:

Let C, C', C" be three configurations. Then they can be expressed as follows:

$C = \Sigma P_i$, $C' = \Sigma P'_i$ and $C'' = \Sigma P''_i$, by applying the *disjointness theorem* in Theorem 2.2.

Then, $C'' = (C \cup C') \Rightarrow$

$\Sigma P''_i = \Sigma_i P_i \cup \Sigma_i P'_i = \Sigma(P_i \cup P'_i)$

The initial state of both P_i and P'_i is s_{0mi}, i=1..n, and they have to be conflict-free from the definition of a configuration.

$\Rightarrow (fs(P_i)\ R^*_{mi}\ fs(P'_i))$ or $(fs(P'_i)\ R^*_{mi}\ fs(P_i))$ from the following result:

$fs(P_i) = Fsv_i(C)$ and $fs(P'_i) = Fsv_i(C')$, by the definition of Fsv(C) as in Definition 2.20.

<= part:

It is now given that:

$(fs(P_i)\ R^*_{mi}\ fs(P'_i)) \vee (fs(P_i)\ R^*_{mi}\ fs(P'_i))$, i=1..n such that: $C = \Sigma P_i$ and $C' = \Sigma P'_i$ are configurations.

$\Rightarrow (P_i \cup P'_i)$ forms a single path P''_i with s_{0mi} as its initial state, i=1..n.

Since P_i, P_j are conflict-free as well as P'_i, P'_j for all i,j: 1..n, i< >j from the *disjointness theorem* (Theorem 2.2) ,

$\Sigma P''_i = \Sigma_i P_i \cup \Sigma_i P'_i = \Sigma(P_i \cup P'_i)$ where P''_i, P''_j are conflict-free for all i,j =1..n, i< >j.

To show that $\Sigma P''_i$ is *upward closed:*

$fs(P''_i) = fs(P_i)$ or $fs(P'_i)$, depending on P_i is contained in P'_i or vice versa.

$\Rightarrow C_i(fs(P''_i)) \subseteq \Sigma.P''_i$ from the fact , C and C' are configurations with $C_i(P_i) \subseteq \Sigma.P_i$ and $C_i(P'_i) \subseteq \Sigma.P'_i$.

$\therefore C'' = \Sigma P''_i$ is a configuration.

The *'such that'* part of the theorem follows from the fact that $fs(P''_i)$ is:

$= fs(P_i)$, if $fs(P_i)$ is reachable from $fs(P'_i)$,

$= fs(P'_i)$, otherwise.

■

Definition 2.23 $Fsv(C) \leq Fsv(C')$ iff: $Fsv_i(C)\ R^*_{mi}\ Fsv_i(C')$, i=1..n

This is the extension of the dependency-order $\leq$ to order the vectors as well.

2.7.7.2 Configurability Corollary

Corollary 2.3 $Fsv(C) \leq Fsv(C') \iff C \subseteq C'$

This is a special case of the theorem above when C is already contained in C'.

■

Fsv(C) is referred to as the *ancestor* of Fsv(C') , the latter being the *descendent*. When C' *succeeds* C, Fsv(C') is a *successor* of Fsv(C).

2.8 Equivalence Classes of Final-state-vectors of ΣM

2.8.1 Asynchrony with respect to an Mpm-state

Consider a local configuration $C_i(s_{mi})$ of state s_{mi} in the state-tree of Mpm M_i, and a general configuration C such that $s_{mi} = Fsv_i(C)$.

Then, $C_i(s_{mi}) \subseteq C$, from the definition of local and general configurations. We say that C and Fsv(C) are reached *asynchronous of* s_{mi}.

The above implies that:

$Mp_i(s_{mi}) \leq Fsv(C)$, for all C such that $s_{mi} = Fsv_i(C)$, from Corollary 2.3 (*configurability corollary)* and Lemma 2.4.

2.8.2 Equivalence Relation, RMp_i

We define a *binary relation* RMp_i as follows:

Definition 2.24 ($Fsv(C)\ RMp_i\ Fsv(C')$) iff:

$Fsv_i(C) = Fsv_i(C') = s_{mi}$, $s_{mi} \in S_{mi}$.

Following is true:

$Mp_i(s_{mi}) \leq Fsv(C)$, for all C such that $s_{mi} = Fsv_i(C)$.

RMp_i is easily verified to be an *equivalence relation* since it is *reflexive*, *transitive* and *symmetric*. This equivalence relation splits the set of Final state vectors into as many classes as there are states of S_{mi}. $Mp_i(s_{mi})$ being the smallest in the order $\leq$ among all the vectors with s_{mi} as a component, is the *representative* of the *equivalence class* formed by RMp_i, one for every $s_{mi} \in S_{mi}$ of M_i. This follows from the following reachability relation:

$Mp_i(s_{mi}) \leq Fsv(C)$, for all C such that: $s_{mi} = Fsv_i(C)$.

We thus get the *equivalence class* of $Mp_i(s_{mi})$ denoted by $[Mp_i(s_{mi})]_{RMp_i}$ which consists of all the state vectors with s_{mi} as their i^{th} component. It is to be noted that there are vectors Fsv(C) within an equivalence class that are in conflict with each other even though they are all reachable from $Mp_i(s_{mi})$.

Among all the Final-state vectors, there are Minimal prefixes (corresponding to local configurations) and those that are not Minimal prefixes, (corresponding to general configurations). The union, $\cup RMp_i$, for all $s_{mi} \in S_{mi}$ splits the set of all the vectors into as many equivalence classes as there are Minimal-prefix vectors of ΣM, given by the cardinality of the union of functions ΣMp_i which is utmost the cardinality of ΣS_{mi}. It is utmost because, simultaneous states have identical Mp vectors.

RMp_i, for every i =1..n, is defined *orthogonally*, where each equivalence relation splits the set of Final-state-vectors into disjoint subsets of $[Mp_i(s_{mi})]_{RMp_i}$, in *n* orthogonal ways/ dimensions, one for every i =1..n. When a given Mpm M_i is traversed as a *primary* one, we perceive the set of Final-state-vectors using the corresponding equivalence relation, RMp_i.

Example 2.19 From Fig. C of *Appendix*, consider $s_{m1} = b_0$ where,

$Mp_1(b_0) = (b_0, p_0, x_0)$;

Let $s_m = Fsv(C) = (b_0, p_0, y_0)$, $s'_m = Fsv(C') = (b_0, q_0, x_0)$.

Then, (b_0, p_0, x_0) RMp_1 (b_0, p_0, y_0), (b_0, p_0, y_0) RMp_1 (b_0, q_0, x_0) ,

(b_0, p_0, y_0) RMp_1 (b_0, q_0, x_0) and (b_0, q_0, x_0) RMp_1 (b_0, p_0, y_0) as well as

(b_0, p_0, x_0) RMp$_1$ (b_0, p_0, x_0).

Fsv(C) and Fsv(C') will be in conflict, when C and C' are in conflict i.e., when some of the respective members are in conflict. For instance,

(d_0, v_1, z_0) RMp$_1$ (d_0, v_0, g_0) even though (v_1 conf$_2$ v_0) and so (d_0, v_1, z_0) $conf_g$ (d_0, v_0, g_0)

Example 2.20

From Mpms M_1, M_2 and M_3 of Fig. C,

$[a_0,p_0,x_0]_{RMp_1} = \{(a_0,p_0,x_0), (a_0,q_0,x_0), (a_0,p_0,y_0), (a_0,p_0,x_4), (a_0,q_0,y_0),(a_0,q_0,x_4)\}$.

$[b_0,p_0,x_0]_{RMp_1} = \{(b_0,p_0,x_0), (b_0,q_0,x_0), (b_0,p_0,y_0), (b_0,p_0,x_4), (b_0,q_0,y_0),(b_0,q_0,x_4)\}$.

$[c_0,s_0,x_0]_{RMp_1} = \{(c_0,s_0,x_0), (c_0,t_0,z_0), (c_0,t_0,g_1), (c_0,r_0,h_0), (c_0,s_1,x_3)\}$.

$[d_0,u_0,z_0]_{RMp_1} = \{(d_0,u_0,z_0), (d_0,v_1,z_0), (d_0,v_0,g_0)\}$.

$[a_1,p_1,x_1]_{RMp_1} = \{(a_1,p_1,x_1)\}$.

$[a_2,p_2,x_2]_{RMp_1} = \{(a_2,p_2,x_2)\}$.

Similarly, equivalence classes of RMp$_2$, RMp$_3$ can be enumerated as well.

$[a_0,p_0,x_0]_{RMp_2} = \{(a_0,p_0,x_0), (b_0,p_0,x_0), (a_0,p_0,y_0), (a_0,p_0,x_4), (b_0,p_0,x_4),(a_0,p_0,x_4)\}$.

$[a_0,q_0,x_0]_{RMp_2} = \{(a_0,q_0,x_0), (a_0,q_0,y_0), (a_0,q_0,x_4), (b_0,q_0,x_0), (b_0,q_0,y_0), (b_0,q_0,x_4)\}$.

etc.

We apply this formalism to develop the equivalence of Final state vectors of ΣM and global-states of ΠM and the concept of *cut-off* at a later section in the sequel.

2.9 Final-state-vectors of ΣM and Global states of ΠM

By *Global-state corollary* of Corollary 2.1, we noted that every Mp-vector is a state of ΠM.

The set of Mp-vectors defined by the functions MP$_i$, i =1..n form only a subset of global states of ΠM since all the non-local components are restricted to be *synchronous output states* in any Minimal prefix, as cited by Lemma 2.2. Equivalently, they are the *Final state vectors* of *local configurations* from Lemma 2.4, which are only a subset of general configurations. This forms the background of the rest of the section.

2.9.1 Equivalence of ΠM and ΣM

2.9.1.1 Equivalence Lemma

Lemma 2.7 The set of global-states of ΠM coincide with the set of Final state vectors of ΣM. i.e., they are equal.

Proof: $s_m \in S_m$ of ΠM <=> (s_{mi} *co* s_{mj}), where:

$s_{mi} \in S_{mi}$ of M_i, $s_{mj} \in S_{mj}$ of M_j, $\forall$ i, j = 1..n, from *global-state theorem*, Theorem 2.1.

For every C in ΣM, ($Fsv_i(C)$ *co* $Fsv_j(C)$), where:

$Fsv_i(C) \in S_{mi}$ of M_i, $Fsv_j(C) \in S_{mj}$ of M_j, $\forall$ i, j = 1..n, from *Fsv theorem*, viz., Theorem 2.3.

=> $Fsv(C) \in S_m$ of ΠM, for every C in ΣM and,

$C(s_m) \in$ Cset, for every s_m in ΠM where Cset is the set of all reachable configurations of ΣM.

■

2.9.2 Minimal prefix and Monotonicity

The following lemma shows that every local transition of an Mpm (and of ΣM) has a corresponding global transition of ΠM. Thus it links the Mpm-states of the sum machine ΣM and the global states of the product machine ΠM, generated *virtually* by the former.

2.9.2.1 Monotonicity Lemma

Lemma 2.8

(i) $\forall\ s_{mi} \in S_{mi}$, $\exists$ C in ΣM such that $Mp_i(s_{mi}) = Fsv(C)$, i=1..n,

(ii) $\forall\ (s_{mi}, s'_{mi}) \in R_{mi}$, $\exists$ C, C' in ΣM and $\exists\ s_m, s'_m \in S_m$ in ΠM such that:

($Fsv(C) \leq Fsv(C')$) and ($s_m\ R_m\ s'_m$) where:

$Fsv(C) = s_m \in [Mp_i(s_{mi})]_{RMp_i}$

$Fsv(C') = s'_m \in [Mp_i(s'_{mi})]_{RMp_i}$

Proof:

(i) follows directly from the fact that for every Mpm-state s_{mi}, there is a Minimal prefix vector defined by the function Mp_i as $Mp_i(s_{mi})$ and the local configuration $C_i(s_{mi})$ such that the $Fsv(C_i(s_{mi})) = Mp_i(s_{mi})$ from Lemma 2.4.

(ii) $(s_{mi}, s'_{mi}) \in R_{mi} \Rightarrow (s_{mi} \leq s'_{mi})$, from the definition of $\leq$ at Definition 2.3.

$\Rightarrow (C_i(s_{mi}) \subseteq C_i(s'_{mi}))$, from the definition of a *local configuration* at Definition 2.16.

$\Rightarrow Mp_i(s_{mi}) \leq Mp_i(s'_{mi})$, from the *configurability corollary*, Corollary 2.3.

$\Rightarrow Fsv(C) \leq Fsv(C')$ where:

$s_m = Fsv(C) \in [Mp_i(s_{mi})]_{RMp_i}$ and $s'_m = Fsv(C') \in [Mp_i(s'_{mi})]_{RMp_i}$, since the same transition $(s_{mi}, R_{mi}, s'_{mi})$ can be made in general, from a set of global-states $s_m \in S_m$ (possibly a singleton) with s_{mi} as their component, which follows from the definition of the *product machine* ΠM, in Definition 2.9.

Hence the result.

■

This lemma is applied in proving the following theorem as a generator of all the global-states of ΠM from the local Mpm-states and Mp-vectors of ΣM. This in turn is applied in the verification algorithm of Chapter-4, since by sequential traversal of local Mpm-states of ΣM we essentially traverse the global states of ΠM.

2.9.2.2 Equivalence Theorem I

Theorem 2.5

(i) There are as many *global-states* of ΠM as there are *configurations* of ΣM.

(ii) The former are *generated* as the *Final state vectors* of the latter, such that:

For every *successor* C' of a configuration C in ΣM such that $Fsv(C) \leq Fsv(C')$, there is a corresponding transition $(s_m\ R_m\ s'_m)$ generated in ΠM where $s_m = Fsv(C)$ and $s'_m = Fsv(C')$.

Proof of (i):

For every C of ΣM, there is a unique Final state vector Fsv(C) , from *Fsv Lemma* at Lemma 2.5.

=> For every configuration C of ΣM, there is a corresponding global-state s_m of ΠM, from the *equivalence lemma* in Lemma 2.7.

=> There are as many global-states of ΠM as there are configurations in ΣM.

Proof of (ii):

Given C' is a successor of C in ΣM,

$Fsv(C) < Fsv(C') \Leftrightarrow Fsv_i(C)\ R_{mi}\ Fsv_i(C')$, i=1..n from Definition 2.23.

=> $(s_m\ R_m\ s'_m)$ in ΠM where $s_m = Fsv(C)$ and $s'_m=Fsv(C')$ such that:

$s_m \in [Mp_i(s_{mi})]_{RMp_i}$

$s'_m \in [Mp_i(s'_{mi})]_{RMp_i}$

from the definition of RMp_i and *Monotonicity Lemma* in Lemma 2.8.

Hence the result.

▩

The detailed discussion of the proof follows:

The initial configuration consisting of the set of all initial Mpm-states, $C_0 =\{s_{0m1}, s_{0m2},...,s_{0mn}\}$ corresponds to the initial global-state of ΠM, viz., $s_{0m}= (s_{0m1}, s_{0m02},...,s_{0mn})$ which is also $Fsv(C_0)$ as well as $Mp_i(s_{0mi})$, i=1..n.

i.e., $s_{0m} = Mp_i(s_{0mi}) = Fsv(C_0) =(s_{0m1}, s_{0m2},...,s_{0mn})$.

Corresponding to every transition r_{tmi} of R_{tmi}, a successor C of C_0 is reached in ΣM and a successor of s_{0m} is generated by a corresponding transition of R_{tm} in ΠM by applying the *Monotonicity Lemma*.

Also, the same transition r_{tmi} can be applied from at most all the states of $[Mp_i(s_{0mi})]RMp_i$, the equivalence class reached by the transitions of R_{tmj}, j <> i executed *asynchronously* with respect to s_{0mi}, from $Mp_i(s_{0mi})$.

This process can be inductively continued for every resulting configuration (and Final-state-vector) of ΣM and the global-state of ΠM from the previous step. In each case, when a successor C' of the configuration C is formed, the Fsv(C') differs from that of the predecessor C only in one component in the case of asynchronous transition and two or more components in the case of a synchronous transition. The corresponding R_{tm} transition in ΠM space generates the successor global-state s'_m of s_m from the previous step, where s'_m=Fsv(C') and s_m = Fsv(C).

2.9.2.3 The Non-equivalence of ΠM and ΣM

But there are two *important differences* in the process of generating configurations in ΣM domain and global-states in ΠM domain:

(i) In the former, only the local Mpm-states and their transitions of the n Mpms ΣR_{mi} are stored using which the configurations and their final state vectors are dynamically and monotonically generated. (Even though the Minimal prefixes of the states given by $Mp_i(s_{mi})$, $\forall s_{mi} \in S_{mi}$ i=1..n are stored as illustrated by the example in Fig. C and Fig. D, these vectors are only used as *links/handles* at the time of changing the primary Mpm from one to the other. This issue will be elaborated in later chapters.)

Consequently, the conflicts are distributed across the disjoint Mpms of ΣM as the disjoint union $\Sigma conf_i$ as opposed to the global, homogeneous conflicts $conf_g$ of ΠM. This amounts to generating the global runs originating from global conflicts among global states of the latter by using the local runs corresponding to local conflicts of the former or in other words, *non-enumeration of all the runs*. As the cardinality of $conf_g$ is much higher than that of $\Sigma conf_i$ due to the distribution of latter, we incur a lot of complexity savings. This issue again will be centrally handled by a different section.

Also since the global-states of ΠM are generated as *Final state vectors* of *configurations* of ΣM, and since a configuration is a set of local Mpm-states, the order in which the local transitions of ΣR_{mi} are executed to reach the destination configuration does not matter, and are not recorded/stored. This is referred to as *non-enumeration of interleavings*, dealt with in a separate section.

On the other hand in ΠM domain, the sequence of transitions made from one global-state to the other till the destination is reached, is recorded explicitly, thus enumerating all the (global) runs as well as interleavings.

We exploit and capitalize on above characteristics of ΣM in the verification of properties of ΠM so that the enumeration of all possible runs and interleavings of global sates are avoided. Instead, those of the specific runs and interleavings as guided by the property checked (i.e., the global-state whose reachability is to be verified) are *dynamically* chosen by adding the local Mpm-states which suffice to be stored *statically* as they are. Since a single configuration represents possibly multiple ordering of transitions depending on the degree of concurrency, we save on space and time.

2.9.2.4 Summation Lemma

Lemma 2.9 If Fsv(C) = $(s_{m1}, s_{m2}, ..., s_{mn})$ then,

C = $C_1(s_{m1})$ U $C_2(s_{m2})$ U...U $C_n(s_{mn})$, where C is a *configuration* and C_i the *local configuration* of s_{mi}, $i = 1..n$.

Proof:

C = $\Sigma_{i=1..n}P_i$ where P_i, i=1..n are conflict-free paths respectively of M_i, i=1..n according to disjointness theorem at Theorem 2.2.

$s_{mi} \in C \Rightarrow C_i(s_{mi}) \subseteq C$, for all i=1..n, by the definitions of local and general configurations, as stated in Definition 2.16 and Definition 2.17 respectively.

Therefore, $\mathbf{U}_{i=1..n}(C_i(s_{mi})) \subseteq C$

Now, C can not have *any more* states other than the above union of left hand side because every s_{mi} is the *final state* of the path P_i, and adding any more state to any of these paths will tend to make the new state *final* in the respective path, a *contradiction*.

Thus, $\mathbf{U}_{i=1..n}(C_i(s_{mi})) = C$.

■

Example 2.21 The configuration, C := $\{d_0, c_0, b_0, a_0, v_1, u_0, t_0, s_0, q_0, p_0, g_1, z_0, y_0, x_0\}$ with Fsv(C) := (d_0, v_1, g_1) is the same as:

$C_1(d_0)$ U $C_2(v_1)$ U $C_3(g_1)$ which can be easily checked.

It is to be noted that unlike the union of certain paths of n Mpm-trees that *disjointly* make up C, the component local configurations are not disjoint. They have non-null intersection by virtue of sharing at least the set of all *initial states* of the Mpms, according to upward closure criterion.

Significance of the Summation Lemma:

The lemma suggests a method of reaching any arbitrary Mpm-state vector (and so a global-state of ΠM) given all the local Mpm-states, by building the local configurations of each of the latter. Since any two local configurations that can make up a third general configuration are provably non-disjoint, (by the definition of a configuration) we only have to add that subset of states of first component configuration, not present in second to get the union of the two and so on.

This lemma along with *disjointness theorem* supports representation of ΣM without explicitly representing the synchronization points (*points of simultaneity*) and so the causality order $\leq$. By storing the entire Mp-vector along with every Mpm-state, all local configurations can be determined by the *disjoint union of paths*, one from every Mpm with the corresponding component of the Mp-vector as its final state. The general configurations can be derived from the *non-disjoint union* of these paths, applying this lemma and the configurability theorem.

The *Equivalence theorem* I showed that every global-state of the product-machine ΠM corresponds to a configuration of the sum-machine ΣM. The summation lemma shows that every general configuration can be reached from the local configurations. Putting the two together, we deduce that *every global-state of ΠM is reachable from the local configurations and hence Minimal prefixes of ΣM.*

These results are used to show that a partially-ordered model-checker (to be introduced in Chapter-4) with ΣM is free of the complexity due to the *totally-ordered* product version of composition ΠM which is posed as the theorem below.

2.9.3 ΠM Generator Theorem

Theorem 2.6 The set of all *Minimal prefixes* (Mps) of all the Mpm-states (local) of ΣM form only a *subset* of the set of all *global states* of ΠM and are *necessary* and *sufficient* to generate the rest of the global-states of ΠM, given ΣM.

Proof:

Minimal prefixes are the Final state vectors of local configurations which form only a *subset* of all general configurations. Therefore, the set of all Mp vectors is only a subset of the set of all Fsvs and hence of all global-states of ΠM, by the *Equivalence theorem I.*

Every local configuration is associated with an Mpm-state and no Mpm-state can be generated without minimally generating its associated Mp-vector, by the definition of a Minimal prefix. Thus the set of Minimal prefixes are ***the minimal ones, necessary*** to generate the set of all Fsvs, and hence the global-states of ΠM.

By *Summation Lemma*, every general configuration C is generated as the *set union of local configurations* $C_i(s_{mi})$, i =1..n of Mpm-states. This is equivalent to generating Fsv(C) and so a global-state of ΠM, from the Minimal prefixes $Mp_i(s_{mi})$ of its component Mpm-states. Thus the set of Mp-vectors is *sufficient* to generate the set of all global-states of ΠM.

The following sequence of steps is presented to make the above proof rigorous:

Every $s_m \in S_m$ of ΠM is Fsv(C) for some C of Cset of ΣM, from *Equivalence Theorem I*, stated as Theorem 2.5.

=> $s_m = Fsv(\cup_{i=1..n} C_i(s_{mi}))$ from the *Summation Lemma* as in Lemma 2.9.

such that:

$C_i(s_{mi}) = \Sigma_{k=1..n} P_k$, where $fs(P_k) = Mp_i(s_{mi})(k)$, k=1..n, P_k is a path of M_k,

according to *disjointness theorem* stated as Theorem 2.2, and from Lemma 2.4 relating a Minimal prefix as the Fsv of a local configuration.

■

The interesting aspect here is that:

Just as an Mp-vector of a state consists of the components that are *necessarily* and *sufficiently* be entered before reaching that state, the *Mp-vectors are the necessary and sufficient global-states* generated in order to generate the rest of them. They form the minimal or the smallest vectors in the order $\leq$ of $\Sigma^* M$ and R_m of ΠM, as *representatives* of the equivalence classes, $[Mp_i(s_{mi})]_{RMp_i}$, $\forall s_{mi} \in S_{mi}$, $\forall$ i =1..n. Using these minimal representatives, the larger vectors are generated using the union of the associated local configurations.

2.9.3.1 Causality Lemma

Lemma 2.10 $C_i(s_{mi}) = \Sigma_{i=1..n} P_i$ =>

$(s_{mi} = fs(P_i)) \wedge fs(P_j) \leq fs(P_i)$, $\forall$ j =1..n , j <> i.

Proof:

From the definition of $C_i(s_{mi})$ which is the local configuration of s_{mi},

$s_{mj} \leq s_{mi}$, for all $s_{mj} \in C_i(s_{mi})$.

Since $fs(P_i) \in C_i(s_{mi})$, $\forall i=1..n$, it is also true that:

$fs(P_j) \leq s_{mi}$, $j = 1..n$.

When j=i in the above, $fs(P_i) < s_{mi}$ is a contradiction since $fs(P_i)$ must be the *largest* (in the order R_{mi} and so in the order $\leq$ as well) of all states of S_{mi} present in $C_i(s_{mi})$, from the definition of *final state* of a *path*.

$\therefore fs(P_i) = s_{mi}$.

Hence the result follows.

■

2.9.3.2 Causality Theorem

__Theorem 2.7__ *Concurrency* relation *co* and *causality* relation $\leq$ are not disjoint.

i.e., $(co \cap \leq) <> Null$

__Proof:__

The proof follows by considering the final-state vectors of local configurations i.e., the Mp vectors.

Let $C(s_{mi}) = \Sigma_{i=1..n} P_i$ be the local-configuration of s_{mi}.

$fs(P_i)$ *co* $fs(P_j)$ $\forall$ j =1..n: j <>i , by *Fsv theorem*, stated as Theorem 2.3.

$fs(P_j) \leq fs(P_i)$, $\forall$ j =1..n : j <> i , by *causality lemma* above.

$\therefore$ The theorem follows.

■

This result can also be proved alternatively using the definition of $sync_{out}$ relation which is a subset of both *co* and $\leq$, more conceptually.

Considering only Mpm-states within a configuration, the above theorem states that concurrency is *not necessarily* due to *incomparable* states in the *partial* dependency-order $\leq$ even though states unrelated by $\leq$ are automatically concurrent. Mpm-states can maintain their causality which can provide very useful information, and still remain concurrent. This is essentially because, simultaneity which is *strong causality/dependency* is the basis of concurrency.

The significances of this theorem and *causality lemma* above are the following:

(i) From the notion of Minimal prefix, it is mathematically provable that by maintaining the dependency-order among the Mpm-states of a given vector, the logical flow or the continuity of time as we *branch in space* from one primary Mpm to the other is maintained, a result that is paramount for the distributed verification of properties with Mpms.

(ii) *A state holding before or after or together with another state is orthogonal to the fact that they are concurrent or not.* This is applied in the checking of the reachability of a global-state in all the *non-deterministic interleavings* without traversing all of them but just an arbitrary one. This issue will be expanded in the following section to some extent, and in detail in Chapter-4 on verification algorithms.

Example 2.22 From Fig. C of *Appendix*,

$Mp_1(d_0) = (d_0\ u_0\ z_0)$ where $d_0 = u_0$ and $z_0 < d_0$.

Therefore, when the primary Mpm is switched from M_1 to M_2 or M_3, the definition of Minimal prefix guarantees that by continuing from state u_0 in M_2 (or z_0 in M_3) we are essentially continuing from the respective *present* and *past* of d_0 that are necessary for the entry of d_0 and so there is no loss of any information i.e., global-states, by this localized search and the continuity of time in this sense is assured.

Because $d_0 = u_0$, i.e., (d_0 $sync_{out}$ u_0), we are guaranteed that if d_0 and u_0 hold conjunctively in one interleaving, they must hold in all interleavings as well. Similarly, d_0 can not be entered before z_0 does, however the interleaved execution of M_1, M_2 and M_3 take place.

2.10 Runs and Interleavings

2.10.1 Conflict-free Sum-machine and Product machine

It is possible to start from the initial configuration C_0 of ΣM and build a single arbitrary ccontinuation of it by choosing only one successor of the previous configuration at every sstep, as long as is possible by simulating the transitions of ΣR_{mi}. We can thus form a *maximal configuration* C_{max}, which is *infinite*, in the case of a non-terminating system.

C_{max} obviously has no two states in conflict and consists of just one path from every Mpm of ΣM. The states of this configuration form a subset of ΣS_{mi}. All the transitions generating C_{max} could be isolated out and composed to form a *conflict-free sum-machine*, denoted $\Sigma r \subseteq \Sigma M$, since it satisfies the definition of a sum-machine except that $\Sigma conf_{ri}$ = Null , $\Sigma conf_{ri}$ denoting the conflicts among the states of C_{max}.

2.10.2 Definition of a Run

Definition 2.25 The *product machine* composed from the Mpms corresponding to a conflict-free sum-machine Σr is a *run*, denoted as Πr.

The Mpms corresponding to Σr consist of only one path each in their respective state-trees in order to form a conflict-free sum-machine. Since Πr is built from a *conflict-free subset* of ΣM, it must only follow that it is a *conflict-free subset of* ΠM, with $conf_{rg}$ = Null.

This goes to say that a run in ΠM manifests as a *maximal configuration* in ΣM domain. There will be as many runs as there are conflict-free subsets of ΣS_{mi} that form maximal configurations each. An example of a *run* Πr and its corresponding *sum-machine* Σr, a subset of ΣM are shown in Fig. 8 and Fig. 9 respectively.

Formally, $\Sigma r \subseteq \Sigma M =$

$(\Sigma S_{ri} \subseteq \Sigma S_{mi},\ \Sigma E_{ri} \subseteq \Sigma E_i,\ \leq_r \subseteq \leq,\ \Sigma s_{0ri} = \Sigma s_{0mi})$ where:

Σr is a conflict-free subset of ΣM with $\Sigma_{i=1..n} S_{ri} = C_{rmax}$, $Fsv(C_{rmax}) = s_{rmax}$.

Then Πr is defined from Σr just as ΠM from ΣM:

$\Pi r = (S_r, E_r, R_r, s_{0r} = s_{0m})$ where:

$\exists s_{rmax} \in S_r : Fsv(C_{rmax}) = s_{rmax}$ and $\forall s_r \in S_r$: $(s_r R_r^* s_{rmax})$,

$\forall s_r, s'_r \in S_r$, $\neg(s_r\, conf_g\, s'_r)$.

The *notation* S_{ri} is chosen to denote the local Mpm-states of $r_i \subseteq M_i$, i=1..n that are present in Σr. Using ΣR_{ri} the disjoint union of the local transition relations of r_i,i =1..n we generate R_r from ΣR_{ri} just like R_m from ΣR_{mi}.

All the ancestors Fsv(C) of $Fsv(C_{rmax})$ correspond to the global states of *run* Πr.

2.10.3 Non-enumeration of Runs and ΣM

Since a run is a product machine, multiple runs occur due to the conflicts among the *global-states* of ΠM, as decided by the cardinality of the $conf_g$ relation. As discussed in an earlier section, the cardinality of *conf* relation of ΣM among the *local* Mpm-states is much smaller than that of $conf_g$.

Even among the elements of *conf*, only the local conflicts of $\Sigma conf_i$ are actually represented as the local branches of individual Mpm-trees, and the rest are the inherited ones of the former without being explicitly represented.

Just like the global-states related by $conf_g$ (derived from *conf*) divide ΠM into multiple runs, Mpm-states related by $\Sigma conf_i$ divide ΣM into multiple *local configurations* that are explicit and stored statically. The rest of the *general configurations* are derived as the various combinations of the set union of the former from *Summation Lemma* at Lemma 2.9. This characteristic of ΣM is referred to as the *non-enumeration of configurations*. Since every run in ΠM space has a corresponding conflict-free sum-machine in ΣM domain, the above characteristic corresponds to *non-enumeration of runs* in ΠM domain. More concretely, the runs resulting *primarily* from the conflicts $conf_i$ of a given *primary Mpm* are referred to as *local runs* or equivalently, *primary runs* of M_i, i=1..n.

Definition 2.26 A *local run* or a primary run of ΠM is the one corresponding to a *conflict-free* subset $C_{irmax} \subseteq \Sigma M$ which forms a *maximal local configuration*.

Example 2.23 The sum-machine shown in Fig. 9 is a *primary run* of M_2 as well as of M_3 such that $Fsv(C_{2rmax}) = (c_0\ s_1\ x_3)$. The execution of M_1 is partial since there is no progress after the state c_0. We avoid the exponential enumeration of all general configurations by storing only the primary /local runs corresponding to all the local maximal configurations alone statically. Using these, we configure the general ones as demanded by the need, viz., the property to be verified by applying the *Summation Lemma*.

When a given Mpm M_i is generated/traversed as the *primary* Mpm, a subset of the local continuations of each of the *secondary* ones M_j, j <> i, as represented by $conf_j$, are correspondingly generated/traversed to configure with those of the primary Mpm M_i. Viewed mathematically, by traversing only the isolated elements of $conf_i$, we are at once able to keep track of those of $conf_j$, j <> i as well, without having to traverse the latter separately and exhaustively. But instead, the latter are accounted for, by reaching the non-local components of Mp vectors of Mp_i function, and forming the *union* of local configurations of C_i and a subset of C_j functions, by applying the *configurability theorem*. This argument is symmetrically applicable for every i =1..n, with j <> i .

The above result can be stated as the following property:

Property 2.6 All the (general) runs are possible to be generated, by traversing a subset of primary, local runs alone.

Proof:

A (general) run Πr is associated with a sum-machine whose member states form a configuration C_{rmax}. Similarly, a local run corresponds to a local configuration C_{irmax}. The result follows from the *Summation Lemma* of Lemma 2.9 stating that every general configuration can be generated as the *union* of n local configurations. Only those local configurations that are relevant to the property checked are considered in the union and hence a *subset* of them alone.

■

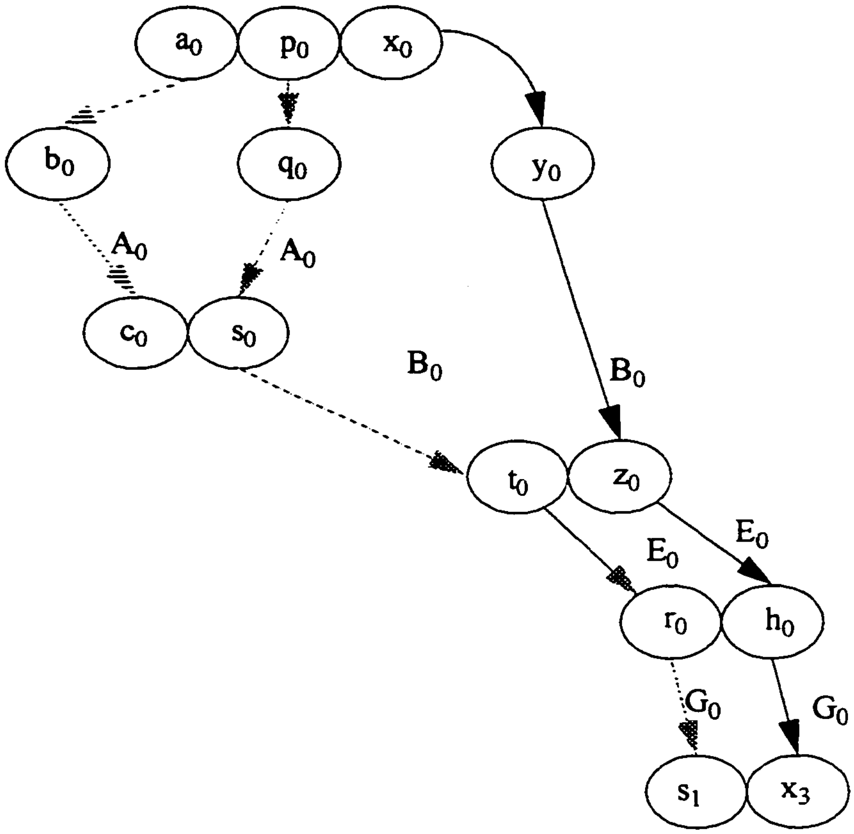
This result is an important consequence of the concept of *Minimal prefixes* and the fact that *global conflicts are manifestations of local ones*. It is implemented using the *configurability theorem* viewing configurations as *disjoint union of a set of paths* and will be elaborated in Chapter-4 on verification.

But it is to be noted that, depending on the system specification, it is possible as an extreme *degenerate case*, that the summation of local configurations do not result in any new configurations due to their possible overlap. In this case, the set of local configurations are already enumerated and coincide with the set of all general configurations. This issue will be explained further in a following section.

Example 2.24 Referring to Fig. C/ Fig.D of *Appendix* (Fig.C is reproduced as Fig. D with Minimal prefixes taken away from the state node entries and stored separately as a table), while traversing M_2 we consider only the local configurations formed by the conflicts of the relation, $conf_2$. Since (v_0 $conf_2$ r_0), where $Mp_2(v_0) = (d_0, v_0, g_0)$ and $Mp_2(r_0) = (c_0, r_0, h_0)$, when we traverse two local configurations of v_0 and r_0 respectively, we automatically will have reached the respective local configurations of g_0 and h_0 of M_3 and those of d_0 and c_0 of M_1. Thus we have taken into account (g_0 *conf*$_3$ h_0) of M_3 while traversing M_2. d_o and c_o form a single continuation of M_1 since (c_0 *seq* d_0).

Therefore, if we switch from M_2 to M_3 as the primary Mpm after reaching v_0, we only start traversing the local continuations of g_0 from M_3. Thus by traversing the local continuations of a single Mpm, we also traverse the non-local ones automatically.

Fig. 8 Σr, Conflict-free Sum machine



The above figure illustrates a conflict-free sum-machine $\Sigma r \subseteq \Sigma M$:

$\Sigma r = (\Sigma S_{ri} \subseteq \Sigma S_{mi}, \ \Sigma E_{ri} \subseteq \Sigma E_i, \leq_r \subseteq \leq, \ \Sigma s_{0ri} = \Sigma s_{0mi})$ where:

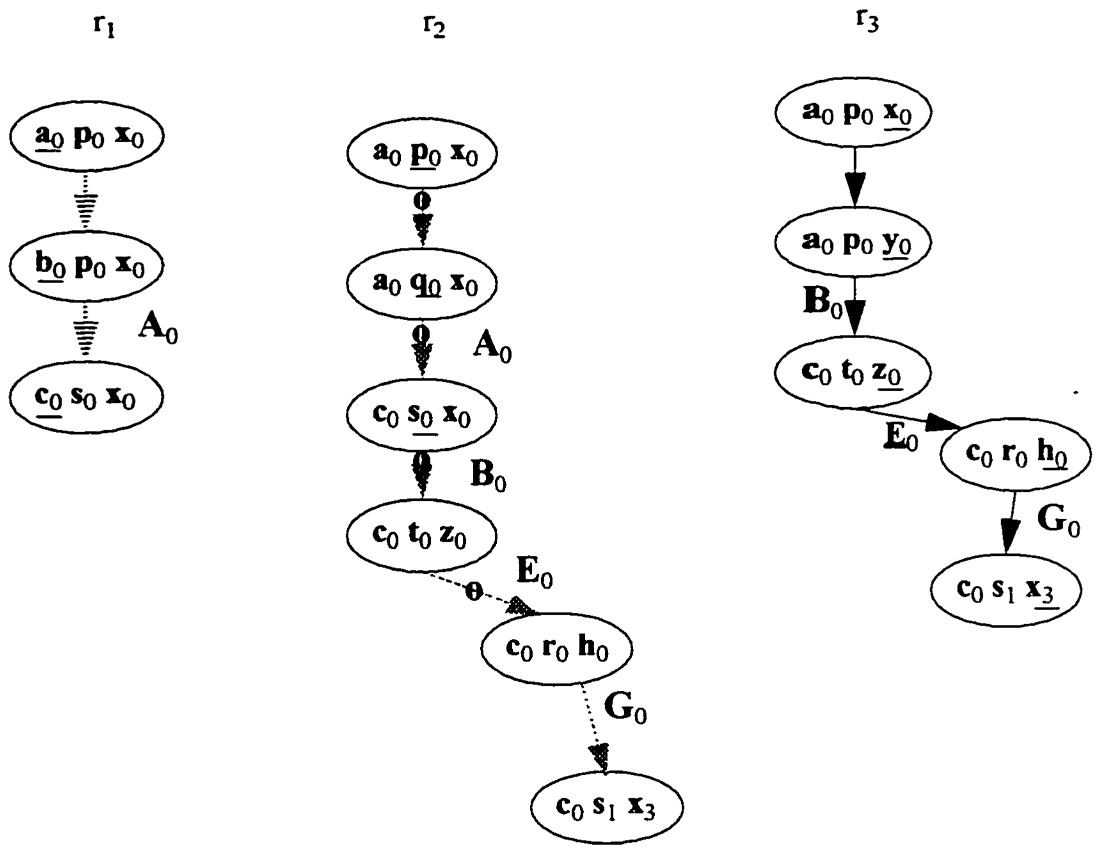
$\Sigma S_{ri} = \{a_0, b_0, c_0, p_0, q_0, s_0, t_0, r_0, s_1, x_0, y_0, z_0, h_0, x_3\}$

The set of synchronous events of $\Sigma E_{ri} = \{A_0, B_0, E_0, G_0\}$

$\Sigma s_{0ri} = \{a_0, p_0, x_0\}$.

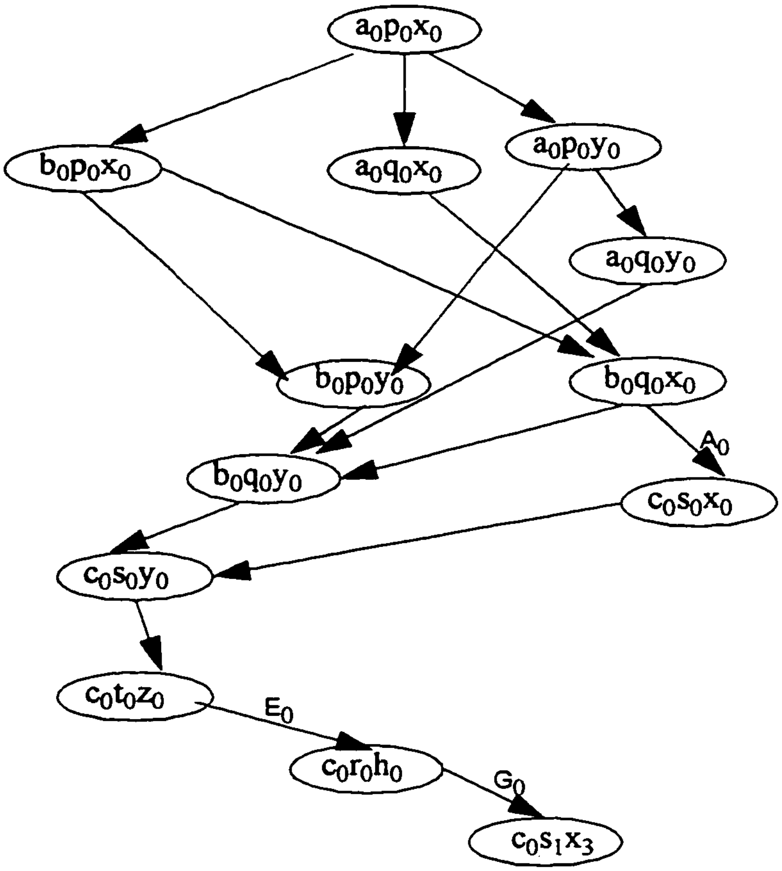
The component Mpms r_1, r_2, r_3 corresponding to Σr are shown in the following Figure:

Fig. 9 Mpms corresponding to a conflict-free sum-machine



The product machine constituting the run Πr, composed from r_1, r_2 and r_3 is shown in the following Fig. 10:

Fig. 10 Run Πr, (infinite) corresponding to a conflict-free sum-machine Σr



Since a run is *conflict-free*, the *choice* among states in the run as illustrated above must only have resulted from *non-deterministic choice* due to $choice_{\text{non-det}}$ relation since $conf_{rg}$ is *null* among the global-states of Πr.

Every path of Πr above from $s_{r0}= (a_0p_0x_0)$ to $s_{rmax}= (c_0s_1x_3)$ is the result of non-deterministic choice referred to as an *interleaved path* of the run Πr. Formal definition of an interleaving follows.

2.10.4 Interleavings of a Run

An *interleaving* is a more restricted subset of ΠM than a *run*, arising as a result of a specific order in which the events of different Mpms are executed in that run, forming a single *maximal path of a run*.

Definition 2.27 An *interleaving* is a *choice-free* subset of a *run* i.e., [1] $\Pi I_r \subseteq \Pi r$ defined as:

$\Pi I_r = (S_{Ir} \subseteq S_r, R_{Ir} \subseteq R_r, E_{Ir} = E_r, s_{0Ir} = s_{0r})$ such that:

R_{Ir}^{-} *totally orders* all the states of S_{Ir} with $s_{0Ir} = s_{0r}$ being the *lowest/least* and

$s_{Irmax} = s_{rmax} = Fsv(C_{rmax})$, being the *highest* in the order.

The *choice* relation among its states S_{Ir} is *Null*. Therefore, the interleaving reduces to a single path of Πr; the prefix Π from its notation may be skipped in the sequel and I_r denotes an *interleaving/interleaved path.*

An interleaving results from choosing just one successor non-deterministically from every global state until s_{rmax} is reached. Thus even the non-deterministic choice component (subset of $choice_{non\text{-}det}$) from its parent run is *Null* in an interleaving.

The notation S_{Ir} is chosen to denote the set of global states of the interleaving ΠI_r of run Πr, and similarly other entities. Upper case I, is chosen for interleaving so as not to interfere with lower case i, the subscript for the component machines; similarly with the other components of ΠI_r.

2.10.4.1 Interleaving Insensitivity/Independence of ΣM

Property 2.7 The set of all local states ΣS_{Iri} and transitions ΣR_{Iri} used to generate any interleaving I_r of a run Πr is equal to that of every other interleaving of Πr, which is the same as the *sum machine* Σr corresponding to Πr itself. We refer to this property as *interleaving insensitivity* or equivalently, *interleaving independence* of the *sum machine.*

In other words, $S_{Iri} = S_{ri}$, $E_{Iri} = E_{ri}$, $R_{Iri} = R_{ri}$, $i = 1..n$, $\forall I_r \subseteq \Pi r$

Proof: Since $s_{Irmax} = s_{rmax} = Fsv(C_{rmax})$, it follows that:

$C_{rmax} = C_{irmax}$ and every transition of ΣR_{ri} will occur in ΣI_{ri} in order to reach $s_{rmaxi} = s_{Irmaxi} \in S_{Iri}$, $i=1..n$ such that $s_{Irmaxi} = Fsv(C_{irmax})$.

Therefore *all states* ΣS_{ri} and events ΣE_{ri} of ΣR_{ri} will be generated to form the states and events of ΣS_{Iri} and ΣE_{Iri} respectively. This explains the property,

---

[1] Since an interleaving ΠI_r just reduces to a single path, it is often denoted as I_r itself without the preceding symbol of the product m/c.

$S_{Iri} = S_{ri}$, $E_{Iri} = E_{ri}$, $R_{Iri} = R_{ri}$, i= 1..n.

Aliter:

Interleaving insensitivity can be alternately proved by the fact that *since a configuration is a set, it is independent of the order in which the member states are added to generate it and so its Final state vector.*

■

Since there are no cycles in a sum machine, and there is only one successor from every state s_{Ir} according to the total-order R_{Ir}, ΣR_{ri} will occur *exactly once* in I_r to generate R_{Ir}. The fact that the transitions of ΣR_{ri} are generated multiple number of times in Πr as opposed to just once in I_r, explains the result $S_{Ir} \subseteq S_r$.

Example 2.25 From the run illustrated in Fig. 5, the following path of global-states,

$S_{Ir} = \{(a_0, p_0, x_0), (b_0, p_0, x_0), (b_0, q_0, x_0), (b_0, q_0, y_0), (c_0, s_0, y_0), (c_0, t_0, z_0), (c_0, r_0, h_0), (c_0, s_1, x_3)\}$

is one of the interleaved paths of the *run* Πr corresponding to the interleaving, I_r. The configuration C_{Irmax} and the *sum machine* ΣI_r formed by the local Mpm-states constituting S_{Ir} above is easily verified to be the same as that of S_r . i.e.,

$S_{Ir1} = \{a_0, b_0, c_0\} = S_{r1}$,

$S_{Ir2} = \{p_0, q_0, s_0, t_0, r_0, s_1\} = S_{r2}$.

$S_{Ir3} = \{x_0, y_0, z_0, h_0, x_3\} = S_{r3}$ and

$C_{Irmax} = C_{rmax} = \Sigma S_{Iri}, {}_{i=1..3}$.

The above can be shown for every interleaving of the run Πr.

There are as many interleavings I_r of a run Πr as there are number of subsets of S_r satisfying the definition above. Intuitively, it is easy to visualize that, the more the degree of concurrency (the less the cardinality of $\leq$ relation) , the more the number of such subsets and so, the more the number of interleavings of Πr; the less the degree of concurrency, the less the number of such subsets and so the less the number of interleavings.

Conceptually, multiple interleavings result from projecting the events and states of all M_i, i=1..n from their respective local time scales onto a single *global, real-time* scale in *total-order* such that every projection also obeys the dependency-order $\leq$.

2.11 CMpms with respect to a given CFsms Specification

As mentioned in the introduction, we assume an input specification as a set of n communicating Fsms (CFsms) F_i, i=1..n from the state-oriented paradigm. Fig. A of *Appendix* illustrates a set of three CFsms F_i, i=1..3 that communicate through the synchronous events specified.

Traditionally, we compose the given set $F_i := (S_{fi}, E_{fi}, R_{tfi}, s_{0fi}), i=1..n$

into a *product machine* denoted as ΠF according to the following definition:

Definition 2.28 $\Pi F := (S_f, E_f, R_{tf}, s_{0f})$ where,

$S_f \subseteq (S_{f1} \times S_{f2} \times \ldots \times S_{fn})$, $E_f = U_{i=1..n}E_{fi}$,

$s_{0f} = (s_{0f1}, s_{0f2}, \ldots, s_{0fn})$ is the *initial state*.

When all the elements of R_{tf}, the *transition relation* are generated, we automatically generate all the states $s_f \in S_f$ and events $e_f \in E_f$ as:

$(s_f, e_f, s'_f) \in R_{tf}$ iff:

$\exists i \in (1..n)$: $(s_{fi}, e_f, s'_{fi}) \in R_{tfi}$ and,

$\forall j <> i : (s_{fj}, e_f, s'_{fj})$, if $s'_{fi} = s'_{fj}$

$s_{fj} = s'_{fj}$,otherwise.

The binary *reachability relation* R_f is nothing other than R_{tf} with the event omitted from every element.

There seems to be a strong correlation between the entities of a set of *n* communicating Mpms and a corresponding set of communicating Fsms. The correlation is so strong that a *mathematical mapping* of the former's product composition *onto* the latter's is possible. Equivalently, given a set of communicating Fsms, it is possible to arrive at a (possibly more than one) set of communicating Mpms *functionally* using the above mentioned mapping. Formal definition of *finite model* of CMpms (which is a *deterministic* model) and the proof of its equivalence with the *non-deterministic model* of CFsms will be presented at a future section.

2.11.1 ΣM Generator Theorem

Theorem 2.8 Given a set of *n* CFsms F_i, i=1..n along with their synchronization requirements (consisting of synchronous events and each of their partner Fsm-identities), a set of *n* CMpms can be generated such that there exists a *function* B_i from every entity of M_i to the corresponding one of F_i, i =*1..n denoted* as:

$B_i : M_i \text{-->} F_i$, i=1..n

The above is a denotation of the following:

$B_{smi} : S_{mi} \text{-->} S_{fi}$, $B_{emi}: E_{mi} \text{-->} E_{fi}$ and extended to $B_{rtmi} \text{-->} B_{rtfi}$

Proof: (By construction)

The *recursive* steps involved in generating M_i, i =1..n constituting the sum machine ΣM from F_i, i=1..n will be shown below abstractly.

Given all transitions $r_{tfi} = (s_{fin}, e_{fi}, s_{fout})$ of the transition relation R_{tfi} of F_i,i =1..n respectively, i.e., $r_{tfi} \in R_{tfi}$, we generate all $r_{tmi} \in R_{tmi}$ of M_i, i =1..n as recursive functions and also the synchronization relations $sync_{in}$, $sync_{out}$. The dependency-order $\leq$ of Σ M is generated implicitly in the process.

Both $r_{mi}(p)$, $r_{mj}(q)$ are *recurrence functions* corresponding to the *levels* p and q of the *state-trees* of M_i, M_j respectively, claimed to be generated. We also use the *auxiliary function* f / f_{sync} chosen by the implementer of the generation algorithm, such that it gives an *occurrence identity* to every state and (event) such that each occurrence is *uniquely* generated at level p / (p,q,...) of M_i / (M_i, M_j,...) respectively in the case of local and synchronous states (events). This will be made clearer below:

Following *auxiliary functions* f_i , f_{syncij} are assumed :

$f_i : S_{mi} \text{ X } R_{fi} \text{-->} N$, $i=1..n$

$f_{0i}(\text{Null}, r_{0fi}) = 0;$

The subscripts 0 and i stand for the *level* p = 0 of the Mpm-tree of M_i respectively.

$f_{syncij} : (S_{mi} \text{ X } R_{fi} \text{ X } {}^1S_{mj} \text{ X } R_{fj}) \text{-->} N$, $i, j=1..n$, i <>j

We adopt the following *notation*, for the elements of r_{tfi}:

If the transition $r_{tfi} = (s_{fin}, e_{fi}, s_{fout})$ then,

$r_{tfi}.in := s_{fin}$, $r_{tfi}.e_{fi} := e_{fi}$ and $r_{tfi}.out := s_{fi}.out$.

Similar break-up for r_{tmi} is followed as well.

$r_{0fi} = (Null, init_f, s_{0fi})$ where,

$r_{0fi}.in$ = Null, /*the initial synchronous input state is considered Null */

$r_{0fi}.e_{fi} = init_f$, the special, *initial* synchronous event.

$r_{0fi}.out = s_{0fi}$;

At level p = 0,

$r_{0tmi} := r_{tmi}(0).in = Null$;

$r_{tmi}(0).e_{mi} = init_m$, the *initial* synchronous event.

$r_{tmi}(0).\ out = (r_{0tfi}.out, f_{0i}(Null, r_{0tfi})) = s_{0mi}$;

Therefore, $r_{tmi}(0) = (Null, init_m, s_{0mi})$;

$\forall\ r_{tmi}(p-1) \in R_{tmi}, r_{tfi} \in R_{tfi}$

$r_{tmi}(p).in = r_{tmi}(p-1).out$; /* output state of one level becomes the input state of next in the state-tree */

if $r_{tfi}.e_{mi}$ is *asynchronous*/local

$r_{tmi}(p).e_{mi} = (r_{tfi}.e_{fi}, f_i(r_{tmi}(p-1).out, r_{tfi}))$,

$r_{tmi}(p).out = (r_{tfi}.out, f_i(r_{tmi}(p-1).out, r_{tfi}))$.

else if $r_{tfi}.e_{fi}$ is *synchronous* with $r_{tfj}.e_{fj}$ (i.e., $r_{tfi}.e_{fi} = r_{tfj}.e_{fj}$)

$\forall\ q,\ r_{tmj}(q)$: $(r_{tmi}(p).in\ \mathit{sync}_{in}\ r_{tmj}(q).in)$

/*Level q refers to the level of the state-tree of M_j *synchronizing* with a state at level p of M_i. This involves generating beyond $r_{tmj}(q-k)$ in M_j for some k < q, corresponding to a sequence of transitions of M_j asynchronous of M_i, up to $r_{tmj}(q)$ where: $Mp_i(r_{mi}(q).in)(j) = r_{tmj}(q-k).in$ */

$r_{tmi}(p).out = (r_{tfi}.out,\ f_{syncij}(r_{tmi}(p-1).in, r_{tfi}, r_{tmj}(q-1).in), r_{tfj})$,

---

[1] In this abstract inductive procedure as well as the concrete algorithm, we assume without loss of generality that all the synchronizations are between two partners (except the special initial one, *init*) only, though more than two partners can be extended likewise. The auxiliary function f_{syncij} can be implemented in many ways. Instead of the range N, it could be just a concatenation of f_i and f_j values. (in general extending to more than two of partner functions).

$r_{tmi}(p).e_{mi} = (r_{tfi}.e_{mi},\ f_{syncij}(r_{tmi}(p-1).in,\ r_{tfi},\ r_{tmj}(q-1).in,\ r_{tfj}))$;

We add $(r_{tmi}(p).in,\ r_{tmj}(q).in)$ to $sync_{in}$, and

$(r_{tmi}(p).out,\ r_{tmj}(q).out)$ to $sync_{out}$.

Thus from R_{tfi}, i=1..n , R_{tmi}, $sync_{in}$, $sync_{out}$ and so $\leq := (\Sigma\ R_{mi}\ U\ sync_{out})^{+}$ (implicitly) of Σ M are generated.

In the above *recursive generation*, the following pattern of mapping is observed:

$r_{tmi}(p)$ --> $(r_{tfi}(p)$, occ#).....*eqn.* (m to f),

where occ# is a *natural number*, an *image* of f_i/f_{syncij} function.

Lemma 2.11 The above mapping of *eqn.* (m to f) is *one-to-one*.

Proof: (by induction)

Basis: The state at level p = 0 is the single *initial state* s_{0mi}, i=1..n and so is unique.

Inductive step: The input state s_{0mi} of level p =0 is mapped to a unique number (occ#) by f_i/f_{syncij}, i,j =1..n, i<>j which is tagged on to the output state of r_{tfi}.out to produce a unique output state for every r_{tfi} from s_{0fi} to generate corresponding r_{tmi}.out.

Inductive Hypothesis: Every input state at level (p-1) produces output states at level p, that are unique, i.e., mapped to unique ordered pairs of the above mapping.

∎

Essentially, the proof follows from the fact that every Fsm transition can be generated at most once from a given input state of the Mpm and so every output state is given a unique tag, as the occ# mapped by f_i/f_{syncij}, i,j =1..n uniquely takes both the Fsm-transition and the input Mpm-state into account.

Therefore, every r_{tmi} (i.e, the state and event) is generated as a *unique occurrence* of r_{tfi}. In other words, from a uniquely generated Mpm-state of the previous level, unique events and output states of the current level are generated (since their occurrence number depends on the already generated input state of the previous level) to form a *state-tree*. Since the *events* are *unique* by the same token as argued for output states, the *I O function* of the definition of M_i, i=1..n is satisfied as well.

When the occ# component is dropped off from the ordered pair of the right hand side, the above *one-to-one* mapping becomes *a non injective* [1]*(many-to-one) one*, labelled as B_i below:

[2]B_i: r_{tmi} --> r_{tfi}

Hence the ΣM generator theorem.

■

The concrete algorithm based on the abstract steps of the generation of state-trees of the Mpms M_i, i=1..n along with $sync_{in}$, $sync_{out}$ relations (which build the *dependency-order* $\leq$ implicitly) and so the *sum-machine* ΣM, is listed in *Chapter-4*. An account of this algorithm will be given after introducing a couple of more concepts incorporated in the algorithm.

Example 2.26 In the CFsm system shown in of Fig. A of *Appendix*,

r_{0tf1} = (*Null*, *init*$_f$, a);

f_{01}(Null, r_{0f1}) = 0,

$r_{0tm1} = r_{tm1}(0)$ = (*Null*, *init*$_f$, (a,0)) = (*Null*, *init*$_f$, a_0) where a_0 := (a, 0)

r_{0tf2} = (*Null*, *init*$_f$, p); $r_{tm2}(0)$ = (*Null*, *init*$_m$, p_0)

r_{0tf3} = (*Null*, *init*$_f$, x), $r_{tm2}(0)$ = (*Null*, *init*$_m$, x_0)

For r_{tf1}= (a, e_{f1}, b) (e_{f1} here is an asynchronous transition not labeled in Fig. A)

$r_{tm1}(1).in = r_{tm1}(0).out = a_0$;

$r_{tm1}(1).out = (b, f_1(a_0)) = (b, 0) = b_0 = r_{tm1}(2).in$.

For r_{f1} = (b, A, c), $r_{tm1}(1).out = b_0$ and, r_{tf2} = (q, A, s), $r_{tm2}(1).out = q_0$,

$r_{m1}(2).out = (c, f_{sync12}(r_{tm1}(1).in, r_{tf1}, q_0, r_{tf2})) = (c, 0) = c_0$, etc.

---

[1] In general, B_i is *not a surjective mapping* since there could be some Fsm-states which are never reached (due to communication deadlocks) and so there are no Mpm-states generated mapping to such Fsm-states. This is because, Mpm-states take into account the global-environment by their Mp-vectors. Generation of an Mpm-state means that there is at least one global-state reachable with that state as a component.

[2] The notation B_i : M_i --> F_i refers to the mapping of every element of M_i onto a corresponding element (state/event) of F_i.

In Chapter-4, where the actual generation algorithm is presented, the various stages of generation of this example will be better illustrated.

2.11.2 ΠF Generator Corollary

Corollary 2.4 There exists a *surjective map* from every entity of ΠM *onto* that of ΠF *denoted*:

B : ΠM --> ΠF where:

$\Pi F = (S_f, E_f, R_{tf}, s_{0f})$ (the product machine of F_i, i=1..n)

$\Pi M = (S_m, E_m, R_{tm}, s_{0m})$ (the product machine of M_i, i=1..n).

We can now define the mapping between the respective global-states and events of ΠM and ΠF as :

$\forall s_m \in S_m$: $B(s_m) = (B_1(s_{m1}), B_2(s_{m2}) \ldots, B_n(s_{mn})) = (s_{f1}, s_{f2}, \ldots, s_{fn}) = s_f \in S_f$, by the application of the definitions of B_i, i=1..n.

$\forall e_m \in E_m$, $B(e_m) = B_i(e_{mi}) = e_{fi}$ for at least one $i \in 1..n$

This is a *surjective (onto) map* since the set of Mpms simulate and map *all the reachable Fsm-state vectors* during their generation.

■

Example 2.27 The example used thus far in Fig. B, Fig.C and Fig. D is indeed generated by the above explained *functions* B_i from the set of CFsms F_i,i=1..3 along with the *synchronization specification* as shown in Fig. A of *Appendix*. The synchronization specification lists the *synchronous events* and the corresponding set of Fsms which synchronize during each such event.

The $sync_{out}$ relation among M_i,i=1..3 are shown explicitly by joining the related states together in Fig. B. These *synchronization points* are not explicitly shown in the version of Fig. C (Fig. D is same as Fig.C with Mp-labels stored in a separate table). As pointed out already, the formation of the sum machine is therefore not explicit in Fig. C unlike in Fig. B. To compensate for the explicit representation of synchronization points and so the dependency-order $\leq$ among M_i,i=1..n to build the *upward closure* (and so the local configuration of a state) we store the Minimal prefix $Mp_i(s_{mi})$,i=1..n along with every state

s_{mi}. We thus build $C_i(s_{mi})$ as $\Sigma P_i, i=1..n$ as claimed by the *disjointness theorem* using conflict-free paths P_i ending at components of $Mp_i(s_{mi})$.

The underlined state of every node in Fig. C represents its state and the rest of the three are the other two components of its Minimal prefix vector. The version Fig. C is more suitable for the view of configurations as a *disjoint union* of n paths of M_i, $i=1..n$ and to apply the configurability and the *generator theorems* (ΠM and ΣM) during verification, as will be detailed in Chapter-4.

The examples for $B_i, 1=1..3$ and B are given below:

$B_1(a_0) = a$, $B_2(v_1) = v$, $B_3(x_3) = B_3(x_2) = x$;

Similarly, $B_2(F_1) = B_2(F_0) = F$ etc.

$B(a_0, p_0, x_0) = (B_1(a_0), B_2(p_0), B_3(x_0)) = (a, p, x)$;

$B(d_0, v_1, z_0) = (d, v, z)$ and so forth.

2.12 Finite Model of CMpms

The CMpms as defined and considered so far are essentially *infinite* due to the possibly indefinite growth of their states and events. Under certain conditions as defined below, it is possible to define a finite model of a given set of infinite CMpms, although it is not guaranteed for every given set of CMpms.

Definition 2.29 A *finite model of* CMpms has a set of Mpm-states called *cut-off points/ states* that form the *leaf nodes* of all the Mpm-trees such that the following property is satisfied: A *cut-off state* is that Mpm-state forming the root of a sub-tree which is *isomorphic*[6] to the sub-tree rooted at an *ancestor* of that state, in the *infinite version* of the Mpm-tree to which it belongs.

Because of the isomorphism mentioned above, it is clear that in the finite model, the growth of each Mpm-tree beyond the cut-off states are unnecessary. Informally, the finite model of CMpms have a recurrent behaviour of its structure after the cut-off points and so, there is no need to grow it beyond these states. As mentioned before, isomorphism and hence finiteness is not guaranteed in any arbitrary set of infinite CMpms.

2.12.1 Finiteness of CMpms with respect to CFsms

We noted that each $M_i, i=1..n$ is in general an *infinite* system. When M_i is mapped to F_i, which is a *finite* machine, multiple states of M_i are mapped onto a single state of F_i. Similarly, multiple Mpm-state vectors (global state) of ΠM are *mapped onto* a single global state of ΠF, *surjectively.*

Cut-off states of ΠM with respect to ΠF:

A global state of ΠF decides its *future behaviour* i.e., the *descendent* states and events that are reachable from it. When two global states s_m, s'_m of ΠM with s'_m being a descendent of s_m are mapped onto the *same* global state of ΠF, s'_m is said to be a *cut-off state* of ΠM with respect to ΠF. Since the future of s_m is going to be repeated from s'_m as far as the behaviour of ΠF is concerned, we will not reach any of those global-states of ΠF by traversing the descendents of s'_m, that were not reachable from s_m.

Example 2.28

$B(a_1, p_1, x_1) = B(a_0, p_0, x_0) = (a, p, x)$

(a_1, p_1, x_1) is a descendent of (a,p,x) .

$B(c_0, s_1, x_3) = B(c_0, s_0, x_0) = (c, s, x)$ with the former global state being the descendent of the latter in ΠM.

2.12.1.1 Cut-off states, as viewed in ΣM

Definition 2.30 A *cut-off configuration* of ΠM with respect to ΠF as viewed from ΣM is defined as a configuration C such that for some configuration $C' \subset C$, $B(Fsv(C')) = B(Fsv(C))$. Fsv(C) is called the *cut-off vector.* Fsv(C') is called the *basis vector* corresponding to the cut-off vector, Fsv(C).

The above definition is general. In order to ease the detection of cut-off vectors during the generation of ΣM and during verification of ΠF using the latter as a platform, the definition has to be further refined with the following reasoning:

When we visit all the global-states of ΠM in the product-machine itself, it is very straightforward to identify the cut-off states by directly applying the above definition since all the global-states are reachable by paths of one single state-graph of ΠM.

When we generate ΣM and deduce the global-states of ΠM using the former's *n disjoint state-trees* we do not exhaustively visit all the state-vectors; and so all possible ancestors or *basis vectors* of ΠM corresponding to all the *cut-off vectors* are not visited. The same global-state which is reached for the first time by one interleaving could be a revisit (and so cut-off) for another interleaving of states.

So, we need to refine the notion of cut-off vectors in order that they are detected at ease using ΣM.

2.12.2 Minimal prefixes, Equivalence relations and Cut-off

The concept of Mp and local configurations further refine the notion of cut-off vectors by distributing and localizing them:

2.12.2.1 Cut-off Lemma

Lemma 2.12 When a configuration C is reached in ΣM, the set of Minimal prefix vectors traversed is the same irrespective of the order (interleaving) in which the members of C are added to build C.

Proof:

This is a direct result of *interleaving insensitivity* of configurations, stated as *Property 2.7.*

Every C has a unique set of Mpm-states.

Every Mpm-state has its unique Mp-vector, by Lemma 2.3.

Thus upon reaching C, the Mp-vectors traversed is the same irrespective of the order in which the member states are visited and added to form C.

■

When we choose to visit the Mp-vector of every member Mpm-state of the configuration C rather than every Final state vector of intermediate configuration generated (depending on the order in which the configurations are summed) during the traversal of C, the set of Minimal-prefixes (that are also global-states of ΠM) traversed to reach C is going to be the same, by *cut-off lemma* above.

So, it seems reasonable as well as convenient to use the Minimal prefixes alone as the *basis* and *cut-off vectors*. In the following, we formally show that the set of Mp-vectors as basis vectors and cut-off vectors guarantee the detection of any *general cut-off vector* following the *general basis-vector* as defined by the original definition.

2.12.2.2 Cut-off Theorem

Theorem 2.9 When $Mp_j(s'_{mj})$ is the *basis vector* corresponding to the *cut-off vector* $Mp_i(s_{mi})$, (possibly, i = j) then for every element of $[Mp_j(s'_{mj})]_{RMp_j}$ as a *basis vector*, there exists a corresponding *cut-off vector* in [1]$[Mp_i(s_{mi})]_{RMp_i}$. This is symmetrically applicable for all i =1..n.

Proof:

The proof essentially follows from the definition of the *equivalence relations* RMp_i, i=1..n.

It is given that, $Mp_j(s'_{mj})$ is the *basis* vector corresponding to the *cut-off* vector $Mp_i(s_{mi})$;

=> Each of the vectors s'_m reachable from $Mp_j(s'_{mj})$, will have some s_m reachable from $Mp_i(s_{mi})$ respectively such that: $B(s_m) = B(s'_m)$, by the definition of *cut-off* vector at Definition 2.30.

=> Each of the elements s'_m of the equivalence class $[Mp_j(s'_{mj})]_{RMp_j}$ (reachable from $Mp_j(s'_{mj})$ asynchronous of s'_{mj}) will have some element s_m of $[Mp_i(s_{mi})]_{RMp_i}$ such that: $B(s_m) = B(s'_m)$, by the definition of *equivalence class* formed by the *equivalence relation* RMp_i of an Mp-vector at Definition 2.24.

Therefore, with $Mp_j(s'_{mj})$ as *basis vector*, if the corresponding cut-off vector $Mp_i(s_{mi})$ can be detected, by the same token, we could also detect any other member of the latter's equivalence class as a *cut-off vector* corresponding to one from the former's equivalence class as the *basis vector*.

Hence the result.

■

Thus *all the general Final-state-vectors* are covered, classified into different disjoint sets of classes by each RMp_i, i=1..n. While traversing M_i as the primary Mpm, we perceive the global states that are in the equivalence classes created by RMp_i and so on.

---

[1] In general, the basis vector need not be an Mp-vector of the same local Mpm so long as it precedes the cut-off vector. An example will be given shortly.

The above lemma leads to the following definition.

2.12.2.3 Cut-off with respect to Local states

Definition 2.31 The Mpm-states whose Mp-vectors are cut-off vectors are referred to as the *cut-off states*. Each cut-off state forms a *leaf node* of the state-tree of M_i, for all i=1..n. For every *cut-off* state s_{mi}, there is a *basis* state s_{mj} such that:

$(s_{mj} < s_{mi})$ and, $B(Mp_j(s_{mj})) = B(Mp_i(s_{mi}))$.

This definition is seen to be consistent with Definition 2.30 and also with the original definition of cut-off in Definition 2.29, as will be illustrated by examples below.

Example 2.29 From Fig. C of *Appendix*,

$Mp_1(a_0) = (a_0\ p_0\ x_0)$ is the *basis vector* and, the corresponding

$Mp_2(a_2) = (a_2\ p_2\ x_2)$, $Mp_1(a_1) = (a_1\ p_1\ x_1)$ are the corresponding *cut-off vectors* such that: $B(a_0\ p_0\ x_0) = B(a_1\ p_1\ x_1) = (a\ p\ x)$.

a_2 and a_1 are *cut-off states* (in conflicting paths/configurations) corresponding to the basis state a_0.

Example 2.30 From Fig. C again, x_3 is a *cut-off* state and its *basis* state is s_0. This follows from: $B(Mp_3(x_3) = (c_1\ s_1\ x_3)) = B(Mp_2(s_0) = (c_0\ s_0\ x_0)) = (c\ s\ x)$ where $s_0 < x_3$; It is noted that s_0 belongs to M_2 and x_3 to M_3. x_3 inherited s_0 as an ancestor at the synchronization point $s_1{=}x_3$. It can be easily checked that if M_3 is simulated beyond the state x_3, the sub-tree rooted at x_3 would be *isomorphic* to the one rooted at x_0 thus satisfying the original definition of cut-off at Definition 2.29. This is because, the state x_3 can simulate all the events of M_3 that x_0 could (even though its environment in ΠF domain represented by the Mp-vector is different from that of x_0 as opposed to being identical, which is often more common).

Cut-off states truncate the possibly infinite Mpm-tree into a finite one, according to the given input specification of CFsm system, with every path of the state-tree terminating at a *leaf node* which is a cut-off state.

This is how the Mp-vectors and so local Mpm-states provide a meaningful representation of all the *cut-off* points, representing all the possible interleavings without the need for actually generating or traversing each of them, as the case may be.

2.12.3 Equivalence between Finite CMpms and CFsms

2.12.3.1 Equivalence Theorem II

Theorem 2.10 The *deterministic, finite model* of CMpms is equivalent to the *non-deterministic model* of CFsms. In other words, for every given *deterministic, finite model* of CMpms, there is a corresponding *non-deterministic model* of CFsms and *vice versa*.

Proof:

The proof follows from the *existence* of the set of mappings ΣB_i, i=1..n (and B).

Given a set of CFsms, it follows from ΣM *Generator theorem*, the set of functions B_i and hence the CMpms/ΣM can be inductively generated.

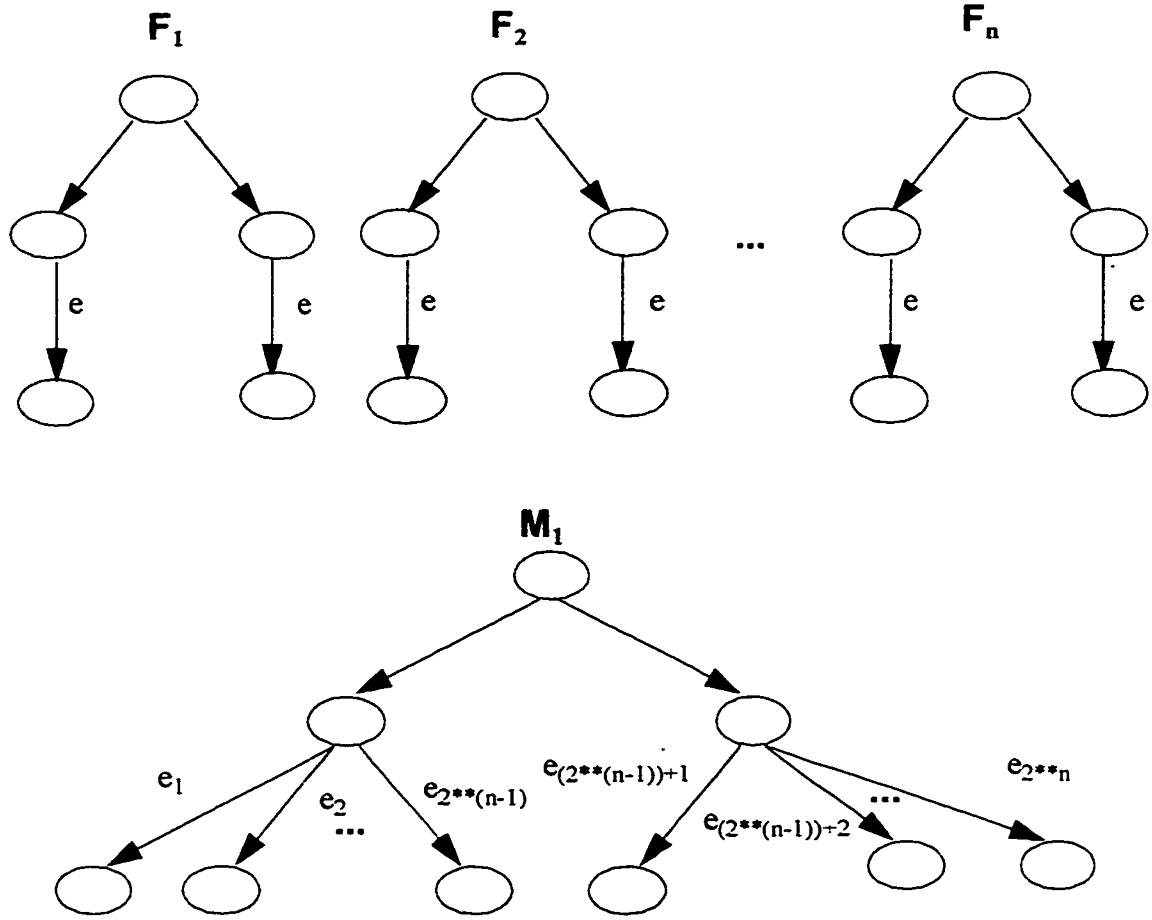
For the reverse direction, given a set of *finite model* of CMpms, unless their entities i.e., states and events are represented as *ordered pairs* of Fsm-entities and their occurrence numbers, it is hard to *actually* generate B_i,i=1..n and hence the corresponding CFsms, but the fact that there exists a set of B_i,i=1..n makes the argument.

Hence the equivalence. ■

2.12.4 Induced Local Conflicts due to Non-deterministic Synchronization

This issue is related to the non-enumeration of runs defined in a previous section. *induced local conflicts* of CMpms have been illustrated already in *Section 2.4.2.2*. When we generate CMpms from a given input system of CFsms, the induced local conflicts are caused by the non-determinism in the synchronization specification of the latter as illustrated in the figure below:

Fig. 11 Example of induced local conflicts due to Non-deterministic and Tight Synchronizations



The non-deterministic synchronization of true choices may cause a large number of induced local conflicts in the output CMpms; as a result of which, there may not be any significant non-enumeration of runs/configurations at all, as the number of local configurations/runs themselves tend to be the same as that of general configurations/runs, and exponential at that. The more the number of such *non-deterministic synchronization transitions* that are *tightly coupled* i.e., with a large number of participating processes, the more will be the number of induced local conflicts and the less will be the difference between the number of general configurations and local ones.

In the worst case, there may be an exponential number of induced local conflicts and correspondingly local configurations due to *non-deterministic synchronization of true choices*, given by all possible combinations of *synchronous states* of the n processes. The *exponential enumeration* of global-states due to all possible combinations of *synchronous* *local states* is different from that due to all possible combinations of *asynchronous local states*. The former is due to *non-deterministic synchronization of true choices* as we just characterized and the latter is due to *non-deterministic interleaving of artificial choices* simulating *true concurrency*. The latter is conventionally characterized as the *state-space explosion* of *total-order models* that simulate concurrency. The former also causes explosion of states but is inevitably induced by the given specification.

By reducing the non-determinism from the tightly coupled synchronizations or the number of participants from those non-deterministic synchronizations, the combinatorial explosion due to *true choices* can be avoided as well. As mentioned in the section on non-enumeration of runs, *summation lemma* can be applied meaningfully to derive general configurations that are different from the local configurations only when there is enough scope to do so as permitted by the combination of non-determinism and tightness in synchronization specification. Otherwise, the set of local configurations form an already enumerated set, and thus exhaustive.

Here again, we notice the *duality between true choice and true concurrency. Non-deterministic synchronization of true choices leads to combinatorial synchronous states and non-deterministic interleaving of true concurrent states leads to combinatorial asynchronous states.*

2.13 Justice, Fairness among Runs/Processes of CMpms

Justice and *fairness* are the notions related to the observation of a system that stretches indefinitely up to *infinity*. Since ΣM is essentially *infinite* and so is ΠM, it is easy to define these with respect to the CMpm system, representing a set of n concurrent processes.

2.13.0.1 Run, an Infinite entity

As seen above, finiteness of ΠM and so of ΣM is enabled by the concept of cut-off. This is not quite applicable to *runs* (and interleavings) as explained below:

Runs are defined to be conflict-free. In other words, at every conflict point, one of the states in conflict is chosen *arbitrarily*. The *cut-off vector* is equivalent to the corresponding *basis vector* only in the sense that all the vectors *possibly reachable* from the latter are reachable from the former as well.

So given a run $\Pi r \subseteq \Pi M$, the global-states actually *chosen* to be reached from the cut-off global-state need not be the same as the ones *chosen* from the corresponding basis vector, if the conflicts were resolved arbitrarily and differently from each other, at every conflict point encountered.

Therefore, the very nature of a run or its interleaving is inherently infinite in size due to the possibility of resolving conflicts in an arbitrary and unpredictable ways, at every conflict point, *without any pattern of recurrence*.

Though the runs $\Pi r \subseteq \Pi M$ are essentially infinite, due to finite generation of ΣM up to cut-off states, only a *finite truncation* of ΠM is generated and hence the *infiniteness of a run* is not observed/recorded in any $\Pi r \subseteq \Pi M$. But it is observable from its corresponding mapping onto ΠF because of the latter's cyclic characteristic/behaviour.

Example 2.31

The Mpm-state s_1 of M_2 with $Mp_2(s_1) = (c_0\ s_1\ x_3)$ is a cut-off vector corresponding to the basis vector, $Mp_2(s_0) = (c_0\ s_0\ x_0)$. Succeeding s_0, the state t_0 is a conflict point with two branches leading to u_0 and r_0 respectively. At u_0, there are two branches transiting to v_1 and v_0 respectively. Though not shown because of truncation, similar instances of conflict points will be possible from the cut-off state s_1 as well.

Now, the resolution of conflicts at descendents of s_0 is possible in many different ways arbitrarily at each cycle of the system execution which may not match *one on one* with each of similar conflict resolutions at descendents of s_1.

The above goes to show how the concept of cut-off can not be applied to terminate a run and define it as a finite entity. In other words, the pattern of a run can not be captured. However, *all the possibilities* of conflict resolutions and so the reachability of global-states are nevertheless predictable, using the cut-off states of the sum-machine, ΣM.

Fortunately, as will be elaborated in Chapter-3 in the context of specifying properties, we generally are interested not in posing properties that depend on the *exact pattern of a run*, but in determining what states are reachable and what are not; whether there is *some or all* *runs* satisfying a certain property or none at all etc., and these are manageable with cut-off states applied usefully, as will be covered in Chapter-4.

2.13.1 Classical Definitions of Justice & Fairness

The following definitions are quoted from [1].

Justice: A concurrent/distributed system is said to be *just* iff every constituent process of the system is *infinitely often disabled or infinitely often executed.* In other words, every enabled process should be eventually executed or disabled.

The above definition can be applied to a run, by posing the condition on the individual *transitions of a run* of the system rather than the process as a whole.

Fairness: A system is said to be *fair* iff: each process, *if infinitely often enabled*, then it will be *infinitely often executed.*

The above definition can be extended to a *run* by posing the same condition above on every transition of a run instead of a process in general.

We consider only *just systems* which means that every process is at least infinitely often disabled which implies, it must *at least be infinitely often enabled.* Such an enabled process is either chosen to be executed or disabled subsequently.

2.13.2 Unfairness in CMpms

Definition 2.32 An *unfair run* is one which is a result of excluding one or more states in favour of others in conflict with the former, *infinitely often*. Consider $C \subset C_{rmax}$ corresponding to a run Πr such that s_{mi} is *configurable* with C. (i.e., $C \cup C_i(s_{mi})$ is a configuration).

Consider the following set:

$\{C \subset C_{rmax} \mid s_{mi}$ is configurable with C such that:$(B_i(s_{mi}) = s_{fi}) \wedge s_{mi} \notin C_{rmax}\}$.

In the process of building C_{rmax}, whenever we reach the subset C with which a state s_{mi} that maps onto s_{fi} is *configurable*, it is *not chosen* to be added to C and some other state in conflict with it is chosen instead; this deprives the former of its membership in C_{rmax} corresponding to the run Πr.

We call Πr (and its corresponding map Πr_f) as an *unfair run* because it is unfair towards all the above states s_{mi} in ΠM, and the state s_{fi} in ΠF domain.

Since the infinite Mpms are truncated after the cut-off states, the manifestation of an unfair run is more easily observable in its mapping on ΠF domain.

Example 2.32 From the run Πr_f of Fig. 10 having the following as one of its *interleaved paths* is unfair:

{(a,p,x), (a,q,x), (b,q,x), (c,s,x), (c,t,z), (c,r,h), (c,s,x), (c,t,z), (c,r,h), (c,s,x),......}

The *unfairness* of the run is due to the fact that in this run the conflict between the states of M_2 mapping onto u and r respectively is always resolved in favour of the latter. The run is therefore *unfair to the state* u.

In the above example, there is a recurrence of global-states between (c,s,x) back to itself, with multiple cycles of the same sequence for ever. This is because, r is the only state in conflict with u. In general, there could be multiple choices other than u and in which case, the recurrence is difficult to establish except to say that u is *starved* or knocked out.

More interestingly, the unfairness to state u of F_2 is propagated to state d of F_1, through the *synchronization point* between d and u corresponding to that between d_0 and u_0 of M_2 and M_1 respectively. We say that process 1 (corresponding to F_1/M_1) is starved by process 2 or the system is *unfair* to process 1i.e., M_1.

2.13.3 Implementing Fairness in ΣM

It is more easy to account for unfairness among processes/Mpms than among runs due to the fact that *runs* are combinatorially too high to keep track of.

Let the relation *nl-conf* = (*conf* − Σ *conf*$_i$, i=1..n) denote the non-local conflict, where no two related states belong to the same Mpm.

Definition 2.33 The state s_{mi} is said to be in *asynchronous non-local conflict* denoted: (s_{mi} *anl-conf* s_{mj}) *iff* the following condition holds:

(s_{mi} *nl-conf* s_{mj}) $\wedge$ (*predec*(s_{mi}) *co* s_{mj}), where *predec*(s_{mi}) is the unique predecessor of s_{mi} in the state-tree of M_i.

In the above definition, in order for an input state and output state of an event of a process/ Mpm to be concurrent and in conflict respectively with a given non-local state, it must be a synchronous event, with *predec*(s_{mi}) as the input state waiting for its partner state s'_{mj}. In addition, s'_{mj} possibly has two successors s_{mj} and s''_{mj} in conflict such that s''_{mj} is a partner of s_{mi}, the former (s''_{mj}) propagating its conflict with its sibling s_{mj} to s_{mi}.

Unfairness Lemma:

Lemma 2.13 A process represented by Mpm M_j starves a process M_i, $i<>j$ iff: for some $s_{mi} \in S_{mi}$ and $s_{mj} \in S_{mj}$: (s_{mi} *anl-conf* s_{mj}) such that s_{mj} is a *cut-off state*, thus making the system unfair.

Proof: The result follows from the following argument:

Let *predec*(s_{mi}) = s'_{mi}.

(s'_{mi} *co* s_{mj}) => there is a configuration C with final-state components s'_{mi} and s_{mj}.

(s_{mi} *conf* s_{mj}) => the successor s_{mi} of s'_{mi} can not be added to make a successor of C .

s_{mj} is a *cut-off state* => For some C_{rmax}, a continuation of C, the infinite sequence of transitions made and the states added to C are such that: infinitely often, there is conflict between some states s''_{mi} and s''_{mj} resolved in favour of s''_{mj} such that: $B_i(s''_{mi}) = B_i(s_{mi})$ and $B_j(s''_{mj}) = B_j(s_{mj})$. In the first cycle, the conflict is between s_{mi} and s_{mj} and subsequently between the generic states s''_{mi} and s''_{mj} .

Hence the run corresponding to C_{rmax} is *unfair* and hence the system. ■

This is how the notion of asynchronous, non-local conflicts is applied as a fairness implementation tool. The application of this lemma in the *unfairness theorem* (to be presented) and model-checking will appear in Chapter-4.

2.13.3.1 Recording of asynchronous, non-local Cutoffs

During the generation of ΣM, the generation of synchronous states in each Mpm call for the following: In order to find the matching partner states (possibly more than one) corresponding to a synchronous input state s_{mi_in} of the local Mpm M_i, the partner Mpms are traversed exhaustively. When we come across the cut-off states s_{mj_cutoff} of those partner Mpms such that:

$(s_{mi\_in}\ sync_{in}\ s_{mj\_in}) \wedge (s_{mj\_out}\ conf_j\ s_{mj\_cutoff}) \wedge (s_{mj\_in}\ R_{mj}\ s_{mj\_out})$, we record $s_{mj\text{-cutoff}}$ as *asynchronous, non-local cut-off state* in conflict with s_{mi_out} where (s_{mi_in} R_{mi} s_{mi_out}) as defined in the previous section.

Example 2.33 In Fig. C, when finding *an input partner state* of s_0 of M_2 by traversing M_3 from $x_0 = Mp_2(s_0)(3)$ (the third component of the Minimal prefix vector of s_0), we find that: (z_0 *conf*$_3$ x_4) where x_4 is a *cut-off state* such that:

$(s_0\ sync_{in}\ y_0) \wedge (s_0\ R_{m2}\ t_0) \wedge (y_0\ R_{m3}\ x_4)$.

z_0 is a *synchronous output state*, propagating its local conflict (with x_4) to its *partner output state* t_0, asynchronous of s_0. It follows that:

(t_0 *anl-conf* x_4) by *inheritance of conflict.*

When t_0 is generated, the above information is recorded while traversing M_3 to generate all possible partners of s_0 in the *sync*$_{in}$ relation.

There is also an *indirect* inheritance of the above asynchronous non-local conflict as follows:

In the same example above, after c_0 of M_1 is generated, in order to find its input partner state (synchronous), M_2 is traversed from $s_0 = Mp_1(c_0)(2)$.

From (t_0 *anl-conf* x_4) , ($t_0 < u_0$) and ($u_0 = d_0$) it follows:

(u_0 *anl-conf* x_4) and so (d_0 *anl-conf* x_4) by *conflict inheritance* and the fact that x_4 is non-local for d_0 of M_1 as well as u_0 of M_2.

Therefore, d_0 is in *asynchronous, non-local conflict* with a *cut-off* state of M_3 through the synchronization of non-local event B which is necessary in order to enable synchronous event C.

2.13.4 Justice among Runs of CMpms

Definition 2.34 A *just run* is one in which *no* Mpm is preempted, i.e., not prevented from executing its local events, by others *infinitely often*; in other words every Mpm will be infinitely often *enabled/given its chance* to execute.

The above definition is consistent with the classical definition that a *just* system should have every process executed or disabled infinitely often.

Example 2.34 The following path of ΠF as viewed in ΣM of Fig. C of *Appendix*, completely preempts M_1 and M_2 in favour of M_3 and so represents an *unjust run*:

{(a,p,x), (a,p,y), (a,p,x),.....}.

This corresponds to the branch of M_3 in Fig. C starting at x_0 and ending at the *cut-off state* x_4, which indicates a cycle.

But within an Mpm, among two or more conflicting states that are ready to occur, some of them may be always chosen as opposed to others which represents *unfairness*, as formerly defined. The unfairness towards a synchronous output state in an Mpm is propagated to other Mpms through that synchronization point and an *apparent, induced injustice* may result.

For example, as a result of an unfair run illustrated in the first example above, M_1 is starved since it waits at state c_0 for ever and hence an *induced injustice* due to unfairness results.

We consider only just runs in this work except the *unjust* ones induced by *unfair* runs. This assumption is embedded into the model checker algorithms discussed in this chapter. This can be made clear by the example above.

For instance, in case injustice were to be allowed, the above shown infinite sequence {(a,p,x), (a,p,y), (a,p,x),.....} of global-states exclusively executing the events of M_3 indefinitely would have been considered a *legal run* and this in turn would have falsified all the formulae with *universal run operator*, (A_r) which could otherwise be provably true.

2.14 Generation Algorithm of CMpms , ΣM with respect to CFsms

The generation of ΣM consists in simulating the Fsms corresponding to every *primary* Mpm generated. Every primary Mpm's generation also involves a partial generation of the other Mpms (corresponding to other Fsms) that are *secondary* , as partners of the former in the synchronous events. In other words, generation of every Mpm-state involves the generation of all those states in its local configuration/upward closure, that belong to local as well as non-local Mpms.

The Mpms that are currently secondary have their own turns to be generated as primary ones in an arbitrary succession. Depending on the generator algorithm (Algorithm (i) or

(ii) listed in Chapter-4), states of these Mpms (that acted as secondary ones previously) already generated may or may not be generated/visited again.

Each of the Mpms is an expanded, *possibly infinite* version of the corresponding Fsm. Hence the Mpm-states retain their *locality* i.e., the component identity, at the same time representing a *minimal globality* by the association of their *respective Minimal prefix vectors.*

When a secondary Mpm becomes the primary one, the subset of its states already generated during its secondary status and their Mp-vectors remain the same as the ones to be generated as members of a primary Mpm. This is because, every transition r_{tmi} of M_i is uniquely generated as a function of its uniquely generated input state (which is the output state of a unique previous transition) and the input transition r_{tfi} of the Fsm F_i being simulated. This unique association of *past and present* with a state *minimally*, gives rise to the important definition of the *Minimal prefix* Mp_i,i=1..n which is a *one-to-one function*, and is independent of the order in which the primary Mpms are generated or the fact whether it is primary or secondary. Mp is a *conceptual* as well as a *pragmatic* notion because, by defining it and associating it with every state, it also gives a clue as to how to generate that state and those on which it depends causally from other Mpms as well.

During the generation of every *synchronous output state*, the cut-off states in *asynchronous non-local conflict* with it are kept track of, and recorded if any. There are many possible partner input states of a synchronous input state and a corresponding unique synchronous transition is generated in every case.

The algorithm is *recursive* and adopts a *distributed, depth-first search* and generation of input Fsms and Mpm-states respectively. More about this traversal and the quantitative generation complexity will be discussed in Chapter-4 since the traversal for generation and verification adopt a similar core methodology.

Thus, we generate all the Mpm-states as members of their respective Mpms, *simulated individually as primary ones, in any arbitrary order*. Together with the Mpm-states, the set of all Mp_i,i=1..n is also generated and stored as the *labels* of respective Mpm-states. ΣMp_i is only implicit in the representation of the example shown in Fig. B of *Appendix* and explicit in Fig. C and Fig. D. Using these Mpm-states and their respective Mp-vectors, the set of all global-states of ΠM (and so of ΠF) can be generated dynamically and *monotonically* by the *sum/set union* of these states to form different configurations corresponding to different runs.

Because of the association of the Mp-vectors as the *labels* of Mpm-states, the traversal of the local Mpm-states alone of the state-tree of an Mpm (corresponding to the traversal of its local runs) takes care of the traversal of non-local Mpms as well. When a particular state is reached in the local Mpm, all the *minimum* number of states traversed by the non-local ones so far can be deduced by using the Mp vector and the *upward closure* of all its components. This is further simplified by the application of the *disjointness theorem*: backtracking the unique *set of n paths* of the n state-trees from the corresponding components of the Mp-vector up to the *initial-states* of the trees. Thus an *abstract PO structure/ configuration* is concretized into paths of *finite automata*.

The detailed generator algorithm is presented in Chapter-4, in the context of model-checking.

2.15 CMpms, CFsms and Formal Languages

In what follows, we outline certain rudiments of automata and language theoretic issues in the context of CMpms model that summarize the purpose of the theory of CMpms presented thus far, and identify the perspectives of this model in the realm of classical languages and automata theory in the process.

CMpms form in general *infinite, deterministic, synchronous automata* as contrasted to CFsms that form in general *non-deterministic, finite, synchronous automata*. Given the latter, what we have shown is that there exists at least one set of CMpms (possibly more, that are *isomorphic* to each other (depending on the *auxiliary functions* chosen by the generation algorithm) which simulate the given set of CFsms and therefore their *infinite state trees* need not be expanded beyond the *cut-off points*. This truncated set of CMpms, by virtue of the *surjective mapping* B, is *equivalent* to the simulated set of CFsms, *up to* their global states and transitions.

It should be noted here that any arbitrary set of *infinite* CMpms need not possess any *cut-off points* that are isomorphic at all to certain of their ancestors at the end of finite prefices, and so there may not exist an equivalent set of CFsms. But their inherently infinite size makes them more powerful to open up the class of possibly recognizable sets of infinite PO-structures.

Modeling, Logical and Algorithmic Applications of CMpms:

- The set of *n disjoint state-trees of* CMpms/*sum machine* correspond to an *acyclic graph,* whose relational structure is a *labelled Partial Order* of Mpm-states each labelled with its *Minimal Prefix*. Thus, CMpms combine the modeling advantage of the *labelled PO* to represent *true concurrency* and the *branching-time semantics* of the *labelled trees* to represent *true choice* in the abstract domain. The modeling advantage of Mp as labels is exploited in the *extended partial order model* of the temporal logic CML, as will be introduced in Chapter-3.

- The splitting of the *labelled PO structure* of the Mpm-trees of ΣM into many *conflict-free structures* called *configurations* and extracting each configuration as a *set of n paths* of the respective *n labelled trees* has an algorithmic advantage in the concrete domain, by the application of the *operational semantics* of individual Mpms to implement the *model checker* for the logic CML above. The complexity is cut down as well by exploiting the localities/identities of Mpms combined with the application of the *labels* of each node/state in the tree structure that are *Mp-vectors,* an extended notion of *causality* and *concurrency.*

- Due to the infinite size of CMpms in general, and their distributed nature with their respective identities/localities intact, visualizing and implementing the classical notions of *justice* and *fairness* are now possible.

Issues of Formal Language Theory:

The truncated version of i.e., *finite, deterministic automata of CMpms* is provably equivalent to the *non-deterministic model of CFsms*, as shown. There are many classes of *finite automaton recognizable languages* respectively over different classes of *acyclic graphs* such as *Words strings, Trees, Traces, Grids and PO structures*; of which, the finite automata over PO structures have not been quite established as those over *words* and *trees* [37],[2],[34]. The CMpms model covers the *language for labelled PO structures (connected into a set of trees*). The more interesting aspect is that, the language of these labelled PO structures is definable in a *monadic third order logic*, as will be introduced in Chapter-3. Therefore it seems viable to extend the *Buchi, Elgot theorem* mentioned in [37], from the language of *words* to language over *conflict-free, PO structures (configurations)* connected into a set of trees, to cover all the *conflicts/branches of time*.

2.16 Complexity Saving with Sum Machine of CMpms

2.16.1 Complexity Lemma

In Section 2.4.2.1, we mentioned that :

(i) The local conflicts $\Sigma conf_i$ are the original sources of conflicts and the global ones ($conf - \Sigma conf_i$) are the local ones propagated by causal-dependency.

(ii) The Mp label of an (Mpm-)state, being a state-vector decides all the states that are causally dependent on it, both locally and non-locally.

Therefore, by maintaining Mp-label and *local conflicts* of every state, it is clear that we can deduce all the *global conflicts*. The significance is that, the local conflicts decide number of *maximal local configurations* (and so that of *local runs*) and the *global conflicts* decide the number of *general maximal configurations* (and correspondingly *global* runs). Since the former is much less than the latter (assuming the specification is conducive to it, by having a minimal number of non-deterministic synchronizations that are tight), and can be used to deduce them using the *Summation Lemma*, we do not have to enumerate all of the global conflicts and equivalently the global configurations/runs. The following Lemma formalizes the above.

Lemma 2.14 The following statements are *equivalent* (in addition to the equivalence within each):

(i) Local scanning of labelled Mpm-states <=> Global scanning of Final-state vectors .

(ii) Scanning of maximal local configurations, C_{rmaxi} , i=1..n <=> Scanning maximal general configurations, C_{rmax}.

(iii) The global relation *conf* is derivable from the *disjoint union* of local conflict relations, $\Sigma conf_i$, i=1..n.

Proof:

The proof of (i) follows essentially from *Monotonicity Lemma* at Lemma 2.8 to generate the Final-state vectors. The labelling of every Mpm-state refers to its Mp vector.

The proof of (ii) follows from *Summation Lemma* at Lemma 2.9, since every $C_{rmax} \in$ Cset can be viewed as $C_{rmax} = \cup C_{rmaxi}(s_{mri})$, where $Fsv(C_{rmax}) = s_{mr}$.

The argument for (iii) follows essentially from ***conflict-inheritance property*** at Property 2.1. It also follows from (ii) as follows: A configuration is *conflict-free*. Therefore, the number of general configurations is proportional to the cardinality of the global conflict relation, $|conf|$ and similarly, the number of local configurations to $|\Sigma conf_i|$, i=1..n.

Since all three statements (i), (ii) and (iii) are ***methods of generating global-state vectors*** from local Mpm-states, they are equivalent.

■

The lemma above leads to the following theorem.

2.17 Complexity Theorem I

Theorem 2.11 Unlike in the product machine ΠF of a given set of CFsms specification, in the case of ΣM generated with respect to those CFsms, there is no *state-explosion* caused by the following, as permitted by the given CFsm specification:

(i) enumeration of *all* the *runs* (of ΠF corresponding to those of ΠM).

(ii) enumeration of *all interleavings of every run.*

Proof: The proof is an application of the above lemma. Since a *maximal configuration* C_{rmax} corresponds to a *run* $\Pi r \subseteq \Pi M$, whatever holds good for the former entity is true with the latter i.e., a *run* as well. Therefore, if every general configuration can be reached by the *union of local ones*, every *global run* corresponding to a *general configuration* can be generated by local runs as well. This means that there is no enumeration of all the runs.

By local scanning of *partially-ordered* Mpm-states we simulate the global scanning of *totally-ordered* state-vectors, by *Monotonicity Theorem*. In that case, interleaving does not come into the picture because it depends only on the path of global scanning.

Alternatively, the proof of (i) also follows from the property of *non-enumeration of runs*, as discussed in Section 2.10.3, as Property 2.6.

The second part (ii) is due to *interleaving insensitivity* of ΣM as stated in Property 2.7.

The above is *when permitted by the specification* because, when the *degree of synchrony* is very high in the given CFsms, the *causal dependency-order* $\leq$ among the Mpm-states may *degenerate to a total-order* and there are no interleavings to be avoided. When the *non-determinism and tightness in the synchronization increase*, the number of local runs may *degenerate to as many as global ones* and hence there are no global runs to be avoided as well. These are the *degenerate cases* posed by the specification and the above non-enumeration of runs and interleavings can be usefully applied only in the absence of such cases or cases tending towards them.

▩

The detailed discussion of the proof is as follows:

Details of proof of (i):

Runs originate from *conflicts*, as modeled by *conf* relation. Since the state-trees of the Mpms are disjoint, many entities of the sum-machine are localized and distributed. Thus only the local conflicts are explicitly represented by $\Sigma conf_i$, where $|\ \Sigma conf_i\ | < |conf|$. In the case of product-machine, the conflict relation $conf_g$ is among global-states where , $|conf| < |conf_g|$. This is appreciable since all continuations of every component are considered from every global state, in a homogeneous or non-distributed fashion.

Instead, by the locality of conflicts, only the local runs corresponding to local configurations are traversed and the required global ones deduced using the Mp-vectors stored along with every Mpm-state. Thus there is no need of enumeration of *all global conflicts* and so of all *possible general runs* corresponding to *general configurations*, except the local ones. Every global state is generated essentially by the *addition i.e, union* of its component states.

Details of Proof of (ii):

Even if there is a single *run*, in the case of product machines, there is *non-deterministic choice* defined by $choice_{\text{non-det}}$ due to *interleaving* the execution of the component machines in all possible *sequences*, artificially.

The *sum machine* corresponding to every *interleaving* ΠI_r of Πr is the same, as that of its parent *run* Πr. Consequently, the *enumeration of all the interleavings of a run is unnecessary* in the case of ΣM; i.e, there are no artificial *non-deterministic choices* and *sequences* in it.

Depending on the order in which the local Mpms are traversed, an arbitrary interleaving takes place automatically till the required global-state is reached. The reachability on all interleavings can be checked by extending the *causality theorem* as will be proved as the *interleaving theorem* in Chapter-3 after the introduction of the temporal logic CML. The theorem states that the causality/dependency-order $\leq$ among the elements of a given state-vector enables the deduction of the reachability of that vector on all interleavings given the reachability in any one of them.

The model-checker algorithm in Chapter- 4 incorporates the features of temporal logic introduced in Chapter-3 along with sum-machine, as mentioned. The logic is required to formulate the properties to be verified which will be introduced in Chapter-3 and further in Chapter-4 on model-checking / verification.

2.18 Summary of CMpms

Theoretically, we have developed a model that represents *causality, sequence, choice* and *concurrency* in their *true* form *faithfully* as exhibited by the given input specification of Communicating Finite state machines. *Concurrency* originates from *simultaneity*, and is reinforced by causality rather than being its complement. This enables *causality* among states to be modeled *orthogonal* to *concurrency*, a characteristic that is applied in the traversal of local runs to generate global/general ones and in the deduction of a property of all interleavings from that of one.

The above conclusive note will be elaborated in subsequent chapters. But for now, it has turned out that from our primary attempt to cut down the enumeration of all interleavings, we have also accomplished the non-enumeration of runs as a secondary but a very desirable bonus. It follows from the fact that both categories of non-enumeration are the result of non-enumerating the global-states, by maintaining only the set of all local states of processes that correspond one on one, to a ***minimal set of global-states*** called Minimal prefixes.

In practice,

- We propose a fixed set of Communicating Minimal prefix machines (CMpms) constituting a state-oriented, *partially-ordered, sum machine*, which overcomes the demerits of the traditional *totally-ordered, product machine*.

- The reachable state vectors of the *sum machine* of CMpms, ΣM *correspond one-to-one* with what are defined as its *configurations* as well as to the *global states* of the product machine of CMpms, ΠM.

- A given set of CFsms is assumed as the input specification. It is shown that a *surjective mapping* from ΠM *onto* the product machine of CFsms ΠF, exists.

- Therefore, composing the above two mappings mentioned, we get the mapping from ΣM *onto* ΠF. The *finiteness/cutoff* of ΣM is defined corresponding to the finite ΠF of given input CFsms. The equivalence between the deterministic and finite model of CMpms and the non-deterministic one of CFsms is shown.

- The properties that are traditionally verified on ΠF with high complexity and low expressiveness are now verifiable on ΣM with the advantages of reverse results: low complexity due to non-enumeration of runs as well as of interleavings (to an extent permitted by the given specification) and the high expressiveness of the system properties starting right at the modeling of *sequence*, *choice* and *concurrency* in their true form as exhibited by the input specification, all at the same basic level of computation.

2.19 Comparison and Contrast of ΣM with ΠF

The following tabulation summarizes the advantages of viewing ΠF *virtually* upon the *real* machine ΣM, rather than actually generating it as in the traditional methodology, given an input specification of a CFsm system.

TABLE 1 Product Machine of CFsms Versus Sum Machine of CFsms

Product Machine, ΠF	**Sum Machine , ΣM**
Totally-ordered and *interleaved* model.	*Partially-ordered* and *non-interleaved* model.
By simulating concurrency with choice (non-deterministic) and sequence, the three relations viz., *sequence, concurrency* and *conflicts* are all corrupted.	All the three relations are faithfully represented in their true form, using a *global, causality relation* that is partial.
Locality/identity of processes and hence their respective entities is not maintained.	*Locality* is very much kept track of and utilized to advantage.
Runs and *interleavings* are not distinguished from one another.	*Runs* and *interleavings* (within runs) are distinct.
To impose *Justice* among processes is not feasible. Defining *fairness* is not straightforward.	*Justice* among processes is easy to impose since locality of processes is defined. *Fairness* is easy to handle due to the distributed simulation of essentially *infinite Mpms*, each *primarily*.
State-space explosion occurs due to exponential enumeration of runs and interleavings within each run respectively.	No state-space explosion since the runs and interleavings are not enumerated.

Chapter 3
Computational Mpms Logic (CML)

3.1 Logic in the Context of System Verification

As mentioned in the introduction, the advantage of logic in the context of system verification is *quoted* in [35][1] as follows:

"The application of logic is an alternative (to the verification tools without the use of formal logic) that has a strong support from a large segment of software engineering community in the area of system verification, specifically in the efficient search of the entire space of possible behaviours. More than the characteristic of infallibility popularly attributed to it, what logic accomplishes is the *efficient* search of *combinatorially large or even infinite state spaces*, for all the known types of bugs in a practical amount of time".

In order to make use of a model of a (concurrent/distributed) system particularly for the purpose of verifying its properties, a formal platform is needed to formulate such properties. Since concurrent systems have processes with their own *local time scales*, consolidation of these time scales is necessary, in the context of the verification of such systems in order to arrive at their *global properties*. This warrants the use of *temporal as well as spatial* (to be introduced) *logic* as a specific logic tool in the verification.

A logic defined formally, i.e., a formal logic is associated with a *language* using which the formulae of the logic are expressed. The language has a *syntax* and *semantics*. The semantics of the language and therefore of the logic are defined with respect to a *computational model* (simply referred to as a *model*) usually denoted by a triple *structure*, to be introduced in the sequel. The terms *model* and *structure* associated with a logic are often used synonymously in a semantic connotation through out the following.

Temporal logic offers a platform with its modal operators to analyze the *past*, the *present* and the *future* properties of the system, posed as formulae in the logic. The *computational model* of the system in turn serves as a platform for the semantics of the logic, by virtue of being the underlying entity whose properties are specified and validated as expressed by

[1] The actual quotation is from the preface of the volume of this reference [35].

the formulae of the logic. A good *logic structure* is designed in such a way that it exploits the features of the underlying *computational model* and vice versa, in order that the best features of both are brought out and utilized in practice by the *model-checker* i.e., the verification method, based on them. In other words, the two entities viz., the logical structure and the computational model have to be compatible with each other. Unless the two reinforce each other, by being mutually compatible, their individual richness cannot be put to use. This is elaborated as follows:

The model supporting the logic is amenable for implementation when it provides an *operational, automata theoretic* support. A temporal logic is considered rich when it combines the partial-order semantics with branching-time semantics. There are certain branching-time temporal logics over *partial-order* structures without the support of a compatible model, such as *F(B)* [2] and the logic reported in [21]. This is because, the underlying model based on *occurrence net* can represent only a *single conflict-free run* at a time, as its process structure. Therefore, the *branching-time* feature cannot be implemented by the *PO semantics* of the model, compatible to the logic. On the other hand, if the logic structure is based on a *total-order* model such as CTL, checking its formulae *directly* on top of *concurrent automata based model* will not be fruitful in utilizing the partial-order semantics of the latter. This is clear from the known results that the complexity of verifying a CTL formula over a concurrent set of automata is not improved i.e., still PSPACE *hard*, just as in the traditional product machine or the *Kripke* structure.

The *partial-order* based *computational model* of CMPms forming a sum-machine is not just a collection of concurrent automata but endowed with the special property that comes with it, viz., of the *Minimal prefix* that forms the backbone of the model. By introducing a logic that is tuned to this property of the model and its extensions by means of its axioms and inference rules, we ensure the design of a compatible *partial-order* based logic on top of the *sum-machine* model, and then show that the *total-order* logic on top of the *product-machine* can be checked using the former pair. The combinatorial explosion due to global-state enumeration present in the *total-order* model/logical structure is thus alleviated. Consequently, the exponential complexity , as permitted by the *non-determinism* in the *specification* given and *property* checked.

3.2 The Perspective of CML, Abstract and Concrete

Abstractly, we define CML (Computational Mpms Logic) as a logic over possibly *infinite, labelled Partial-Orders*, derivable from the *sum machine* as explained in Chapter-2. Since these PO structures can be *finitely truncated* and extracted as a *set of n paths* of *n* respective *finite Mpm-trees* according to the *disjointness theorem* at Theorem 2.2 of Chapter-2, it is easy to deduce the properties of these PO structures with the *operational semantics* of the *finite automata*. CML has a *branching-time semantics* as well by relating the local conflicts of the individual Mpm-trees with the global conflicts among runs or equivalently, the maximal configurations formed by the *labelled PO*. Thus CML is a logic that combines a *PO semantics* with *branching-time* one.

The *PO semantics* mentioned above has a clear link with the *Total-Order semantics* that is provably equivalent to the former, as will be shown. This equivalence only adds on the advantages of the latter that has strong ties with the concrete, operational domain to those of the former. By the above link, we are able to reason about the global-states of all the *interleavings* of every *run* without explicitly enumerating *every* one of them.

We propose three versions of CML, each one as a *monadic third-order logic* (as will be explained shortly) : $CML_{\Pi F}$ whose underlying *total-order model* is the *product machine* ΠF of a given CFsm system ΠF, $CML_{\Pi M}$ based again on the *total-order model* of the product machine ΠM, $CML^{*}_{\Sigma M}$ which is based on the *extended partial order model* of the *sum machine* $\Sigma^{*}M$.

ΠF is viewed as a (surjective) mapping of ΠM which in turn is viewed as a *virtual product machine* through the *real* sum machine ΣM. This follows from the ΠF and ΠM *Generator theorems* respectively of Chapter-2.

Since ΠF is viewed and generated as a *virtual machine* by the *physical* sum machine ΣM, $CML^{*}_{\Sigma M}$ serves as the actual logic from which $CML_{\Pi M}$ and $CML_{\Pi F}$ are derived/visualized. More precisely, the set of temporal formulae to check both *safety* and *liveness* properties of ΣM and hence of ΠF are expressed with operators for qualifying both *true concurrency* and *true choice* (*conflicts*) as exhibited by the specification, and modeled in ΣM. Thus the enriched expressiveness of CML stems from ΣM and is carried over to ΠF

in the specification of the latter's properties. These ideas will be formalized in the following.

3.3 Background of CML

The syntax and semantics of the logic CML is close to that of *F(B)*[2], a *partial-order oriented, branching-time, monadic second-order* temporal logic on top of *occurrence net* model. Unfortunately, *F(B)* can not be implemented , as it is the case till to date, that there is no *finite acceptor for prime even structures with conflicts*.

Both CML and *F(B)* have borrowed the *modal operators* from CTL, though the latter is not a partial-order logic. Both CML and CTL address the state-based product-machine as their underlying model. But in the case of CML, the product machine is *viewed as the mapping* of sum machine that makes all the difference in their semantics as well as complexity of model-checking later on. Further comparison and contrast will be made between CML and CTL in the summary and later chapters.

3.4 CML, A Branching Space-time Logic

3.4.1 Branching Time Aspect

Depending on whether or not *conflicts* of a system are taken into account in the logic, we have *linear-time* and *branching-time* temporal logics. Both are popularly in vogue in the literature. The former assumes the model of the system to be a disjoint *set of sequences* representing *runs* while the latter views those sequences to be connected into a *tree*. The branches of the tree take place due to *true choices* or *conflicts* of the system in time scale and hence the name branching-time[5][2]. An account of the taxonomy of different computational models of a concurrent/distributed system and temporal logics was given in Chapter-1.

The *branching time* aspect of CML is thus linked to the *conf* relation in ΣM, whose structure is a *disjoint set of trees*. Two conflicting configurations represent two different *continuations (future)* of the initial configuration but are not the continuations of each other. Thus they can be considered as two different branches in the time scale. Two conflicting successors of a given configuration represent two different *branches of time* in the immediate future.

The conflicts of ΣM are originated by $\Sigma conf_i$. In other words, the source of conflicts is local and distributed and so are the continuations of configurations and hence the branches of time. This has an important implication in the distributed and localized model checking, since the set of local trees corresponding to local runs (manifesting as local configurations) alone are *statically* stored.

As discussed in Chapter-2, the cardinality of the conflict relation $|conf_g|$ is much higher in ΠM than that of the relation $|\Sigma conf_i, i=1..n|$ of ΣM. This is because, from a given global-state of the former, the conflicts of each component have to be considered; in other words, the successors of every component have to be visited. Whereas in the case of ΣM, only the local conflicts are considered during the traversal of a given Mpm; only upon the success of local traversal (according to the property checked), any non-local traversal is carried out. While scanning the local conflicts, the global ones are automatically accounted for according to *Complexity Lemma* I (Lemma 2.14).

3.4.2 Branching Space, A New Dimension of CML

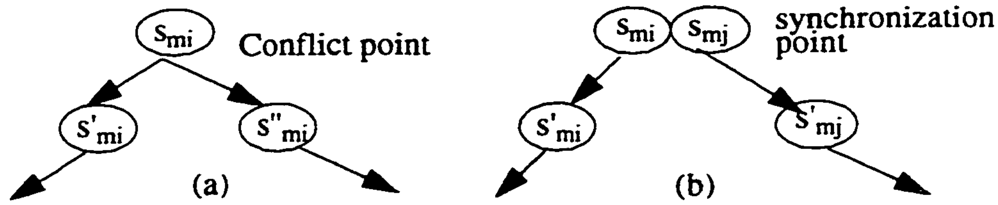
The *branching space* aspect of CML is related to the branching-off in ΣM from the *space state-tree* of one Mpm to that of another. This dimension of branching is through *synchronization points* among the otherwise *disjoint set of state-trees* of the Mpms. These points of contact are contributed by the *synchronous output states*. These states manifest themselves as non-local components of *Minimal prefixes* of a given primary Mpm and serve as *handle states* in branching from the local, (primary) Mpm to the other non-local (secondary) Mpms.

We refer to each of these as *handle states* because of the following: we continue the traversal of the current configuration/run from these states in the secondary Mpms; these non-local Mpms (to which the branches can be made later), have already progressed up to these states to enable the primary Mpm (from which branch is made) to reach its *current state*, through the necessary synchronization points/states.

3.4.2.1 Duality of Conflict and Synchronization Points

The *synchronization points* for *branching space* are analogous and dual to the *conflict points* for *branching time* as illustrated below:

Fig. 12 *Branching time* (conflict) and *Branching space* (synchronization) points



In the example of Fig. 12 above, (a) shows a sub-tree of the state-tree of some Mpm M_i and (b) shows a couple of sub-trees of both M_i and M_j.

In (a), s_{mi} represents a *conflict point* from which there are two branches (of time) such that the states s'_{mi} and s''_{mi} of the two branches respectively are in conflict with each other. From the *conflict-inheritance* property, all the descendents (in the order $\leq$) of s'_{mi} are in conflict with those of s''_{mi}.

Similarly in (b) of Fig. 12 above, s_{mi} and s_{mj} as *synchronous output states* (s_{mi} $sync_{out}$ s_{mj}) together represent a *synchronization point*. These two states that are simultaneous, also represent the *source of concurrency* as mentioned in Chapter-2. The concurrency is inherited by all the descendent states of s_{mi} and s_{mj} that are reachable *asynchronous* of each other. Thus, assuming s'_{mi} and s'_{mj} are reached asynchronous of s_{mj} and s_{mi} respectively, we have: (s'_{mi} *co* s_{mj}), (s_{mi} *co* s'_{mj}).

3.4.2.2 Branching Space Versus Branching Time

So, we see that a *conflict point* formed within a state-tree of an Mpm by a single state gives rise to different *branches of time*, that continue independently to give rise to different *futures*. Likewise, a *synchronization point* formed by multiple partner output states belonging to disjoint state-trees of respective Mpms gives rise to different *branches of space*, as many as there are partners, that may progress independently/asynchronously in *space*, i.e., within their respective localities of state-trees.

The branching in space is orthogonal to branching in time. The branch of *time* or continuation in time while executing the events of one Mpm *before* branching in space is maintained even *after* the branching in *space*, during the execution of events at a different Mpm.

3.5 CML, A 'Monadic Third-order' Logic

3.5.1 Third Order Logic

A *higher-order* logic as opposed to a *first-order* logic, allows the predicate names to have other *predicate names* or *function* symbols as arguments. Furthermore, quantification can be applied not only to variable symbols, but also to function symbols and to predicate names [43].

Temporal logics typically are based on higher-order theories as they use the modal operators as predicate names that take propositions as their arguments to express the temporal properties which in turn are in the scope of certain quantifiers. For instance, CTL[1], [5] is a traditional *second-order logic* where the path quantifiers operate on *first-order formulae* of modal predicates.

A *monadic second-order* formula begins with a prefix of quantifiers followed by a *first-order formula*. More detailed and formal definition can be found in [37] and its cross-references.

A *monadic third-order* formula begins with a *prefix of quantifiers*, that quantify a *monadic second-order formula*.

The definition of CML as a *third-order logic* in the following section exemplifies the above concepts.

3.5.1.1 Break-up of a Monadic Third-order Formula of CML

(i) The *innermost first-order* sub-formulae will consist of *modal predicates* with modal operators qualifying the *propositions*. These are also referred to as the *interleaving formulae*, to be operated upon/quantified by the *interleaving quantifier*. A *run* as defined in Chapter-2 is a *conflict-free product machine* with a corresponding *maximal configuration* of the *sum machine*. Since a *run* consists of many *interleaved paths*, we extend the above mentioned *interleaving formulae*, with the *run quantifiers/configuration quantifiers*, operating on them.

(ii) The *second-order formula* resulting from the *interleaving quantifier*, quantifying a *first-order modal formula/interleaving formula* is called a *run formula* since it will be quantified by a *run quantifier*.

(iii) The *third-order state formula* results from a *run quantifier* operating on a *second-order, run formula*.

The above formulae are in *monadic* form. By inductively nesting the above categories of formulae, we get still higher-order formulae than the third-order ones that are non-monadic. The term monadic probably was coined to define simply one elementary level of arriving at the state-formulae with only one *run operator/quantifier* and an *interleaving operator* without any nesting. By restricting ourselves to *monadic formulae*, we isolate out the higher than third-order formulae due to nesting. Consequently, we manage to check the entire formula *in one pass*, without breaking it up into multiple levels of *state-formulae* and consequent need for *labeling algorithms* to verify the entire *nested formula*. This issue will be elaborated in Chapter-4 on model checking.

3.5.2 Branching Space and Interleavings

Since ΣM is the underlying model of CML, any global-state of ΠM is reachable as the final state vector of a configuration C. From the *interleaving insensitivity/independence* property of a configuration, it follows that C represents all the interleavings of the run in ΠM domain corresponding to the configuration C. The *causality theorem* is recalled from Chapter-2:

$(co \wedge \leq) \Leftrightarrow Null$,

which is exploited to check if the components of the final state vector Fsv(C), wait for each other by checking their *causal dependency order*.

Given Fsv(C), if the components are only related by the concurrency relation *co*, and *unrelated* by $\leq$, we say that the components *possibly wait for each other* in order to *hold simultaneously* at some instant of time. We also say that the vector Fsv(C) is reachable by *some interleaved path* of a *partial run* corresponding to C. The fact that the components of Fsv(C) are unrelated by $\leq$ means that they are not bound to wait for each other.

On the other hand, if these components are related by $\leq$ in such a way that they are bound to *wait for each other necessarily* as will be elaborated, it means that irrespective of the interleaved order of execution, the vector is reachable. We say that Fsv(C) is reachable by *all interleavings* of the partial run corresponding to C and the components *must wait for each other.* This will be formally stated and proved in the sequel.

From the above paragraphs we see that, it is possible and useful to have an *interleaving operator* along with the *run* operator, the latter *alone* corresponding to *branching time*. We call the former, the *branching space operator* because, it stems from the *causality theorem* which in turn originates from *simultaneity* (*strong concurrency*), represented by *synchronization points* that link the otherwise *disjoint Mpms in space*. *The synchronization points in space are dual to conflict points in time* as cited before.

Two main advantages of interleaving operator are:

(i) *It takes away the ambiguity or non-determinism in expressing the reachability of a state and so in the checking of a predicate. This enhances the expressiveness of all properties, since any property essentially asserts the reachability of global state(s), aided by the modal and branching operators. This results in a tractable procedure, since it is deterministic.*

(ii) The reasoning of interleavings can be made *orthogonal* to that of runs. This way, both *concurrency* and *conflict* can be analyzed at the *same basic level*, independent of each other. This has been precisely the very goal of our research, which fills the void entry of the survey table of Reisig [2] as explained in Chapter-1.

3.5.3 Branching-Time and Runs

Branching-time semantics in CML is contributed by the *conflict-points* of the individual Mpm-trees. By maintaining the *localities* of individual processes/Mpms,

(i) we maintain the original sources of *conflicts* sprouting within the processes alone, and *only manifest globally through causality.* (i.e., the source of conflicts is *intra* and not *inter* process).

(ii) the distinction between *runs* and *interleavings* is made easily, since the latter does not come into the picture within local states at all. (i.e., the source of interleavings is *inter* and not *intra* process).

Thus CML turns out to be a richer logic with the *branching space* feature associated with *concurrency* and expressed by interleaving operators, in addition to the *branching time* feature associated with *conflicts* within a given *space* or an Mpm's *state-tree* as expressed by *run operators*.

3.6 Building Blocks of CML

Before formally defining the *syntax* and *semantics* of CML and its *structures*, we introduce the following input assumption followed by certain basics:

We are given a CFsm system constituted by F_i, i=1..n that communicate by a specified set of synchronization events (each with specified partner Fsm-identities) and a set of *labeling functions* p_{fi} i=1..n, assigning *a unique atomic proposition* to every Fsm-state of F_i, i=1..n respectively such that:

p_{fi} : S_{fi} --> Ap_{fi} is a *bijection* from S_{fi} to A_{pfi} where,

S_{fi} is a set of local states of F_i and

Ap_{fi} is a *set* of *atomic propositions* of F_i.

Example 3.1 From Fig. A of *Appendix*,

$p_{fi}(a) = ap_a$, $p_{fi}(b) = ap_b$ etc.

Assuming ΠM is the *product machine* of a set of CMpm system, $\{M_i, i = 1..n\}$ generated with respect to the given CFsm system such that:

$B_i : M_i$ --> F_i, i=1..n and,

B: ΠM --> ΠF is a *surjective map* as defined in Chapter-2,

we define the following:

Definition 3.1 Given the set of atomic propositions Ap_{fi} over S_{fi}, a corresponding set of atomic propositions Ap_{mi} over S_{mi} is *generated* such that:

$B_i : Ap_{mi} \dashrightarrow Ap_{fi}$ and,

$p_{mi} : S_{mi} \dashrightarrow Ap_{mi}$ is a *bijection.*

By the generation of Mpm-states S_{mi},

For every $s_{mi} \in S_{mi}$ there exists some $s_{fi} \in S_{fi}$ such that:

$s_{mi} = (s_{fi}, occ\#)$ where occ# is as defined in the proof of ΠM *Generator theorem*, in Theorem 2.8, in the generation of ΣM given $\{F_i, i=1..n\}$.

So, p_{mi} is generated such that:

$p_{mi}(s_{mi}) = p_{mi}(s_{fi}, occ\#) = (ap_{fi}, occ\#) = ap_{mi}$ where $ap_{fi} = p_{fi}(s_{fi})$,

Then,

$Ap_{mi} \dashrightarrow Ap_{fi}$ is a *mapping* derived from the mapping $S_{mi} \dashrightarrow S_{fi}$ and,

p_{mi} is a *bijection* that results by the same token as the *bijection* p_{fi}.

Note: B_i (and B) are *structures*, with a unique element denoting each entity's map from M_i to F_i as follows:

$B_{smi} : S_{mi} \dashrightarrow S_{fi}$ for states,

$B_{emi} : E_{mi} \dashrightarrow E_{fi}$ for events,

$B_{apmi} : Ap_{mi} \dashrightarrow Ap_{fi}$ for atomic propositions etc.

But for simplicity, we let B_i (and B) denote each of the above set of functions, decoded with reference to the context.

Example 3.2 A_{pm1} is the set of *atomic propositions* of M_1 *generated* such that:

$p_{mi}(a_0) = p_{mi}(a, 0) = (ap_a, 0) = ap_{a0}$,

$p_{mi}(b_0) = ap_{b0}$ etc.

From Fig. A and Fig. B of *Appendix* we see that,

$A_{pf1} = \{ap_a, ap_b, ap_c, ap_d\}$ which is *given* and,

$A_{pm1} = \{ap_{a0}, ap_{a1}, ap_{a2}, ap_{b0}, ap_{c0}, ap_{d0}\}$which is *generated.*

(The atomic propositions are not labelled in Fig. B)

$A_{pf2} = \{ap_p, ap_q, ap_s, ap_t, ap_u, ap_r, ap_s, ap_v, ap_r\}$ and,

$A_{pm2} = \{ap_{p0}, ap_{q0}, ap_{s0}, ap_{t0}, ap_{u0}, ap_{r0}, ap_{s0}, ap_{v1}, ap_{v0}, ap_{p1}, ap_{p2}\}$.

Example 3.3 Following is a sample of the typical examples of real-life systems, in general:

ΣAp_{fi} = {(buffer empty), (ready to receive), (ready to produce),...}

ΣAp_{mi} = {(buffer empty)$_0$, (buffer empty)$_1$, (ready to receive)$_0$, (ready to receive)$_1$, (ready to receive)$_2$,...}

The following extensions are defined :

Definition 3.2 $p_f : S_f \dashrightarrow Ap_f$ is a *bijection* such that:

$p_f(s_f) = \{(ap_{fi} = p_{fi}(s_{fi})), i=1..n\}, \forall s_f \in S_f$ of ΠF .

$p_m : S_m \dashrightarrow Ap_m$ is a *bijection* such that:

$p_m(s_m) = \{(p_{mi}(s_{mi}), i=1..n\}, \forall s_m \in S_m$ of ΠM .

$B : Ap_m \dashrightarrow Ap_f$ is a *surjection* such that:

$B(p_m(s_m)) = p_f(s_f)$.

Example 3.4 From Fig. 13 and Fig. 14 corresponding to ΠM and ΠF respectively,

$p_m(a_0, p_0, x_0) = \{ap_{a0}, ap_{p0}, ap_{x0}\}$,

$p_f(a, p, x) = \{ap_a, ap_p, ap_x\}$ and,

$B(\{ap_{a0}, ap_{p0}, ap_{x0}\}) = \{ap_a, ap_p, ap_x\}$.

Fig. 13 Partial product machine ΠM corresponding to Mpms of Fig.C

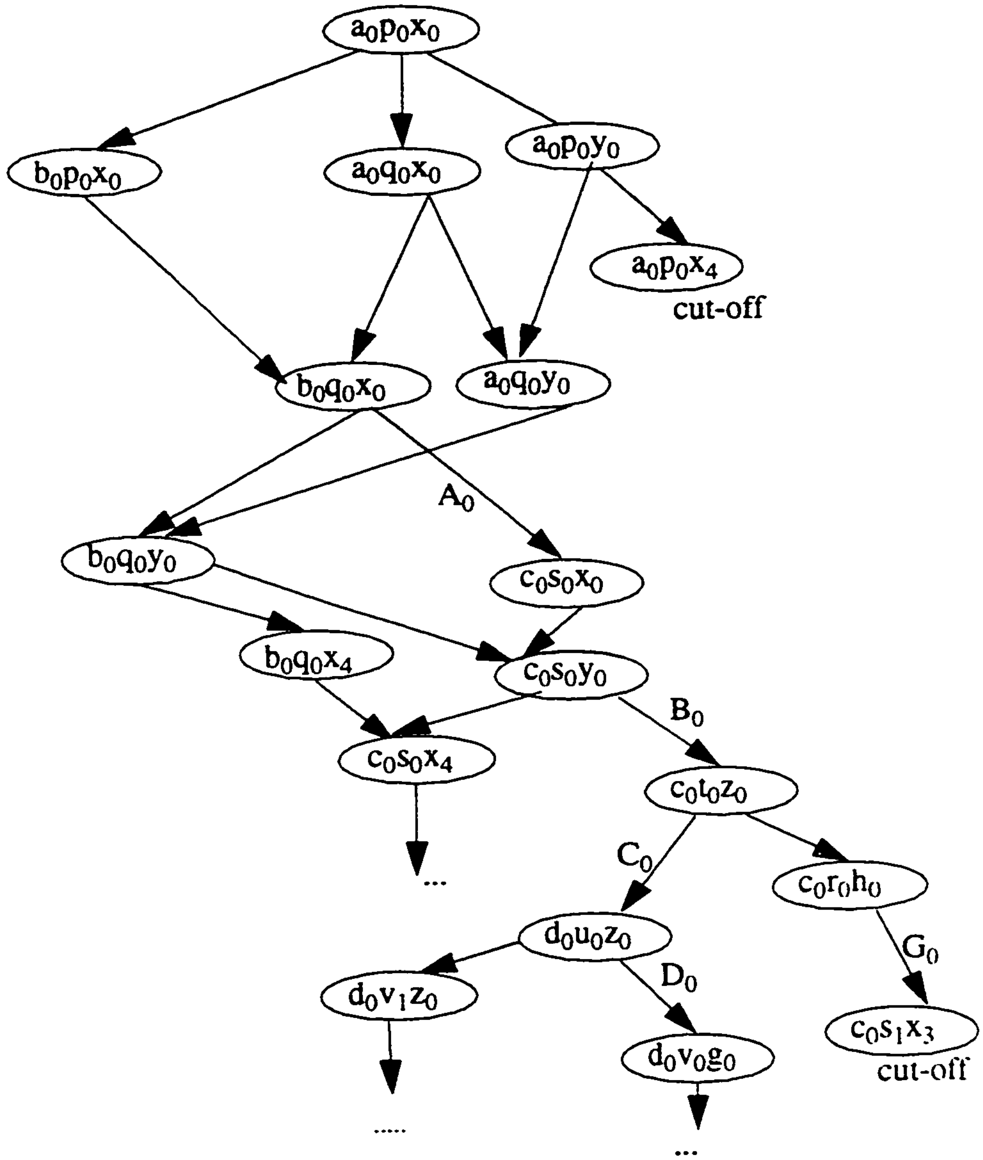
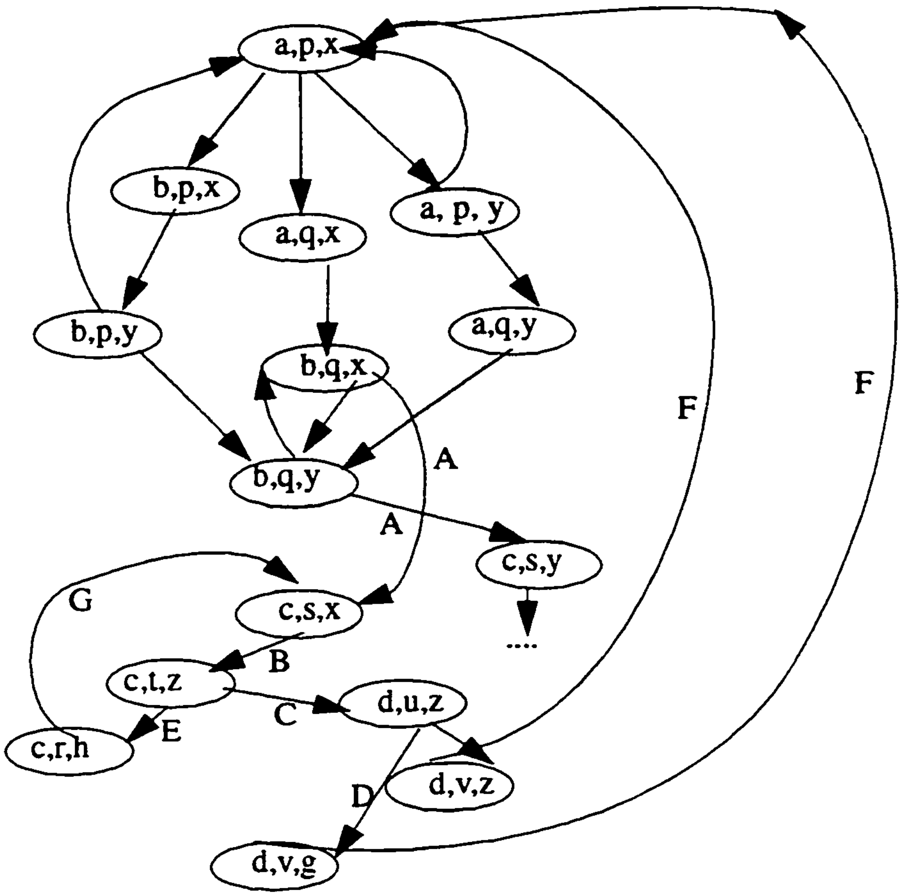


Fig. 14 Partial product machine ΠF corresponding to ΠM above



3.6.1 Propositional Operators of CML

3.6.1.1 Atomic Proposition & Satisfiability

$s_m \models_m ap_{mi}$ iff: $ap_{mi} = p_{mi}(s_{mi}) \in p_m(s_m)$, $i = 1..n$, where: $ap_{mi} \in Ap_{mi}$ and s_{mi} is the i^{th} component of s_m.

We say that, s_m *satisfies* the proposition $ap_{mi}, i=1..n$ in ΠM, denoted by: $\models_m$

Example 3.5

$s_{0m} = (a_0, p_0, x_0)$ is the initial-state of ΠM.

$s_{0m} \models_m ap_{a0}$ since $p_{m1}(s_{0m1}) = ap_{a0}$;

$s_{0m} \models_m ap_{p0}$, since $p_{m2}(s_{0m2}) = ap_{p0}$;

$s_{0m} \models_m ap_{x0}$, since $p_{m3}(s_{0m3}) = ap_{x0}$;

Similar definition can be made with states of S_f as well:

$s_f \models_f ap_{fi}$ iff $p_{fi}(s_{fi}) = ap_{fi}, i = 1..n$, $ap_{fi} \in Ap_{fi}$.

We say that, s_f *satisfies* the proposition ap_{fi} in ΠF, denoted by: $\models_f$.[1]

Example 3.6

$s_{0f} = (a_0, p_0, x_0)$ is the initial-state of ΠF.

$s_{0f} \models ap_{a0}$, from the fact $p_{f1}(s_{0f1}) = ap_{a0}$;

$s_{0f} \models ap_{p0}$, from $p_{f2}(s_{0f2}) = ap_{p0}$;

$s_{0f} \models ap_{x0}$, from $p_{f3}(s_{0f3}) = ap_{x0}$;

3.6.1.2 Conjunction of Propositions

$s_m \models (g \wedge h)$ iff:

$(s_m \models g) \wedge (s_m \models h)$ where g and h are propositions.

Example 3.7 $s_{0m} \models (ap_{a0} \wedge ap_{p0} \wedge ap_{x0})$.

Conjunction of atomic propositions, of all the n components of a state s_m viz., $\wedge_{i=1..n} p_{mi}(s_{mi})$ is called a *primitive conjunctive proposition*, satisfied by s_m.

$s_m \models \wedge_{i=1..n} p_{mi}(s_{mi})$

The above can be defined correspondingly in ΠF domain as well.

For instance,

$s_{0f} \models (ap_a \wedge ap_p \wedge ap_x)$ and in general,

$s_f \models \wedge_{i=1..n} p_{fi}(s_{fi})$

Note: It is to be noted that not all the components of the atomic propositions need to be added as a conjunct to be satisfied by s_m. n is just the upper limit.

3.6.1.3 Disjunction of propositions

$s_m \models (g \vee h)$ iff:

---

[1] Whenever there is no confusion, the satisfiability operators $\models_m$, $\models_f$ are denoted with the *subscripts viz.*, m or f dropped, and their domain understood from the context of usage.

$(s_m \models g) \vee (s_m \models h)$, where g and h are propositions.

Example 3.8

$s_{0m} \models (ap_{a0} \wedge ap_{p0} \wedge ap_{x0}) \vee (ap_{b0} \wedge ap_{q0} \wedge ap_{x0})$

The above can be defined correspondingly in ΠF domain as :

$s_{0f} \models (ap_a \wedge ap_p \wedge ap_x) \vee (ap_b \wedge ap_q \wedge ap_x)$.

We consider the disjunction of conjunctions (*disjunctive normal form*) as shown above, in the *model-checker* algorithm of Chapter-4, that checks one conjunction at a time.

3.6.1.4 Complement of a proposition

$s_m \models$ ^g iff

^($s_m \models g$) where g is a proposition, ^ is the *negation* operator.

Example 3.9

$s_{0m} \models$ ^$(ap_{b0} \wedge ap_{q0} \wedge ap_{x0})$;

The mapping of the above formula in ΠF domain is:

$s_{0f} \models$ ^$(ap_b \wedge ap_q \wedge ap_x)$.

3.6.1.5 Proposition with Implication

$s_m \models (g \Rightarrow h)$ iff:

$s_m \models$ (^g $\vee$ h) where g and h are propositions.

This proposition often is used along with modal and branching operators (to be defined) qualifying the proposition, h. When g is an atomic proposition or a conjunction of atomic propositions, it automatically defines the states to be satisfied, and so the satisfiability operator, $\models$ along with s_m can be skipped. Thus often, implication propositions are *universal* in the sense that they express satisfiability *over all the states* (as g either holds or not holds). When a proposition/formula is satisfied by all the states, it is called a *valid proposition*/formulae.

$\models (g \Rightarrow h)$ is a *valid formula*, satisfied by all the states of the system.

Example 3.10

$\models (ap_{d0} \Rightarrow ap_{u0})$.

This is the case since $d_0 = u_0$ due to simultaneity. In this case, due to symmetry, the converse of the implication is true as well or in other words, equivalence holds.

3.6.1.6 Propositions versus Predicates of CML

A CML *proposition* is formed by atomic propositions combined by boolean operators as defined above, viz., conjunction, disjunction, complementation and implication.

A proposition is often used with *modal* and *branching operators* (in *time* and *space*) qualifying and quantifying the proposition respectively, to define various CML *predicates*, which will be defined in the following section.

3.7 Modal and Branching Operators of Propositions

Within the framework of a *temporal logic*, the propositions as defined above can be enhanced by qualifying them with *modal operators* that are typical of a conventional temporal logic.

If the states of the *model of the logic* are simply formed as a *disjoint set of sequences*, there are no more operators needed other than the modal operators above to quantify the states of the model, and the resulting logic is called a *linear time logic*.

Alternatively, the states of the model could be formed into a *tree* with its *conflict points* representing the *(true) choices* made by the system sourcing the different *branches in time* that represent different *runs* of the system. Then, in addition to the modal operators, *universal* and *existential run operators* are incorporated to quantify the runs in which the specified states are reached. The associated logic is called a *branching time* logic.

In the model ΣM, which essentially (by generating ΠM) supports CML, the states are formed as a *disjoint set of trees*. The branches within each tree represent conflicts and so *branches in time*. In addition, the trees are tied at representative states called *synchronous output states* called *synchronization points*. These representatives source the different *branches in space* due to *strong concurrency/simultaneity*, subsequently causing multiple *interleavings* due to *possible concurrent* states reached asynchronous of each other, as explained in a few contexts before. This makes CML, a *branching space* as well as a *branching time* logic (*branching space-time logic*). This is how we get both the *run oper-*

ators corresponding to branches of time, and *interleaving operators* corresponding to branches of space.

3.8 Formal Definition of CML

3.8.1 CML Structures

Definition 3.3 The CML *structure* can be defined with respect to the *product machine* of CFsms as a triple: $CML_{\Pi F} := (S_f, p_f, R_f)$ and,

with respect to the *product machine* of CMpms as the triple: $CML_{\Pi M} := (S_m, p_m, R_m)$.

Both $CML_{\Pi F}$, $CML_{\Pi M}$ are referred to as *total-ordered structures* since R_f and R_m are respectively *total* among S_f and S_m.

Definition 3.4 The CML *structure* with respect to a *sum machine* ΣM can be defined as a triple: $CML_{\Sigma M} := (\Sigma S_{mi}, \Sigma p_{mi}, \leq)$, where $\leq$ is the *causal dependency-order* among Mpm-states. It is *partial-order structure* since $\leq$ is a PO among the Mpm-states S_{mi}. $CML_{\Sigma M}$ is a *propositional* logic over Mpm-states.

3.8.1.1 From Partial to Total Order Structure

Definition 3.5 The CML structure $CML^*_{\Sigma M} := (Fsv(Cset), p_m, \leq_{succ})$ is an extended structure of $CML_{\Sigma M}$ and is defined with respect to the *extended sum machine* $\Sigma^* M$ which is an enrichment of sum machine with configurations and final state vectors defined in Chapter-2.

To recall from Chapter-2,

$Cset \subseteq P(\Sigma S_{mi})$ where $P(\Sigma S_{mi})$ is the *powerset* of ΣS_{mi},

$Fsv: Cset \dashrightarrow S_{m1} \times S_{m2} \times \ldots \times S_{mn}$ and,

$\leq_{succ}$ refers to the *successor relation* among the *(final state) vectors* (extended from $\leq$ the causal order) of the set of all configurations, Cset :

. i.e., $C \subseteq C' \Leftrightarrow (Fsv(C)$ [1]$\leq_{succ} Fsv(C'))$

---

[1] When there is no confusion of this relation with the causality among Mpm-states, the subscript *succ* may be skipped.

Lemma 3.1 The CML structure with respect to the *extended sum machine*, $CML^*_{\Sigma M}$ and the CML structure with respect to the product machine $CML_{\Pi M}$ are *equivalent up to the reachability of their global-states*, as well as the structures $CML_{\Pi F}$ and $CML_{\Pi M}$ where,

B: ΠM --> ΠF.

Proof: The proof follows from the results proved in Chapter-2 as quoted below:

Equivalence of $CML_{\Pi M}$ and $CML^*_{\Sigma M}$:

Fsv(Cset) = S_m and R_m among S_m is same as $\leq_{succ}$ among Fsv(Cset) from the *Equivalence Theorem I (Theorem 2.5)* of Chapter-2.

The only difference between the two structures is that $\leq_{succ}$ among Fsv(Cset) is not explicit in ΣM unlike the relation R_m among S_m in ΠM.

By the above theorem, the *total-order structure* of $CML_{\Pi M}$ becomes an *extended partial-order structure* of $CML^*_{\Sigma M}$ and vice versa. By virtue of this equivalence, we derive the merits of both the structures at once. This is exploited in the *model-checker algorithm* of Chapter-4.

Equivalence of $CML_{\Pi M}$ and $CML_{\Pi F}$:

The structures (S_f, p_f, R_f) and (S_m, p_m, R_m) are equivalent by virtue of the mapping B, given that, B: ΠM --> ΠF according to the *Equivalence Theorem II* at Theorem 2.10 of Chapter-2.

■

The above *structures* of state-based logic CML define the *reachability* of the local and global vectors of Mpm-states over *interleavings* and *runs* through its formulae.

3.8.2 The *Modal* operators

Definition 3.6 CML defines the following modal operators:

X - 'next-state' operator

F- 'sometime in future' operator

G - 'always in future' operator

until - (binary) left proposition *is true until* the right proposition becomes true.

For instance,

$s_m \models X\, g_m$ means that: *next state* of s_m satisfies the proposition g_m in the product machine, ΠM.

Similarly, $s_f \models X\, g_f$ means that: *next state* of s_f satisfies the proposition g_f in ΠF.

The operators X, F, G and *until* are associated with the *future modality* in the sense that they imply propositions that will hold in the future, with respect to the states satisfying them. The corresponding *past* operators can be similarly defined with an underlined denotation of the corresponding operators.

For example, $s_f \models \underline{X}\ g_f$

means that: *predecessor state* of s_f (in the order R_f) satisfies the proposition g_f in ΠF.

and, $s_f \models (g_f \text{ until } h_f)$ means that g_f is true *until* the proposition h_f is true as well.

Similarly, *since* is the corresponding *past* operator of *until*, meaning that the left proposition of the operator is true ever since the right is true.

3.8.3 The Branching Operators -- Space & Time

In the last section above, the meaning of $s_f \models X\, g_f$ was defined as: *next state* of s_f, satisfying g_f. From the definition, it is not clear whether it refers to *some* next state or *all* next states of s_f that satisfy g_f. More specifically, whether it is a next state of *some interleaving* of *some run* or a next state of *all interleavings* of *all runs* or any of the other two combinations of *interleaving* and *run quantifiers*.

In order to aid the interpretation of the *modal operators* without any ambiguity and *thus the specification of the reachability of global-states deterministic*, all the above modal operators are *quantified* by the *branching operators* that are: *run operators* corresponding to *branching time*, and the *interleaving operators* corresponding to *branching space*, denoted as follows:

A_{rm} - Universal *run* operator/quantifier;

E_{rm} - Existential *run* operator;

A_{Irm} - Universal *interleaving* operator;

E_{Irm} - Existential *interleaving* operator/quantifier.

All the above operators are in ΠM domain. Corresponding operators in ΠF domain viz., A_{rf}, E_{rf}, A_{Irf}, E_{rf} can be defined as well[1]. The precise syntax of these operators will be defined in the following section.

Every run of CML formula refers to a *conflict-free product machine* Πr in ΠM domain and a *maximal configuration* C_{rmax} in ΣM domain. Every interleaving refers to a *path* of Πr in ΠM domain and a *succession of configurations* leading to C_{rmax} in ΣM domain (this is actually reflected in the *causal dependency* among the components of $Fsv(C_{rmax})$ due to the fact that the set C_{rmax} is *insensitive* to the order in which its members are reached).

3.8.4 Syntax of $CML_{\Pi M}$/ $CML_{\Pi F}$ Language

3.8.4.1 State, Interleaving (Path), Run Formulae

This section and the following section adopt the style of definitions from [5]:

Given below is the formal definition of the syntax of the language $CML_{\Pi M}/CML_{\Pi F}$. The language is an enrichment of CTL[1] and *F(B)*[2] and provides an expressive framework for specifying the properties of the input CFsm system. This framework is instrumental in the efficiency of model-checking algorithm in Chapter-4.

We start with the set of atomic propositions $\Sigma Ap_{mi}/\Sigma Ap_{fi}$ and *inductively* define a set of *primitive state formulae* and a set of *interleaving/path formulae*, *run formulae* as well as , *monadic, third-order state formulae* that are *non-primitive* and the *higher-order ones* of all the above categories mentioned due to inductive nesting.

Definition 3.7

- Each *atomic proposition* is a *(primitive) state formula*.
- If g, h are *state formulae* then so are $(g \wedge h)$, ^f. The conjunction of atomic propositions is called a *primitive conjunction.*
- If g is a *state formula*, then Fg and Xh are *interleaving/path formulae*.

---

[1] Whenever there is no confusion, the operators A_{rm}, A_{Irf} are denoted with the *subscripts viz., m or f* dropped, and their domain understood from the context of usage.

- If g is an *interleaving formula*, then $E_{Ir}g$ and $A_{Ir}g$ are *run formulae*.
- If g is a *run formula*, then $E_r g$ and $A_r g$ are *(non-primitive) state formulae*.
- If g, h are *state formulae* then (g *until* h) is an *interleaving formula*.
- If g, h are *state formulae*, (g *pos-wait-for* h), (g *must-wait-for* h), (g *pos-co-wait* h), (g *must-co-wait* h) are *run formulae*.
- If g, h are either *interleaving or run formulae*, then so are $(g \wedge h)$ and ^g, as the case may be.

The formulae with other boolean combinations with disjunctive operator etc., can be derived from the *basic* ones of conjunctions and negation such as: $(g \vee h)$ = ^(^g $\wedge$ ^h) and so on; The *modal* operator G is derivable as Gg = ^F^g.

The *past* operators corresponding to the *future* ones X, F, G and *until* are respectively $\underline{X}$, $\underline{F}$, $\underline{G}$ and *since*; these can be substituted in places of their corresponding *past* operators as well and similarly defined.

3.8.5 Syntax of $CML_{\Sigma M}$ / $CML^*_{\Sigma M}$ Language

$CML_{\Sigma M}$ is a simple *propositional logic* over which *first* and *higher-order formulae* are built in the extended model Σ^*M. $CML_{\Sigma M}$ consists of the set of atomic propositions ΣAp_{mi} each of which is an *Mpm-state formula*.

3.8.5.1 Global-state, Succession and Configuration Formulae

Given the *Mpm-state formulae* of $CML_{\Sigma M}$, we then *inductively* define a set of *global-state formulae* (corresponding to state formulae of $CML_{\Pi M}$) and a set of *succession* and *configuration formulae* (corresponding to *interleaving* and *run* formulae of $CML_{\Pi M}$ respectively) to build the *extension*, $CML^*_{\Sigma M}$. It can be easily seen that *global-state formulae, succession formulae and configuration formulae of $CML_{\Pi M}$, defined in the last sub-section correspond respectively to the state-formulae, interleaving/path formulae and run formulae of $CML^*_{\Sigma M}$.*

Definition 3.8

- Every *Mpm-state formula* is a *(primitive) global-state formula*.

- If g, h are *global-state formulae* , so are (g $\wedge$ h) and ^f. If g, h are *primitive*, so are (g $\wedge$ h) and ^f.
- If g is a *global-state formula*, Fg, Xg are *succession* (of configurations/ global-states) *formulae*. X and F could be replaced by the corresponding *past* operators $\underline{X}$, $\underline{F}$.
- If g is a *succession formula*, $E_{ICr}g$, $A_{ICr}g$ are *configuration formulae*.
- If g is a *configuration formula*, $E_{Cr}g$, $A_{Cr}g$ are (*non-primitive* i.e., *monadic, third-order*) *global-state formulae*.
- If g, h are *state formulae* then (g *until* h) is a *succession formula* and so is (g *since* h).
- If g, h are *global-state formulae*, (g *pos-wait-for* h), (g *must-wait-for* h), (g *pos-co-wait* h), (g *must-co-wait* h) are *configuration formulae*.

3.8.6 Models and Semantics of CML

CML *structures* define the *reachability* of states. CML models link the reachability with the *satisfiability* of formulae in these states.

3.8.6.1 Total and Partial Order Models

Definition 3.9 A *Total-Order model* is a *structure* TM := (S, L, R) such that: for all states s $\in$ S, and a *function* L from states to *formulae*(*primitive* and *non-primitive*), R is a *total binary relation* among states of S, and formula g is *satisfied* in TM denoted: <TM, s> |= p *iff* g $\in$ L(s) where |= refers to the *satisfiability relation*.

Definition 3.10 If R is *partial* in the above definition, then PM := (S, L, R) is referred to as a *Partial-Order model*, the rest of the definition remaining the same.

From the above definitions, we see that the structure $CML_{\Pi M}$:= (S_m, p_m, R_m) is a *Total-Order model* (and so is $CML_{\Pi F}$:= (S_f, p_f, R_f)) and the structure $CML_{\Sigma M}$:= (ΣS_{mi}, Σp_{mi}, $\leq$) is a *Partial-Order model*.

$CML^*_{\Sigma M}$ = (Fsv(Cset), p_m, $\leq_{succ}$), an *Extended Partial Order model* (EPM) (since it is extended from $CML_{\Sigma M}$) that is proved equivalent to the *Total-Order model* $CML_{\Pi M}$. We note that this idea is consistent with the mathematical fact that a *total-order* can be considered as an *extension* of a *partial-order* or the latter as a *restriction* of the former.

3.8.6.2 Semantics of $CML_{\Pi M}$, a Total-order Model

Given a *model* $CML_{\Pi M} := (S_m, p_m, R_m)$ or $CML_{\Pi F} := (S_f, p_f, R_f)$ we define the notion of *truth* in it through the relation |=. Given a *state* s_m , *an interleaving* I_r *and a run* Πr[1] , and correspondingly, a *state formula* g, *interleaving formula* g' and *run formula* g", we write:

$<CML_{\Pi M}, s_m> \models g$, $<CML_{\Pi M}, I_r> \models g'$ and $CML_{\Pi M}, \Pi r \models g''$

which means that g is *true at state* s_m, g' is *true of the interleaving* I_r, and g" is *true of the run* Πr.

|= is inductively defined as follows:

<u>**Definition 3.11**</u>

<u>TM.1: State Formula</u>

For a *(primitive) state formula* g that is an *atomic proposition*, $<CML_{\Pi M}, s_m> \models g$ iff $g \in p_m(s_m)$.

<u>TM.2:</u> If g, h are *state formulae*, $<CML_{\Pi M}, s_m> \models (g \wedge h)$ iff : $<CML_{\Pi M}, s_m> \models g$ and $<CML_{\Pi M}, s_m> \models h$; $<CML_{\Pi M}, s_m> \models$ ^g iff: ^($<CML_{\Pi M}, s_m> \models g$).

<u>TM.3: Interleaving Formula</u>

If g,h are *state formulae* and $I_r = (s_{1m}, s_{2m}, \ldots s_{km}, \ldots)$ is a *sequence of states* or an *interleaving/path* (of a run Πr), then

$<CML_{\Pi M}, I_r> \models Fg$ iff: for some s_{km} on I_r, $<CML_{\Pi M}, s_{km}> \models g$;

$<CML_{\Pi M}, I_r> \models Xg$ iff: $<CML_{\Pi M}, s_{2m}> \models g$.

Similar definition of the *past* operators $\underline{X}$, $\underline{F}$ can be made as well in a symmetric manner.

<u>TM.4: Run Formula</u>

If g, h are *interleaving formulae* and $\Pi r \subseteq \Pi M$ is a *run*, then,

---

[1] When we define formulae at state s_m, we let I_r and Πr denote interleavings and runs emanating from that state as opposed to the original denotations of the interleavings and runs from s_{0m}, the *initial state*.

$<CML_{\Pi M}, \Pi r> \models E_{Ir}\, g$ iff: *for some interleaving* I_r of Πr, $<CML_{\Pi M}, I_r> \models g$

$<CML_{\Pi M}, \Pi r> \models A_{Ir}\, g$ iff: *for all interleavings* I_r of Πr, $<CML_{\Pi M}, I_r> \models g$

TM.5: Monadic, Third-order, State Formula

If g, h are *run formulae* and $s_m \in S_m$ is a *state*, then,

$<CML_{\Pi M}, s_m> \models E_r\, g$ iff: *for some run* $\Pi r \subseteq \Pi M$, $<CML_{\Pi M}, \Pi r> \models g$

$<CML_{\Pi M}, s_m> \models A_r\, g$ iff: *for all runs* $\Pi r \subseteq \Pi M$, $<CML_{\Pi M}, \Pi r> \models g$

TM.6: *Until, Since* Operators [2]

If g, h are *state formulae*, then,

$<CML_{\Pi M}, I_r> \models$ (g *until* h) iff: for *some interleaving* $I_r = (s_{1m}, \ldots s_{km}, \ldots)$,

$<CML_{\Pi M}, s_{km}> \models$ [1]$(g \wedge h)$ and for all $1 \leq i \leq k$, $<CML_{\Pi M}, s_{im}> \models g$.

$<CML_{\Pi M}, I_r> \models$ (g *since* h) iff: for *some interleaving* $I_r = (s_{km}, s_{(k-1)m}, \ldots)$ such that:

I_r is a continuation of $I'_r := (s_{1m}, s_{2m}, \ldots s_{km})$ and,

$<CML_{\Pi M}, s_{1m}> \models (g \wedge h)$ and for all $1 \leq i \leq k$, $<CML_{\Pi M}, s_{im}> \models g$.

TM.7: *pos-wait-for* Operator

This operator is *tense free* and so, either *until* or *since* operator can imply it. It is non-specific of an *interleaving* consistent to its tense-free characteristic and so is a *run formula*. It is in contrast to a formula with *until/since* operator which is an *interleaving formula* by virtue of the specificity of *tense* in the operator.

If g, h are *state formulae* and $\Pi r \subseteq \Pi M$ is a *run*, then,

$<CML_{\Pi M}, \Pi r> \models$ (g *pos-wait-for* h) iff: *for some interleaving* I_r of Πr,

$<CML_{\Pi M}, I_r> \models$ ((g *until* h) $\vee$ (g *since* h)).

TM.8: *must-wait-for* Operator

If g, h are *state formulae* and $\Pi r \subseteq \Pi M$ is a *run*, then,

---

[1] We assume as in [2], that g continues to hold *until and inclusive* of the instant when h becomes true, as opposed to some conventions where the inclusive aspect is not guaranteed and hence the conjunction. Similar assumption for *the* operator *since* is made also.

$<CML_{\Pi M}, \Pi r> \models$ (g *must-wait-for* h) iff: *for all interleavings* I_r of Πr,

$<CML_{\Pi M}, I_r> \models$ ((g *until* h) $\vee$ (g *since* h)).

TM.9: *pos-co-wait* Operator

If g, h are *state formulae* and $\Pi r \subseteq \Pi M$ is a *run*, then,

$<CML_{\Pi M}, \Pi r> \models$ (g *pos-co-wait* h) iff:

$<CML_{\Pi M}, \Pi r> \models$ ((g *pos-wait-for* h) $\vee$ (h *pos-wait-for* g)).

TM.10: *must-co-wait* Operator

If g, h are *state formulae* and $\Pi r \subseteq \Pi M$ is a *run*, then,

$<CML_{\Pi M}, \Pi r> \models$ (g *must-co-wait* h) iff:

$<CML_{\Pi M}, \Pi r> \models$ (g *must-wait-for* h) $\vee$ (h *must-wait-for* g).

TM.11: If g , h are *interleaving formulae*, $<CML_{\Pi M}, I_r> \models (g \wedge h)$ iff :

$<CML_{\Pi M}, I_r> \models g \wedge <CML_{\Pi M}, I_r> \models h$ and,

$<CML_{\Pi M}, I_r> \models$ ^g iff: ^($<CML_{\Pi M}, I_r> \models g$)

TM.12: If g and h are *run formulae*, $<CML_{\Pi M}, \Pi r> \models (g \wedge h)$ iff :

$<CML_{\Pi M}, \Pi r> \models g \wedge <CML_{\Pi M}, \Pi r> \models h$ and,

$<CML_{\Pi M}, \Pi r> \models$ ^g iff: ^($<CML_{\Pi M}, \Pi r> \models g$)

3.8.6.3 Semantics of $CML^*_{\Sigma M}$, the Extended Partial-order Model

Given the model $CML_{\Sigma M} := (\Sigma S_{mi}, \Sigma p_{mi}, \leq)$, we define the notion of *truth* in the extended model $CML^*_{\Sigma M} = (Fsv(Cset), p_m, \leq_{succ})$ through the relation $\models$.

Given an *Mpm-state* s_{mi}, a *global-state* s_m, a *succession* of (configurations) I_{C_r} and a *configuration* C_r , as well as an *Mpm-state formula*, a *global-state formula* g_i, a *succession formula* g' and a *configuration formula* g'', we write:

$<CML_{\Sigma M}, s_{mi}> \models g_i$, $<CML^*_{\Sigma M}, s_m> \models g_i$, $<CML_{\Pi M}, I_{C_r}> \models g'$ and

$<CML_{\Pi M}, C_r> \models g''$ which mean that :

g_i is *true at Mpm-state* s_{mi}, g_i is also *true at global-state* s_m, and g' is *true for the succession* I_{Cr} and lastly, g'' is *true for the configuration* C_r, where C_r is the *maximal configuration* corresponding to run Πr.

Satisfiability (|=) over $CML^*_{\Sigma M}$ is inductively defined as follows:

Definition 3.12

EPM.1: Global-state Formula

For an *Mpm-state-formula* that is an *atomic proposition* g_i, $<CML_{\Sigma M}, s_{mi}> \models g_i$ iff $g_i = p_{mi}(s_{mi})$ which is also a *global-state formula* in $CML^*_{\Sigma M}$ as follows:

For a *(primitive) global-state formula* that is an *atomic-proposition*, $<CML^*_{\Sigma M}, s_m> \models g_i$ iff $g_i \in p_m(s_m)$.

EPM.2: If g, h are *global-state formulae*, $<CML^*_{\Sigma M}, s_m> \models (g \wedge h)$ iff : $<CML^*_{\Sigma M}, s_m> \models g$ *and* $<CML^*_{\Sigma M}, s_m> \models h$; $<CML^*_{\Sigma M}, s_m> \models$ ^g iff: ^($<CML^*_{\Sigma M}, s_m> \models g$).

EPM.3: Succession Formula

If g is a *global-state formula*, $C_r \subseteq$ Cset is a *configuration* (corresponding to a *run* $\Pi r \subseteq \Pi M$), and $I_{Cr} = (C^1_r, C^2_r, \ldots, C^k_r, \ldots)$ is a *succession of configurations*, all contained in C_r (corresponding to the interleaving or [1]*succession of Fsvs* $I_r = (Fsv(C^1_r), Fsv(C^2_r), \ldots, Fsv(C^k_r), \ldots) = (s^1_{mr}, s^2_{mr}, \ldots, s^k_{mr}, \ldots)$, $s^i_{mr} \in S_{mr}$ of the run Πr), then,

$<CML^*_{\Sigma M}, I_{Cr}> \models Xg$ iff $<CML^*_{\Sigma M}, Fsv(C^2_r)> \models g$ and,

$<CML^*_{\Sigma M}, I_{Cr}> \models Fg$ iff $<CML^*_{\Sigma M}, Fsv(C^k_r)> \models g$.

The *succession formulae* defined above are *modal predicates* , to be quantified by *succession quantifiers*, in the next definition.

f Similarly, *succession formulae* with *past modalities* viz., $\underline{X}$, $\underline{F}$ can be defined.

EPM.4: Configuration Formula

If g is an *interleaving formula*, C_r is a *configuration* and I_{Cr} is a *succession*,

$<CML^*_{\Sigma M}, C_r> \models E_{ICr}\, g$ iff: *for some succession* I_{Cr} of Πr, $<CML^*_{\Sigma M}, I_{Cr}> \models g$

[1] In $CML^*_{\Sigma M}$, *succession of configurations* and that of their Fsvs can be interchangeably used where there is no confusion even though only the latter is identical to an *interleaving* I_r of $CML_{\Pi M}$.

$<CML^*{}_{\Sigma M}, C_r> \models A_{ICr}\, g$ iff: *for all successions* I_{Cr} of Πr, $<CML^*{}_{\Sigma M}, I_{Cr}> \models g$

EPM 5: Monadic, Third-order Global-state Formula:

If Fsv(C) is a *global-state* of Σ^*M and g is a *configuration formula*, then,

$<CML^*{}_{\Sigma M}, Fsv(C)> \models E_{Cr}g$ iff: *for some continuation* C_r of C such that: $C \subseteq C_r$ (corresponding to a run $\Pi r \subseteq \Pi M$), $<CML^*{}_{\Sigma M}, C_r> \models g$.

$<CML^*{}_{\Sigma M}, Fsv(C)> \models A_{Cr}g$ iff: *for all continuations* C_r of C such that: $C \subseteq C_r$ (corresponding to a run $\Pi r \subseteq \Pi M$) $<CML^*{}_{\Sigma M}, C_r> \models g$.

Often, $C = C_0$, the initial configuration where $s_{0m} = Fsv(C_0)$ in the above.

EPM 6: *Until* Operator

$<CML^*{}_{\Sigma M}, I_{Cr}> \models (g$ *until* $h)$ iff: for some *global-state formulae* g, h and for *some succession* $I_{Cr} = (C^1_r, C^2_r, ..., C^k_r)$,

$<CML^*{}_{\Sigma M}, Fsv(C^k_r)> \models (g \wedge h)$ and for all $1 \leq i \leq k$, $<CML^*{}_{\Sigma M}, Fsv(C^i_r)> \models g$.

The definition of satisfiability of g *since* h, can be made likewise.

EPM 7: *Pos-wait-for* Operator

If $C_r \subseteq$ Cset is a configuration (corresponding to a run $\Pi r \subseteq \Pi M$) and g, h are *primitive global-state formulae*, then ,

$<CML^*{}_{\Sigma M}, C_r> \models (g$ *pos-wait-for* $h)$ iff: *for some succession* of configurations $(C^1_r, C^2_r, ..., C^k_r)$ all contained in C_r, $<CML^*{}_{\Sigma M}, Fsv(C^k_r)> \models (g \wedge h)$ and for all $1 \leq i \leq k$, $<CML^*{}_{\Sigma M}, Fsv(C^i_r)> \models g$.

'*for some succession of configurations*' is equivalent to '*for some interleaving*' but the former is not explicit because, a configuration which is a set of Mpm-states, is *interleaving independent* as shown in Chapter-2.

EPM 8: *must-wait-for* Operator

If $C_r \subseteq$ Cset is a configuration (corresponding to a run $\Pi r \subseteq \Pi M$) and g, h are *primitive global-state formulae*, then, $<CML^*{}_{\Sigma M}, C_r> \models (g$ *must-wait-for*[1] $h)$ iff: *for all successions* of configurations $(C^1_r, C^2_r, ..., C^k_r)$ that are contained in C_r, $<CML^*{}_{\Sigma M}, Fsv(C^k_r)> \models h)$ and for all $1 \leq i \leq k$, $<CML^*{}_{\Sigma M}, Fsv(C^i_r)> \models g$.

EPM 9: *Pos-co-wait* Operator

If $C_r \subseteq Cset$ is a configuration (corresponding to a run $\Pi r \subseteq \Pi M$) and g, h are *primitive global-state formulae*, then,

$<CML^{*}_{\Sigma M}, C_r> \models$ (g *pos-co-wait* h) iff:

$<CML^{*}_{\Sigma M}, C_r> \models$ ((g *pos-wait-for* h) $\vee$ (h *pos-wait-for* g))

Similarly, *must-co-wait* can be derived as well.

EPM 10: If g , h are *configuration formulae*, $<CML^{*}_{\Sigma M}, C_r> \models (g \wedge h)$ iff:

$<CML^{*}_{\Sigma M}, C_r> \models g \wedge <CML^{*}_{\Sigma M}, C_r> \models h$.

$<CML^{*}_{\Sigma M}, C_r> \models$ ^g iff: ^($<CML^{*}_{\Sigma M}, C_r> \models g$).

It is easily seen from the above TM and EPM definitions that, *configuration formulae* of $CML^{*}_{\Sigma M}$ are equivalent to *run formulae* of $CML_{\Pi M}$ and so are the *succession formulae* of the former to *interleaving formulae* of the latter as well as the *global-state formulae* of the former to *state formulae* of the latter and also there is one-to-one correspondence between other definitions viz., *until, wait-for* and *co-wait* though a couple of them are skipped in EPM; they can be similarly derived as mentioned.

3.8.7 Equivalence of the Models $CML_{\Pi F}$, $CML_{\Pi M}$ and $CML^{*}_{\Sigma M}$

Lemma 3.1 established the equivalence of the three versions of CML *structures*, up to the *reachability of the global-states* in their respective structures. A *model* is an extension of a *structure* with an additional constraint of *satisfiability of the formulae* in the states reachable in the latter. The following theorem establishes the *equivalence* of the models *up to the satisfiability of the formulae* of their respective states.

[1] *wait-for* operator can also be expressed in terms of both *until* and *since* (independent of the *tense*) in the model EPM just as in TM and vice versa i.e., TM could have similarly defined *wait-for* from first principles just as in EPM.

3.8.7.1 Equivalence Theorem III

<u>**Theorem 3.1**</u> The models $CML_{\Pi F}$, $CML_{\Pi M}$, $CML^*_{\Sigma M}$ are *equivalent* up to the *satisfiability* of *monadic, third-order* state formulae:

(i) $<CML_{\Pi F}, s_f> \models g_{\Pi F} <=>$

(ii) $<CML_{\Pi M}, s_m> \models g_{\Pi M} <=>$

(iii) $< CML^*_{\Sigma M}, Fsv(C_r)> \models g_{\Sigma^* M}$ where:

$s_m = Fsv(C_r)$, $B(s_m) = s_f$, $B: \Pi M \rightarrow \Pi F$.

<u>**Proof:**</u>

<u>*Equivalence of (i) and (ii):*</u> For every *state*, *run* and *interleaving* of ΠM, the corresponding entities of ΠF can be mapped *onto*, by the *surjective function* B such that, for every formula over a given entity of ΠM, there is one over the corresponding mapped entity in ΠF.

<u>*Equivalence of (ii) and (iii):*</u> This follows from the equivalence of the models $CML_{\Pi M}$ and $CML^*_{\Sigma M}$:

From the definitions of a *run* and *interleaving* of ΠM (from Chapter-2), for every run Πr and interleaving I_r in ΠM, there exists a corresponding *maximal configuration* $\Sigma r = C_{rmax}$ (C_{rmax} is also referred to as C_r) and a *succession of configurations* I_{Cr} in Σ^*M such that, the final state vectors of configurations of I_{Cr} form a sequence *identical* to a sequence of states of some interleaving I_r in Σ^*M. Similarly, for every state s_m of ΠM, there is a vector $Fsv(C_r)$ in Σ^*M.

Therefore, every formula over any entity of ΠM has a matching formula over the corresponding entity of Σ^*M and hence (ii) and (iii) are equivalent.

The equivalence of (i) and (iii) follows transitively from those of (i) and (ii) and (ii) and (iii).

■

3.8.8 Satisfiability of CML formulae and Global-state Reachability

From the above sections, it is clear that when a CML formula (in any of the three equivalent versions) is satisfied at a *global-state*, that particular state is *reachable* in *some* or *all interleavings* of *some* or *all runs* as quantified in the formula.

3.8.8.1 Primitive Conjunctive Propositions and Global-states

A *primitive conjunctive proposition* is a conjunction of *atomic propositions* as defined. By the bijective definition between *atomic propositions* and *Mpm-states* and the definition of the *conjunctive propositions* in Section 3.6.1.2, it can be seen that the *reachability of a global-state* corresponds to *satisfiability of a primitive conjunctive proposition* in that state. Reachability of that global-state in *some or all runs* and *interleavings* is quantified by the *run* and *interleaving operators* which quantify that proposition.

3.8.8.2 Polynomial Versus Exponential size of Propositions

Satisfiability of a *primitive conjunction* of n conjuncts/atomic propositions in ΠF domain corresponds to reachability of a *unique* global-state by possibly multiple runs and interleavings. A formula in the *disjunctive normal form* therefore corresponds to checking the reachability of as many global-states as there are disjuncts in it. Thus the size of the formula is a polynomial in the number of disjuncts.

Non-determinism in the Conjunctive Normal Form:

On the other hand, a CML formula in *conjunctive normal form* in the following form is a conjunction of *n propositions*, where each conjunct is a disjunction of up to *m local atomic propositions*. *It is possible that every local disjunct can be in conjunction with any of the m disjuncts of each of other (n-1) conjuncts, due to asynchrony/concurrency among the n Mpms.* Therefore, the actual size of the input formula is exponential, with (m ** n) *primitive conjunctions* which amounts to checking the *reachability* of that many global states.

The disjunction of m different local atomic propositions as a conjunct is perceived and characterized here as the *non-determinism* in the *property/formula* to be verified.

3.8.8.3 Formulae in ΠF domain and Cut-off

It is recalled that, any formula in ΠM domain can be *mapped onto* that in ΠF domain as in the definition of X operator below:

$<CML_{\Pi F}, s_f> \models A_r\, E_{Ir}\, X\, g$ iff :

$\forall \Pi r_f \subseteq \Pi F, \exists \Pi I_{rf} \subseteq \Pi r_f, \exists\, s_f, s'_f \in S_{Irf} : (s_f\, R_{Irf}\, s'_f)$ where:

$s'_f \models g$, B: $\Pi r \dashrightarrow \Pi r_f$, and B : $\Pi I_r \dashrightarrow \Pi I_{rf}$

For example,

$<CML_{\Pi F}, (a, p, x)> \models A_r\, E_{Ir}\, X\, (ap_q)$.

If there are two global-states of ΠM, both mapping onto the same global-state of ΠF, whatever formula is satisfied by one will be satisfied by the other when projected onto the ΠF plane. This is precisely the reason why one must be the *cut-off state* of the other, unless they are in two conflicting paths.

The *cut-off states* of ΠM represent the *recurrence* of ΠF. Though truncated after cut-off states, ΠM essentially remains *infinite*. The *infinite growth* of ΠM corresponds to the infinite number of cycles of the *finite system* ΠF.

3.8.8.4 Revisiting the Role of Interleaving operator

It is noted that in a given interleaving of a specific run, there is a *unique* successor for every (global) state. So, the meaning of the operator X as '*next state*' can be interpreted unambiguously. Without the interleaving operator and with just the run operator there is an implicit interpretation of '*some successor*' added on to the operator X.

For example, it can be seen from Fig. 13,

$<CML_{\Pi M}, (a_0, p_0, x_0)> \models A_r\, E_{Ir}\, X\, (ap_{q0})$.

Without the existential interleaving operator E_{Ir}, q_0 is the *next state*, only if M_2 is executed before M_1 or M_3. In case E_{Ir} is not added, X has to be interpreted as *some or possible next state* of (a_0, p_0, x_0). In other words, in the absence of E_{Ir}, the reachability of the next state conveyed explicitly by it has to be implicitly conveyed by X alone; and this causes ambiguity or *non-deterministic interpretation* of the formula.

Interleaving and Degree of Certainty/Finite and Infinite Modality:

Information from a different perspective will be conveyed by the interleaving operator in the interpretation of the operator F. The interleaving operator associated with the modality of F throws more light on the *degree of certainty* of the qualified proposition as follows: The *existential operator* E_{Ir} indicates only the *possibility* of the proposition i.e., to mean it could occur at a possibly *infinite future* after indefinitely many number of cycles of the system. While the *universal interleaving* operator A_{Ir} indicates that the predicate *must hold* in the *definite, near future (necessity)*, irrespective of how the component machines are interleaved.

The long wait in the case of E_{Ir} (*possible* interleaving) comes from having to wait for the *possible chance of a specific execution ordering* of Mpms, depending on the predicate, which could for ever be eluded. The *necessity* condition in the case of A_{Ir} implicitly imposes the reachability of the target global-state within the first cycle of the system itself. For, if it can wait till the second cycle, it can as well be waiting through multiple cycles. This rules out the *indefinitely long wait* through multiple cycles of the system.

Thus, *possibility* is associated with an *infinite future* and *necessity* is associated with a *finite* one.

Example 3.11 For example, referring to Fig. 13 and Fig. 14 again,

[1] $B(s_{0m}) \models E_r\, E_{Ir}\, F\, (B(ap_{a0} \wedge ap_{q0} \wedge ap_{x0})) \Leftrightarrow$

$s_{0f} \models E_r\, E_{Ir}\, F\, (ap_a \wedge ap_q \wedge ap_x)$.

Considering the formula in the ΠF domain, it states that *there exists a run* which has *some interleaving* in which, a certain future state of the initial state s_{0f} satisfies the conjunction in the above formula. The conjunction $(ap_a \wedge ap_q \wedge ap_x)$ depends on the order in which the component Fsms are executed. It is possible that in every cycle of the system, (after the reset back to (a, p, x)) the global state (a, q, x) corresponding to the conjunction $(ap_a \wedge ap_q \wedge ap_x)$ is missed if F_1 or F_3 is always executed before F_2, leaving only the *theoretical possibility* that the conjunction can occur at some *future*.

---

[1] When clear from the context, the model denotation can be omitted from the left side of the satisfiability operator, $\models$.

The above fact is captured or depicted by the interleaving operator E_{Ir} qualifying the conjunction. Without this operator, the interpretation is ambiguous as to whether the conjunction is definite in the future or just a possibility. Accordingly, the ***model-checker algorithm*** to be discussed in Chapter-4 can be designed ***deterministically***.

Example 3.12

$(c_0, s_0, x_0) \models E_r\ A_{Ir}\ (ap_{c0}$ *until* $ap_{t0})$ where:

$ap_{c0} = p_{m1}(c_0)$, $ap_{t0} = p_{m2}(t_0)$.

The boundary condition is that both the Mpm-states corresponding to the atomic propositions enter *simultaneously*.

For example,

$(c_0, s_0, x_0) \models E_r\ A_{Ir}\ (ap_{c0}$ *until* $ap_{s0})$.

The operator *since*, gives a *retrospective* perception, as opposed to the *futuristic* view of *until* operator.

Example 3.13 From Fig. 13 and Fig. 14 ,

$<CML_{\Pi M}, (d_0, v_1, z_0)> \models E_r\ A_{Ir}\ (ap_{z0}$ *since* $ap_{u0}) <=>$

$<CML_{\Pi F}, (d, v, z)> \models E_r\ A_{Ir}\ (ap_z$ *since* $ap_u)$.

The above means that ever since ap_{u0} is true, ap_{z0} is true.

From Fig. 14, we can verify that (d, u, z) is in the *past* of (d, v, z). It is easy to see that (ap_z $\wedge$ ap_u) is satisfied at (d, u, z) which is the *past* (in particular, a predecessor) of (d, v, z) where ap_z continues to be true.

3.8.9 Non-monadic CML Formulae

The recursive definition of the language CML as defined in Definition 3.7 and Definition 3.8 does not restrict the *operand* of the *first-order modal* operator to be a *primitive state-formula* or a *proposition* but includes in general a *higher order* state formula by the inductive nesting of *third-order state formulae*.

Example 3.14 From Fig. 13,

$(a_0, p_0, x_0) \models A_r\ A_{Ir}\ F\ (E_r\ A_{Ir}\ (ap_{c0}$ *until* $ap_{t0}))$.

This is an example of *non-monadic formula* in which the operand of the modal operator F is not a *proposition*, i.e., a *primitive state formula* but is another third-order state-formula.

The above means that, for *all interleavings* (A_{Ir}) of *all runs* (A_r), there is an *intermediate state* in the future (F) of the state (a_0, p_0, x_0), that satisfies (E_r A_{Ir} (ap_{c0} *until* ap_{t0}).

The above mentioned intermediate future state satisfies ap_{c0}, from which there is *a run*, (E_r) all of whose interleavings (A_{Ir}) continue to satisfy ap_{c0} *until* ap_{t0} holds as well. The state (c_0, s_0, x_0) is one such intermediate state of (a_0, p_0, x_0) satisfying the inner *monadic, third-order state formula* as below:

(c_0, s_0, x_0) |= E_r A_{Ir} (ap_{c0} *until* ap_{t0}).

(c_0, s_0, y_0), (c_0, s_0, x_4) are other possible intermediate states. Theoretically, any number of levels of nesting is possible. Handling them in practice is beyond the scope of this work.

Example 3.15 From Fig.B/C of *Appendix* for ΣM and Fig. 13 for ΠM.

(a_0, p_0, x_0) |= E_r ($p_{m1}(a_0)$ *pos-wait-for*[1] $p_{m2}(q_0)$).

Stated differently, the above formula is:

<$CML_{\Pi M}$ > |= $p_{m1}(a_0)$ => E_r E_{Ir} ($p_{m1}(a_0)$ *until* $p_{m2}(q_0)$)

The above is a *validity formula*, since its specification is independent of states.

It becomes a *valid formula*, if it is *true* or *satisfied*.

<$CML_{\Pi M}$, (a_0, p_0, x_0)> |= E_r E_{Ir} ($p_{m1}(a_0)$ *until* $p_{m2}(q_0)$)

Also,

<$CML_{\Pi M}$, (d_0, u_0, z_0)> |= E_r ($p_{m3}(z_0)$ *pos-wait-for* $p_{m2}(t_0)$) and,

<$CML_{\Pi M}$> |= $p_{m3}(z_0)$ => E_r E_{Ir} ($p_{m3}(z_0)$ *since* $p_{m2}(t_0)$)

Example 3.16

<$CML_{\Pi M}$, (b_0, p_0, x_0)> |= E_r ($p_{m1}(b_0)$ ***must-wait-for*** $p_{m2}(q_0)$) and,

<$CML_{\Pi M}$, (b_0, p_0, x_0)> |= E_r A_{Ir} *(*$p_{m1}(b_0)$ *until* $p_{m2}(q_0)$ *)*

Example 3.17 Considering Fig. 13 again,

<$CML_{\Pi M}$, (a_0, p_0, x_0)> |= E_r ($p_{m1}(b_0)$ *pos-co-wait* $p_{m2}(q_0)$)

---

[1] Even though the related propositions and hence the states may be causally independent, due to interleaving of execution, they may still wait for each other in the global time scale.

<=>

$CML_{\Pi M}, (a_0, p_0, x_0)> \models E_r E_{Ir} ((p_{m1}(b_0)$ *until* $p_{m2}(q_0)) \vee ((p_{m2}(q_0)$ *until* $p_{m1}(b_0)))$

<=>

$<CML_{\Pi M}> \models (p_{m1}(b_0) \Rightarrow E_r E_{Ir} (p_{m1}(b_0)$ *until* $p_{m2}(q_0))$

$\vee \quad (p_{m2}(q_0) \Rightarrow E_r E_{Ir} (p_{m2}(q_0)$ *until* $p_{m1}(b_0))$

Since b_0 and q_0 of M_1 and M_2 can respectively be entered independent of each other, either of them can be entered first, and possibly continue to hold till the other is entered.

Example 3.18 $(a_0, p_0, x_0) \models E_r (p_{m1}(b_0)$ *must-co-wait* $p_{m2}(q_0))$ <=>

$<CML_{\Pi M}> \models (p_{m1}(b_0) \Rightarrow E_r A_{Ir} (p_{m1}(b_0)$ *until* $p_{m2}(q_0))$

$\vee \quad (p_{m2}(q_0) \Rightarrow E_r A_{Ir} (p_{m2}(q_0)$ *until* $p_{m1}(b_0))$

It is easy to see that the *concurrency* relation *co* among the Mpm-states *is equivalent to pos-co-wait* modality among their respective (atomic)propositions; similarly, the *strong concurrency* relation *sync* among *Mpm-states is equivalent to must-co-wait* modality among their respective propositions.

In the above example,

$(p_{mi}(a_0)$ *pos-co-wait* $p_{mi}(q_0))$ <=> $(a_0$ *co* $q_0)$.

$(p_{mi}(b_0)$ *must-co-wait* $p_{mi}(q_0)$ <=> $(b_0\ sync_{in}\ q_0)$ and

$(p_{mi}(c_0)$ *must-co-wait* $p_{mi}(s_0)$ <=> $(c_0\ sync_{out}\ s_0)$.

But there are more complex (non-atomic) propositions that *co-wait* on each other and correspondingly, the relationship among the associated states also is more complex though there is underlying synchronization involved directly or indirectly relating them.

3.9 CML with respect to ΣM

$CML_{\Pi F}$ *structure* is based on the product machine of the input CFsms. The function B (from ΠM *onto* ΠF) acts as a window to view the states of ΠF through those of ΠM, which in turn is *virtual*, and generated dynamically from ΣM, the *sum machine* which is *static* and *real* (as opposed to *virtual*).

Specifically, the states of ΠM are generated as the Final state vectors of the *configurations* C of the *extended sum machine* $\Sigma^{*}M$ as explained and proved in Chapter-2. The essential characteristics of ΣM are:

Simultaneity due to *synchrony* is the origin of concurrency and the basis of Minimal prefixes. Minimal prefixes are the basis of the following:

(i) Visiting local Mpm-states corresponding to local branches of time, without losing globality. This means, *branching in space* from one Mpm to the other without losing track of the continuity in time i.e., the current *branch of time*, is possible.

(ii) Deducing the reachability of a global-state in *all interleavings from one*.

The following set of axioms are consolidation of the above notions, and link the properties of Mpm-states and *final state vectors* of *configurations* of ΣM with those of the states of ΠM.

3.9.1 Axioms of CML

Following axioms assume a configuration C of ΣM, global-state s_m of ΠM, with an initial configuration C_0 corresponding to its Fsv/initial global-state s_{0m}.

Axiom 3.1

$\mathrm{Fsv}(C) = (s_{m1}, s_{m2}, \ldots, s_{mn})$ <=>

$<{}^{1}CML^{*}{}_{\Sigma M}, \mathrm{Fsv}(C)> \models \wedge_{i=1..n} (p_{mi}(s_{mi}))$ <=>

$(s_{mi}\ co\ s_{mj}),\ \forall i, j=1..n, i<>j$.

This follows since every state of ΠM is virtually generated as a Final state vector of a configuration of $\Sigma^{*}M$. This axiom links a *global state-vector* with *concurrency* among Mpm-states with *conjunction* of their respective *atomic propositions*.

Axiom 3.2

$(s_{m1}\ co\ s_{m2}\ co \ldots co\ s_{mn})$ <=>

$\exists C: \mathrm{Fsv}(C) = s_m: s_m \models \wedge_{i=1..n}(p_{mi}(s_{mi}))$ <=>

$\mathrm{Fsv}(C) \models E_{Cr}\ (\wedge_{i,j=1..n}(p_{mi}(s_{mi})\ \textit{pos-co-wait}\ p_{mj}(s_{mj}))),\ i<>j$ <=>

[1] The denotation of this model may be skipped from the left side of the operator $\models$ when it is clear from the context of usage. Also, it can be replaced with CML $_{\Pi M}$ and *vice versa* since the models are equivalent.

$<CML^*_{\Sigma M}, Fsv(C)> \models E_{Cr}\, I_{Cr}\, F(\wedge_{i=1..n}(p_{mi}(s_{mi}))) <=>$

$<CML_{\Pi M}, s_{0m}> \models E_r\, E_{Ir}\, F(\wedge_{i=1..n}(p_{mi}(s_{mi})))$

Follows by the definition of *pos-co-wait* operator in TM.9 and EPM .9 from definitions at Definition 3.11 and Definition 3.12 respectively. Equivalence of the formulae follows from the equivalence of the models TM and EPM.

Similarly,

$<CML_{\Pi M}, s_{0m}> \models E_r\, (\wedge_{i,j=1..n}(p_{mi}(s_{mi})\ \textit{must-co-wait}\ p_{mj}(s_{mj}))),\ _{i<>j} <=>$

$s_{0m} \models E_r\, A_{Ir}\, F(\wedge_{i=1..n}(p_{mi}(s_{mi})))$

The above follows from the definition of TM.10 .

Axiom 3.3

$(s_{mi}\ \textit{sync}\ s_{mj}) <=>$

$Fsv(C_0) \models E_{Cr}\, (p_{mi}(s_{mi})\ \textit{must-co-wait}\ p_{mj}(s_{mj})) <=>$

$s_{0m} \models E_r\, ((p_{mi}(s_{mi})\ \textit{must-co-wait}\ p_{mj}(s_{mj})) <=>$

$s_{0m} \models E_r\, A_{Ir}\, F\, ((p_{mi}(s_{mi}) \wedge p_{mj}(s_{mj}))$

It is noted that the Mpm-states are related by their entry order. One of the *synchronous input* states may be entered before the other and the one entered first must wait for the second one to be entered, in order to synchronize on the common event. In the case of synchronous output states, they are entered *simultaneously*, which is a stronger condition, and so they also obey the following axiom.

Axiom 3.4

$(s_{mi}\ \textit{sync}_{out}\ s_{mj}) <=>$

$s_{0m} \models E_r\, A_{Ir}\, ((p_{mi}(s_{mi}) <=> p_{mj}(s_{mj}))$. The operator <=> within the interleaving formula is used as a boolean connective.

This follows from the *simultaneity* of entry of *synchronous output states*.

Axiom 3.5

$s_m \models E_r\, A_{Ir}\, (p_{mi}(s_{mi})\ \textit{until}\ p_{mj}(s_{mj})) <=>$

$s_m \models p_{mi}(s_{mi}) \wedge E_r\, A_{Ir}\, F(p_{mi}(s_{mi}) \wedge p_{mj}(s_{mj}))$

This follows from the definition of *until*. The axiom relates the *until* operator with *conjunction in the future*.

$s_m \models E_r\, A_{Ir}\, (p_{mi}(s_{mi})\ \textit{since}\ p_{mj}(s_{mj})) \Leftrightarrow$

$s_m \models p_{mi}(s_{mi}) \wedge E_r\, A_{Ir}\, \underline{F}\ (p_{mi}(s_{mi}) \wedge p_{mj}(s_{mj}))$

This follows from the definition of *since*, and it relates the *since* operator with *conjunction in the past*.

From the set of axioms and the definitions of TM and EPM, it is clear how *concurrency* among Mpm-states and the *conjunctive propositions*, *until*, *wait-for* and *co-wait* operators are interconnected along with the *interleaving* and *run quantifiers*. It is also clear as to how the modeling of concurrency aids the interpretation and hence the implementation of these operators.

3.9.2 Inference Rules

Rule 3.1

$(s_{mi} \preceq s_{mj}) \wedge (s_{mj} \preceq s'_{mi})$, for some *successor* s'_{mi} of $s_{mi} \Leftrightarrow$

$Fsv(C_0) \models E_{Cr}\, ((p_{mi}(s_{mi})\ \textit{must-wait-for}\ p_{mj}(s_{mj}))$

If proposition $p_{mi}(s_{mi})$ has to wait for $p_{mj}(s_{mj})$, it is necessary that s_{mi} is entered before s_{mj} and also the *successor* of s_{mi} cannot be entered before s_{mj}.

Rule 3.2

$\exists\ s'_{mi}, s'_{mj} \in C$ such that: s'_{mi} is a *successor* of s_{mi}, s'_{mj} is a *successor* of s_{mj} and,

$(s_{mi} \preceq s'_{mj})) \wedge (s_{mj} \preceq s'_{mi})) \Leftrightarrow$

$Fsv(C_0) \models E_{Cr}\, ((p_{mi}(s_{mi})\ \textit{must-co-wait}\ p_{mj}(s_{mj}))$

must-co-wait means that either s_{mi} is entered first and waits for s_{mj} or *vice versa*. In both cases it is true that, s_{mi} is entered before s'_{mj} and s_{mj} is entered before s'_{mi}.

Rule 3.3

$\langle CML_{\Pi M}, s_{0m}\rangle \models A_r A_{Ir} F g \vee \langle CML_{\Pi M}, s_{0m}\rangle \models A_r A_{Ir} F h \Leftrightarrow$

$\langle CML_{\Pi M}, s_{0m}\rangle \models A_r A_{Ir} F(g \vee h)$

where g and h are *primitive conjunctive propositions*.

This rule is derived as below:

$<CML_{\Pi M}, s_{0m}> \models A_r A_{Ir} Fg$ means that there is a future state s_m of s_{0m} satisfying g.

The above implies the reachability of the global-state s_m

Similarly, $<CML_{\Pi M}, s_{0m}> \models A_r A_{Ir} Fg$,

which implies the reachability of a global-state s'_m satisfying h, *in the future* of s_{0m}.

$<CML_{\Pi M}, s_{0m}> \models A_r A_{Ir} F(g \vee h)$.

The above formula implies the reachability of some global-state s''_m satisfying g *or* h, *in the future* of s_{0m}.

Hence, the validity of the rule is established.

The above rule will not hold for *conjunctive operator* between g and h since s_m and s'_m may not be reachable *at the same instant of future* if both g and h have n conjuncts (as opposed to less than n) since there can only be one *global-state* reachable at a given time which can satisfy only one *primitive conjunction* at that time.

Rule 3.4

The following inference rule is stated as a theorem below and proved.

$s_{0m} \models E_r E_{Ir} F (\wedge_{i=1..n}(p_{mi}(s_{mi}))) \wedge (\wedge_{i,j=1..n} (p_{mi}(s_{mi})$ *must-co-wait* $p_{mj}(s_{mj}))$, $i <> j$

$<=>$

$s_{0m} \models E_r A_{Ir} F (\wedge_{i=1..n}(p_{mi}(s_{mi})))$

3.9.2.1 Interleaving Theorem

Theorem 3.2 If a *conjunction of atomic propositions* referred to as *primitive conjunctive proposition* is satisfied by *some interleaving* of a given run, and if all the pairs of local propositions necessarily *co-wait* (as defined) for each other, then the conjunction is satisfied by *all interleavings* of that run.

Proof: => part:

Let $Fsv(C) = s_m$ where $C \subseteq \Sigma r$ corresponding to a run Πr, and

let C be reached by adding its members *arbitrarily* corresponding to *some interleaving* I_r of Πr.

=> s_{mi}, i=1..n, are reachable by sum-machine corresponding to *every interleaving*, i=1..n, of run Πr ;..... (1),

by the definition of an interleaving and *interleaving insensitivity* at Property 2.7 in Chapter-2.

(s_{mi}= $Fsv_i(C)$ *must-co-wait* s_{mj}= $Fsv_j(C)$), $\forall$ i, j =1..n , i $<>$ j , which is given.

=> $\models (s_{mi} \Rightarrow E_r\, A_{Ir}\, F\, (\wedge_{i=1..n}(p_{mi}(s_{mi}))))$, by the definition of *must-co-wait*;..... (2)

(1) and (2) => $s_{0m} \models E_r\, A_{Ir}\, F\, (\wedge_{i=1..n}(p_{mi}(s_{mi})))$.

The proof essentially claims that every local Mpm-state corresponding to the local proposition is reachable irrespective of the interleaving. Whichever state reaches first waits for the others in that order, which will eventually be reached .

<= part :

Follows trivially from the definition of *must-co-wait* and A_{Ir}.

Hence the theorem is proved or the rule established.

∎

This theorem and the preceding *axioms* and *inference rule* are directly applied in implementing the verification algorithms of the model-checker to check whether a given CML formula is modeled by CMpms; in particular, satisfied with respect to Σ^*M and hence with respect to ΠM and ΠF; and also in claiming the alleviation of exponential complexity due to enumeration of global states (all runs and all interleavings).

These are explained in the next chapter.

3.10 Summary of CML

The Fsm states of every F_i, i=1..n of the input CFsm system is assumed to be associated with *atomic propositions* as defined by the bijection, p_{fi}. This is used to generate the bijection p_{mi}, along with the Mpm-states of M_i, i=1..n generated in the CMpm system.

The atomic propositions are extended to general ones using the logical operators $\wedge$, $\vee$, =>, ^ etc., to form respectively conjunction, disjunction, implication and complementation. Any combination of these operators along with the *modal* and *branching operators* defined later can be used to build the *monadic, third-order CML formulae*.

A *modal* operator consists of *a 'next state'* (X), *'some future state'* (F) and *'all future states'* (G) operators qualifying the *future* of the *qualified* state and the corresponding operators to qualify its *past*. In order to take away the ambiguity or the *non-determinism* conveyed only by modal operators *in the specification of the reachability of states*, we introduce additional pairs of branching operators, one to *quantify* the *interleaving* that specifies the path(s) containing those states, and the other to quantify the *run* which contains the specified interleaving(s). This leads to monadic, third-order state formulae in CML.

Equivalence theorem III shows how the *satisfiability* of a CML formula in ΠF domain is equivalent to that in ΠM domain which in turn is equivalent to the formulae in Σ^*M domain.

The interleaving operator is a novel aspect in CML. It not only helps to *deterministically* specify the *reachability*, but also to define the *must-co-wait* and *pos-co-wait* operators corresponding to *strong concurrency* and *concurrency* (the degrees of concurrency) respectively. These are the *symmetric* extensions of *wait-for* operators, that correspond to *tense-free* versions of *until* and *since* operators.

The above and the other properties of Σ^*M used in the verification are stated as *axioms* and *inference rules*. Although it is possible to state more inference rules, only the ones to be applied in the next chapter on verification are stated.

TABLE 2 The Logics CTL Versus CML

CTL is a *monadic second-order* logic.	CML is a *monadic third-order* logic.
CTL model is a TM, defined over a blend of runs and interleavings, (the latter considered as runs as well) that are formed as *paths of a single tree*.	$CML^{*}_{\Sigma M}$ is an EPM, defined over *labelled partial-orders formed as sets of paths of n distinct state-trees*. By virtue of its equivalence with the Total-Order model $CML_{\Pi F}$, the original CTL formulae can be easily extended to $CML_{\Pi F}$ formulae, enjoying all the advantages of the EPM.
CTL therefore does not cover *partial-order semantics* corresponding to *concurrency*, nor *branching-time semantics* that must be due purely to *conflicts* only.	CML combines both the *PO* as well as *branching-time semantics*, and is implementable concretely by the equivalence of all the three versions of its models.

Chapter 4

System Verification with CMpms and CML

Having introduced the theory of CMpms with respect to a given input system of CFsms, and the three versions of the *branching space-time logic* CML, we are now in a position to introduce the *verification of* the properties of the input system. The verification algorithm is designed to check if the *safety* and *liveness* properties of the set of CFsms, specified as the $CML_{\prod F}$ formulae are *true*. The $CML_{\prod F}$ formulae are transformed to the equivalent version of $CML_{\prod M}/CML^{*}_{\Sigma M}$ formulae and their *satisfiability* are verified. The verification process consists in checking the *recognizability* of the formulae by the sum machine of CMpms generated with respect to the input CFsms.

In practice, a verification algorithm determines the reachability of a set of states (possibly a singleton) over the *disjoint state-trees* of ΣM as expressed by the $CML^{*}_{\Sigma M}$ formula (corresponding to the input $CML_{\prod F}$ formula according to the *Equivalence Theorem III* of Chapter-3). If this mechanical checking is *successful*, it is decided that the $CML^{*}_{\Sigma M}$ formula and hence its equivalent original $CML_{\prod F}$ formula is *true*. This algorithm is referred to as *model-checking* since the procedure checks the satisfiability of the formula within the *model* $CML^{*}_{\Sigma M}$ and hence within the model $CML_{\prod F}$.

4.1 Minimal prefix and Orthogonal branching in Space & Time

It may be recalled that in the Minimal prefix (Mp) of a state, the state represents its *present* and each of the other (n-1) components represents either its *past* or *present*. The concept of Minimal prefix therefore imparts the following: Every Mpm is generated as if it is the *primary* engine to execute its local events, and the rest of the Mpms are considered as the *secondary* machines with respect to the former; these secondary Mpms are driven only to execute those of their local events necessary to participate in the synchronization with the primary Mpm and thus enable the latter's subsequent progress. This perspective is applicable symmetrically to every Mpm. With this recall of the concept, we assert the following significance of Mp while *branching in space*:

When a secondary Mpm becomes a primary one, the *branching in space* is said to occur from the previous primary Mpm to the current one. By definition, Mp-vector $Mp_i(s_{mi})$ of a state s_{mi}, is the *smallest* (in the order $\leq$ among vectors) in the equivalence class $[Mp_i(s_{mi})]_{RMp_i}$ such that each non-local component is necessarily entered before or in partnership with s_{mi}. This condition ensures the fact that upon switching the primary Mpm from M_i to M_j and continuing the traversal from $s_{mj} = Mp_i(s_{mi})(j)$, we *maintain the current branch of time* chosen by a local branch of M_i. The condition also ensures that all the states in M_j that are concurrent with s_{mi} are reachable because $M_{pi}(s_{mi})$ is the smallest of such concurrent vectors, and we are continuing from one of its non-local components that is asynchronous to s_{mi}. This does not rule out the states of M_j that are reachable in synchrony with s_{mi}. But our main goal is to maintain s_{mi} as the destination state once the local atomic proposition of M_i is reached. This will be explained further in the following sections.

4.2 Monadic Third-Order CML Formulae handled by the Model-checker

We consider *monadic third-order formulae* of CML.

A monadic (third-order) formula is a *state formula*, which consists of a *run operator quantifier* followed by a *run formula*, which is again a *monadic second-order formula* consisting of an *interleaving quantifier* followed by an *interleaving formula*, which is a *first-order formula*, consisting of a *modal operator* followed by a *proposition*.

We consider only propositions in *disjunctive normal form* and so formulae of the following form:

<run opr><interleaving opr><modal opr> $(g_1 \vee g_2 \vee \ldots \vee g_q)$ for some q >1 where:

<modal opr> is a modal operator such as X, F, G (*until* is expressible in terms of F) or their corresponding *past* operators and,

each g_i is a *primitive conjunction* which is a conjunction of *atomic propositions*, (as defined in Chapter-3) such as:

$g_i = (ap_{f1} \wedge ap_{f2} \wedge \ldots \wedge ap_{fk})$, in ΠF domain, where $k \leq n$, $i = 1..q$.

It is to be noted that the *complements* of atomic propositions are excluded from consideration in the *primitive conjunction* for the same reason as their disjunction (as in the *conjunctive normal form*) as explained in Chapter-3. It may be recalled that checking of the *satisfiability* of a proposition consists in searching for a state that satisfies the proposition.

The *negation* of an atomic proposition is satisfied by much more states than by its assertion that is specifically satisfied by only *certain target Mpm-states* holding their respective atomic propositions. Therefore, checking for *negation* of a proposition is *non-deterministic* as well as a *disjunction of multiple atomic propositions* and may involve the checking of the reachability of combinatorial number of global-states, at the worst. But in practice, formulae involving disjunctions and complements are equally decidable and tractable so long as they are not exponential in size, as explained in Chapter-3 which is reiterated in the following section for emphasis.

4.2.1 Choice of Propositions handled by the Model-checker

The fact that the *satisfiability* of a CML formula with a *primitive conjunction* qualified and quantified by the modal and branching operators respectively, corresponds to the *reachability* of global-state(s) was noted in Chapter-3. The *conjunction among atomic propositions* imposes *concurrency/strong concurrency among (non-local) Mpm-states* and hence involves *possible/necessary wait-for* operator or the *until/since* operator as shown in Axiom 3.5 of Chapter-3. Thus, the primitive conjunction thus seems to be the most non-trivial element of CML formulae that are checkable, as it requires branching-off/switching of Mpms as many times as the number of conjuncts. The following section reinforces this idea.

4.2.1.1 Polynomial Versus Exponential size of input CML formulae Checked

Interpreted in ΠF domain, satisfiability of a primitive conjunction of n conjuncts corresponds to reachability of a unique global-state by possibly multiple runs and interleavings. Verifying the satisfiability of a CML formula in the *disjunctive normal form* therefore corresponds to checking the *reachability* of as many global-states as there are disjuncts and thus the complexity of the formula is polynomial in the number of disjuncts.

On the other hand, let us suppose that we want to check the satisfiability of a CML formula in *conjunctive normal form* having n conjuncts, where each conjunct is a disjunction of up to m local atomic propositions. With every local disjunct, it is possible that any of the m non-local disjuncts can be in conjunction due to asynchrony/concurrency among the local states. Consequently, the actual size of the input formula is exponential with (m ** n) *primitive conjunctions*. This amounts to checking for the reachability of that many *global states* and leads to exponential complexity of the formula checked.

In the worst case, checking for the reachability of one global-state itself may involve exploring all interleavings of all runs. So, it seems reasonable to confine ourselves to *primitive conjunctions* which are reasonably complex; specifically, they are equivalent in power and so can substitute the formulae with *wait-for/until* operators that are considered to represent the most complex of all formulae in CTL[1].

The complementation operator amounts to a disjunction of a large number of local atomic predicates. We do not consider formulae with complementation nor disjunction of atomic propositions, in the conjunction. This does not mean that the CML formulae with conjuncts involving disjunction or complementation can not be checked. The methodology that we discuss can be applied as long as the input size of the formula is not exponentially high.

4.2.2 Translation of $CML_{\Pi F}$ to $CML^*_{\Sigma M}$ Formulae

Checking the *satisfiability* of a CML formula involves, finding the interleaved path(s) within one or more run(s) in which the state s_f can be reached in ΠF such that:

$<CML_{\Pi F}, s_f> \models (ap_{f1} \wedge ap_{f2} \wedge .... \wedge ap_{fk})$ where,

$p_{fi}(s_{fi}) = ap_{fi}$, for all i =1..k.

The above satisfiability-checking is equivalent to finding the corresponding *interleaved paths* within the *run(s)* of ΠM in which s_m can be reached such that:

$<CML_{\Pi M}, s_m> \models (ap_{m1} \wedge ap_{m2} \wedge ... \wedge ap_{mk})$ where,

$p_{mi}(s_{mi}) = ap_{mi}$, for all i =1..j and,

$B(s_m) = s_f$,

$B_i(ap_{mi}) = ap_{fi}$, i=1..j ,

$B(\Pi I_r) = \Pi I_{rf}$ and,

$B(\Pi r) = \Pi r_f$.

The $CML_{\Pi F}$ formula, $<CML_{\Pi F}, s_{0f}> \models A_r E_{Ir} F (ap_{sfi} \wedge ap_{sfj} \wedge ap_{sfk})$ ....(i)

where ap_{sfi}, ap_{sfj} and ap_{sfk} are the *atomic propositions* of F_i, F_j and F_k respectively is *equivalent* to the following $CML_{\Pi M}$ formula:

$<CML_{\Pi M}, s_{0m}> \models A_r E_{Ir} F (ap_{smi} \wedge ap_{smj} \wedge ap_{smk})$ ....(ii) where ap_{smi}, ap_{smj} and ap_{smk} are the *atomic propositions* of M_i, M_j and M_k respectively by virtue of the functions B, and ΣB_i, i=1..n., and by the *Equivalence theorem III* viz.,Theorem 3.2 of Chapter-3.

The reachability of a state s_m in the future of s_{0m} in formula (ii) above, satisfying the conjunctive proposition ($ap_{smi} \wedge ap_{smj} \wedge ap_{smk}$) in turn consists in finding a configuration C in Σ^*M corresponding to ΠM such that :

$C \subseteq \Sigma r$, corresponding to Πr where $Fsv(C) = s_m$.

The following $CML^*_{\Sigma M}$ formula in Σ^*M domain is *equivalent* to the formulae (i) and (ii), again by virtue of Theorem 3.2, and the definition of the function p_{mi} of Chapter-3:

$<CML^*_{\Sigma M}, Fsv(C_0)> \models A_r\, E_{Ir}\, F\, (p_{mi}(Fsv_i(C) \wedge p_{mj}(Fsv_j(C) \wedge p_{mk}(Fsv_k(C))$ ....(iii)

where,

$Fsv(C_0) = s_{0m}$ and $Fsv(C) = s_m$, $p_{mi}(Fsv_i(C)) = ap_{smi}$.

The other combinations of branching operators are handled similarly .

A proposition in *disjunctive normal form* is a disjunction of above conjunctions in which case, the checker algorithm is repeated as many times as there are disjuncts; this follows from Rule 3.3 of Chapter-3.

This is how the formulae (i), (ii) and (iii) in CML are linked with each other in all three domains of ΠF, ΠM and Σ^*M respectively. Since the *configurations* in Σ^*M domain are generated independent of their interleavings, the corresponding runs in ΠM domain are also generated independent of their interleavings. In contrast, if the product machines ΠM and ΠF were to be directly generated, interleavings and runs can not be distinguished since the *non-deterministic choices* due to interleavings are intertwined with the choices due to *conflicts* among the runs. In these product machines, both the *non-deterministic* and *true choice* entities merge and all of them are treated as *runs* themselves, in these product machines.

The model-checker algorithm checks the $CML^*_{\Sigma M}$ formula upon the *extended sum machine* Σ^*M by generating the required configurations as needed by the formula while traversing the *basic sum-machine* ΣM.All the general runs are generated from the local runs by associating them with general and local configurations respectively according to *Summation Lemma* (Lemma 2.9) of Chapter-2.

4.3 Definitions of Keywords used in Model-checking

Following are the definitions, possibly repetitions, that are useful to recall here for the benefit of the rest of the discussion on model-checking:

- Primitive conjunctive proposition: The conjunction of up to n atomic propositions of the n Mpms of ap_{mi}, i=1..n, is a *primitive conjunctive proposition.*
- Current configuration: The current configuration is a configuration C of ΣM, which is the union of local configurations of all the Mpm-states visited thus far corresponding to a run $\Pi r \subseteq \Pi M$ such that: $C \subseteq \Sigma r$, the latter being a *conflict-free sum-machine* corresponding to Πr. At the beginning of the algorithm, C is set to C_0, the initial configuration of ΣM.
- Primary and Secondary Mpms: The Mpm M_i, whose *state-tree* is being traversed from the *current configuration* in a depth-first manner to cover the *paths* of the state-tree, that belong to different local configurations corresponding to local runs. The traversal *branches in time* at every *conflict point,* as dictated by $conf_i$, until the local atomic proposition ap_{mi} is encountered. The rest of the non-local Mpms M_j, j < > i are the secondary Mpms encountered while traversing M_i.
- Local continuations: All the configurations $C' = C \cup C_i(s'_{mi})$ such that: C is the current configuration and s'_{mi} are the descendents of $s_{mi}=Fsv_i(C)$, that are *configurable* with C are called local continuations.
- Switching configuration: The current configuration C at the time of *branching in space or switching* from the current primary Mpm M_i upon reaching s_{mi} satisfying $ap_{mi} = p_{mi}(s_{mi})$, to a secondary Mpm M_j (the next primary Mpm) to check for ap_{mj}, is called the switching configuration.
- Handle-state: When the switching of primary Mpm takes place from M_i to M_j, if the switching configuration is C, then $s_{mj} = Fsv_j(C)$; that is, the j^{th} component of the *final state vector* of C is the *handle-state*. This is the state from which the traversal of M_j begins rather than from its initial state.

- *Traversal and Visit:* An Mpm-state is said to be *visited* and the visits of a set of Mpm-states, say, a configuration is collectively said to be *traversed* during the nested, depth-first search of Mpm-trees.

4.4 Outline of Model-checking

4.4.1 Distributed, Nested Depth First Search

The model-checking algorithm works over ΣM to verify the properties of ΠF.

It involves *the recursive, depth-first* search of state-trees of M_i, i=1..n in a *distributed, nested* fashion; that is, the depth of traversal within one state-tree is carried over to the next tree in ΣM. *The Mp-label of every state encodes the minimal depth of traversal of every other non-local tree, in order to enable the reachability of the current local state.* This aids in branching from one Mpm-tree to the next one as a continuation of the same run once the local atomic proposition is reached, as explained in previous subsections.

For instance, when the local conjunct ap_{mi} is found in M_i, we switch to M_j as the next *primary* Mpm to check through its local continuations of the switching configuration C for the next *local conjunct* ap_{mj}, using $Fsv_j(C)$ as the *handle* state.

For *every* switching configuration reached at a local Mpm, there are *multiple* continuations to be traversed and checked at the next Mpm from the handle-state. This process is continued until all the conjuncts are satisfied. This may mean traversal of the same Mpm-tree more than once, in cases when the satisfied conjunct no more holds due to synchronization requirements of the local Mpm with non-local ones. However, since the traversal is *continuous in time*, no Mpm-state is visited more than once as a member of the primary Mpm-tree. However, as a member of two conflicting local-configurations of a given non-local, primary Mpm, a state of a secondary Mpm may be visited more than once which is accounted for in the visitation of the entire configuration of every traversed state of the primary Mpm.

The above procedure is referred to as the *distributed nested, depth-first traversal, as the entire depth of one single time continuum is broken up across multiple Mpm-trees.*

4.4.2 Disjointness of the Search

As proved in the *Monotonicity theorem* of Chapter-2, the recursive, depth-first search is suitable for traversing the configuration that grows *monotonically* with the local transitions of ΣR_{mi} corresponding to every run Πr of ΠM. The *general configuration* that is the *union of local configurations* of the Mpm-states belonging to a given run is maintained as the current configuration C, by the *Summation Lemma* of Chapter-2.

ΣM as shown in Fig. B of *Appendix* represents the *global, causal dependency-order* $\leq$, running across all Mpms through the *synchronization points* (the synchronous output states) represented *explicitly*. In this case, every local configuration has to be formed by backtracking across all the synchronization points and states of all Mpms from a given state, thus building its *upward closure literally* as dictated by its definition in *Definition 2.17*.

Fig. C and Fig.D of *Appendix* represent M_i,i =1..3 in *disjoint form*, without showing the synchronization points explicitly as in Fig. B. Instead, Mp-vector of every state is stored in its node in Fig. C or separately as a table in Fig. D. *Minimal prefixes are the derivatives of synchronization points* according to Mp-lemma at Lemma 2.2.

By storing the Minimal prefixes, a local configuration can be formed as a *disjoint union of n conflict-free paths* according to *disjointness theorem* of Chapter-2 rather than as an *upward closure* using directly the synchronization points. The former view of a configuration is more efficient and useful for the checking of *configurability* by applying the *configurability theorem* of Theorem 2.4. This is because, the checking is divided into a fixed set of n *disjoint, paths* (*untangled*) in the latter, as opposed to the *non-disjoint (tangled)*, overlapped paths without any structure in the case of the of the former. But it takes n times more space to store the Mp-labels explicitly in the representation of Fig. C & Fig. D. This issue will be illustrated with an example in one of the following sections.

4.4.2.1 Conservation of Visits in the Recursive Search

The strategy of depth-first search is quite appropriate since the membership of a'state is conserved among the successive set of configurations. Thus we minimize the number of visits necessary to cover any *succession* of configurations during the traversal of ΣM. Once an Mpm-state is *visited* as a member of the *current configuration* C, it continues to be retained as a member of *all the continuations* C' of C such that C $\subseteq$ C' until the *cut-off point* of each continuation. It is described with the following illustration:

Example 4.1

From Fig.C of *Appendix*, considering M_2, the Mpm-states p_0 through t_0 and their respective local configurations (and thus paths of other Mpms M_1 and M_3) will be *retained* by all the configurations up to the maximal, cut-off configurations leading to the cut-off states p_2, p_1 and s_1 respectively, and also by all the non-local continuations, continued from the handle-states. In this example, there is no such non-local continuation since the cut-off states of M_3 viz., x_2, x_1 and x_3 are already contained in the cut-off configurations of p_2, p_1 and s_1 respectively of M_2.

Similarly, the state u_0 of M_2 is retained by the configurations of v_1, p_2 and v_0, p_1 in a cumulative fashion, as well as the non-local continuations of these configurations, *till* the time when u_0 is *unvisited* when the search backtracks and then continues from the sibling r_0, of u_0.

The above is true for all Mpms, primary and secondary together, the latter through the Mp vectors of primary Mpm-states. This is how the number of visits of Mpm-states are minimized.

4.4.3 Localized Search implies Global — Non-enumeration of Runs

The *conflicts* originate locally, which alone are propagated *globally*. Therefore, it is possible to check for a primitive conjunction by focussing on a single Mpm as a primary one, and checking *only* its local configurations till the local conjunct is satisfied, as though it is the only state-tree to be traversed. During this primary traversal, the paths traversed in the secondary Mpms are monitored as well by the Mp-labels.

The *run quantifier* A_r or the *configuration quantifier* A_{Cr} of CML corresponds to conflict points handled by the distributed, local conflicts represented by $\Sigma conf_i$. This disjoint union of local conflicts lead to local configurations that are in conflict. They also account for the general *conf* relation automatically when the local configurations are formed as the *upward closure* of Mpm-states. The *inheritance* propagated by the *causal dependency-order* $\leq$ accounts for the difference, ($conf - \Sigma conf_i$). The set of local configurations

$\Sigma C_i(s_{mi})$, for all s_{mi} in S_{mi}, i=1..n splits all the states ΣS_{mi} into conflict-free sets. As local continuations are traversed in one Mpm M_i, as formed by *conf*$_i$, the non-local continuations due to *conf*$_j$, j<>i are traversed as well as dictated by the *causality* $\leq$ and monitored by Mp-labels of the function Mp_i.

In practice, as we traverse the states and events of any one Mpm M_i *primarily*, we are traversing different *branches of time* as decided by the *conflict points* due to *conf*$_i$ of M_i, and *in space* associated with M_i. When another Mpm M_j is chosen to be a primary one, the *branching in space* takes place from M_i to M_j. Upon doing so, the current branch of time as decided in M_i can be maintained in M_j by continuing its local traversal from $s_{mj} = Fsv_j(C)$ (where C is the switching configuration) rather than beginning the traversal of M_j right from the *initial state* s_{m0j}.

These ideas were formally expressed as *Complexity Lemma* at Lemma 2.14.

Example 4.2 In Fig. B & Fig. C of *Appendix*, after traversing M_2 up to v_1, let us say that we branch-off in *space* to M_3. Since $Mp_2(v_1) = (d_0, v_1, z_0)$, the traversal of M_3 has to be continued from z_0. By traversing M_2 up to v_1, we have also traversed M_1 up to d_0 and M_3 up to z_0, thus maintaining the current branch of *time* across all the non-local Mpms as well.

The above phenomenon is quite advantageous because, it was noted in Chapter-2, that the union of local conflicts Σ*conf*$_i$ in ΣM is much smaller compared to *conf*$_g$ of ΠM. In addition, we now see that by covering the local conflicts *conf*$_i$ of one Mpm M_i, we are also covering *in parallel*, *conf*$_j$, j<>i through the medium of Minimal prefixes, *without* the need for *exhaustively traversing* M_j from the initial state. This issue is addressed as *non-enumeration of runs*, for runs are the entities caused by conflicts. The resulting complexity savings will be elaborated in a future section.

Furthermore, during the traversal of M_j after that of M_i i<>j, there is a subset of local continuations due to *conf*$_j$, that will not be the continuations of the *switching configuration* C as explained in what follows.

4.4.4 Secondary Mpms, Continuations and Configurability

If Mpm M_j is chosen to be a primary one after the local traversal of M_i, and the traversal *continued* from the j^{th} component of $Mp_i(s_{mi})$, we *branch-off* in space. Since we are *continuing* the traversal in *time* from the *past* of s_{mi} (possibly from its *present* itself if s_{mj} is a synchronous output partner of s_{mi}), it is possible that some of the local future states of s_{mj} may be in conflict with s_{mi}; this possibility arises from the definition of the Minimal prefix and the fact that s_{mi} is possibly a *future* state of s_{mj}.

Therefore, with every switching of the primary Mpm, the number of local continuations to be considered will become progressively smaller than $|conf_k|$, where M_k is the switched primary Mpm.

Example 4.3 Let us consider the same example in Fig. C, where $Mp_2(v_1) = (d_0, v_1, z_0)$. After traversing M_2 up to v_1, let us say that we branch-off in space to M_3 and continue the traversal from z_0 on M_3. Now, even though h_0 is a successor (and hence a possible future) of z_0, since the current state (i.e., the *present*) is at v_1 and it is in conflict with h_0 (as shown by the implication below), it *can not be a future* of v_1. Therefore, the traversal *in time* up to v_1 (in M_2's space) *can not be continued* to visit h_0 (in M_3's space).

The above is the consequence of *conflict inheritance*:

$(u_0\ conf_2\ r_0) \wedge (v_1 > u_0) \wedge (h_0 = r_0) \Rightarrow (v_1\ conf\ h_0)$.

The state h_0 lies at a different *branch of time* that conflicts with v_1.

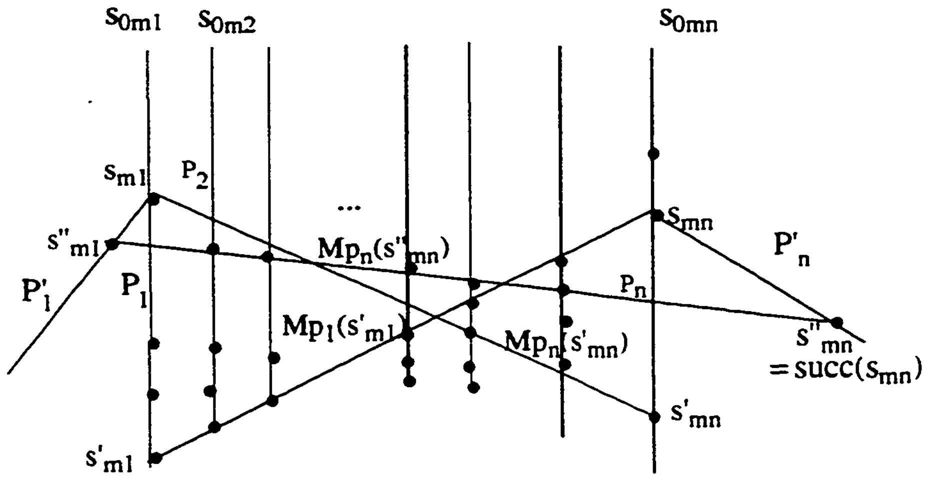
Instead of visiting v_1 at M_2, v_0 could have been traversed with $Mp_2(v_0) = (d_0\ v_0\ g_0)$. In that case, when we branch-off in space to M_3, the traversal will be continued from g_0 which is in local conflict with h_0. This is another *branch of time* from the one discussed above.

The above example illustrates that, upon *branching-off in space* from one Mpm M_i to the other M_j, some of the local successors of the latter may not be *continued* as *branches of time*, as they are in conflict with the current branch of time originated at a conflict point of M_i.

This is because, the traversal in M_j starts possibly from the *past* of s_{mi}, such that: $(s_{mj} < s_{mi})$. In particular, s_{mj} could have a successor s'_{mj} such that: $(s'_{mj}\ conf\ s_{mi})$. Consequently, s'_{mj} will not be *configurable* with the switching configuration C.

Referring to Fig. 15 below, let us say that the traversal begins at M_1 and then switches to M_n upon reaching the state s'_{m1}. This is when the current configuration $C = C_1(s'_{m1})$ is reached such that,

Fig. 15 Configurability of Local ones to derive General Configurations



$Fsv(C) = Mp_1(s'_{m1})$, shown by the vector s'_{m1}-s_{mn}. After branching off to Mpm-tree M_n, there are two successors for s_{mn} viz., s'_{mn} and s''_{mn}. The n paths of $C_1(s'_{m1})$ and those of $C_n(s'_{mn})$ form a conflict-free union of their respective n paths, to form a new configuration containing the two of them. We say that s'_{mn} is configurable with C. s''_{mn} is not configurable with C because, $C_n(s''_{mn})$ contains path P'_1 which is in conflict with P_1 of C.

This phenomenon cuts down the complexity of traversal further more, in addition to the savings discussed in the previous section.

4.4.5 Cyclicity Theorem

The *cyclicity theorem* to be stated below, is applied to terminate the model-checking algorithms after partially traversing the current run, to decide on the falsity of a given formula checked.

Theorem 4.1 If a *cut-off state* is reached during the traversal of ΣM without satisfying a *conjunctive proposition*, then there is at least one *run* in which the proposition and hence the formula is not satisfied.

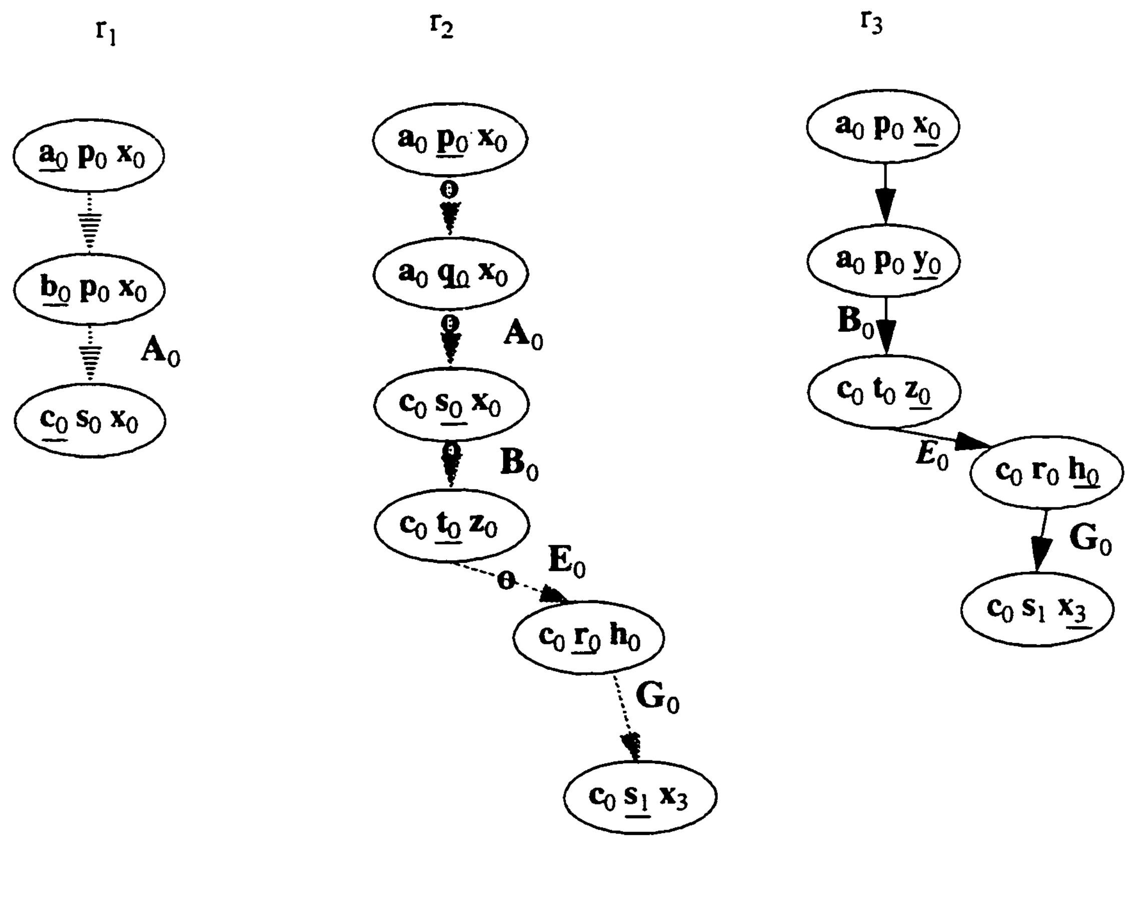
Proof: The proof follows from the definition of the *cut-off* state. It is recalled that from a cut-off state, there is a *recurrence* of the *past* of global states with respect to ΠF.

In particular, the recurrence of the exact *path* (in ΠF domain) that led to the given cut-off state is possible infinitely. Therefore, if the first cycle of states till cutoff was reached without the success of satisfying the required conjunction, the same will be the case with all subsequent cycles of recurrences. Hence the result.

■

Example 4.4

Fig. 16 A conflict-free sum-machine Σr corresponding to a run.



In the conflict-free sum-machine of the above figure, s_1 is a cut-off state in M_2, the *cut-off vector* being $Mp_2(s_1) = (c_0, s_1, x_3)$, with a corresponding *basis vector* $Mp_2(s_1) = (c_0, s_0, x_0)$. Suppose we check for the formula, $s_{0r} \models E_r\, E_{Ir}\, F\, (ap_c \wedge ap_g)$. Upon reaching c_0 at M_1, we switch to M_2 using s_0 as the handle state. When s_1 is reached, since it is a cut-off state, we can be assured that there is at least one run, viz., the retrace of what was thus far covered, leading to infinite number of cycles without reaching any occurrence of local Mpm-state: (g, occ#) satisfying ($ap_{g.\,occ\#}$) such that: $B(ap_{g,\,occ\#}) = ap_g$. If the local conjunct is not satisfied, the whole conjunction can not be satisfied either.

4.5 Fairness among Mpms and Model-checking

Fairness has been dealt with in Chapter-2 at some length.

The advantage of ΣM is that the recognition of unfairness is quite easy because of the maintenance of *identity locality of the processes* represented by individual Mpms.

Every path of the state-tree of an Mpm is associated with *at least one distinct run*, as can be seen without proof. So, considering only fair runs with respect to a given Mpm is equivalent to considering *all the paths or branches of the state-tree* so long as the configurability is not violated after branching in space from a previous primary Mpm to the current one.

The above source of unfairness just like local conflicts *propagates* to the non-local Mpms through *synchronization causality*. In fact, it turns out that this propagated unfairness is easier to detect rather than within the original source of occurrence within a given Mpm. Also, often we are more interested in one or more processes corresponding to non-local Mpms starving a local process/Mpm under consideration (rather than local runs within that Mpm knocking out each other); we say that the (CMpm) system is *unfair to* the given process/Mpm. The source of this *unfairness* is contributed by the *asynchronous, non-local conflicts* defined in Definition 2.37 of Chapter-2. This is more precisely stated by the *unfairness theorem* below.

4.5.1 Unfairness Theorem

This theorem is applied in detecting the termination of model-checking algorithm with *universal run* operator, whenever the assumption that *runs need not be fair* holds good.

The definition of asynchronous, non-local conflict denoted: ***anl-conf*** stated at Definition 2.33 of Chapter-2 is recalled and applied in this theorem.

Theorem 4.2 If a synchronous output state is in *asynchronous, non-local conflict* with a *cut-off* state, then there is *at least one run* in which neither that state nor any of its local descendents (with respect to the order R_{mi}) will be reached in the future.

Proof: It follows from the definition of *asynchronous non-local conflict* and *unfairness lemma (Lemma 2.13)* of Chapter-2 and the *cyclicity theorem* Theorem 4.1 in the last section.

Let (s_{mi_out} *anl-conf* s_{mj_cutoff}) where s_{mj_cutoff} is a cut-off state.

Let also s_{mi_out} be a *synchronous output state* with a corresponding input state, s_{mi_in}.

The above means that there is a run in which, while s_{mi_in} of M_i is waiting for its partner input state s_{mj_in} of M_j , s_{mj_cutoff} in conflict with s_{mj_out} is entered instead. Since s_{mj_cutoff} is a cut-off state, by *cyclicity theorem*, the recurrence of the same sequence of past states (with respect to ΠF) of s_{mj_cutoff} can take place infinitely; hence s_{mj_out} is never entered nor is its partner output-state s_{mi_out}.

Thus, the entry of s_{mi_out} is prevented infinitely by non-local states, and so if the local atomic proposition ap_{mi} is not satisfied in some partial run (configuration) before the entry of s_{mi_out}, there is no possibility of it being satisfied henceforth in that particular run (which is unfair) with s_{mi_out} in it.

Hence the result is proved for all states s_{mi_in} satisfying the above condition.

If the local proposition cannot be satisfied, any conjunction with the local proposition as a component conjunct cannot be satisfied either. So, this theorem is useful in pre-terminating the checking of a primitive conjunctive proposition qualified by the *universal run* (A_r) *operator* even *without exhaustively traversing all the Mpms including the current one* whose local propositions are the component conjuncts. This pre-termination of the model-checking/verification algorithm is done with the assumption that unfair runs are allowed.

On the other hand, if we *consider only fair runs*, we disregard all the states that are in *asynchronous, non-local conflict* with cut-off states while the current run/configuration is being traversed. In that case, we would be considering all the processes in a fair manner; in other words, only the fair runs would be considered.

This can be easily taken into account in the *model-checker* algorithm, by toggling between checking and not checking for *cut-off states in asynchronous, non-local conflict* with the *synchronous output states* encountered during traversal. The propagation of non-local conflict is always through *synchronization points* as it follows from theory of CMpms. *This is yet another advantage of maintaining the synchronization points and causality among state entries so that allowing unfair runs or disallowing them is implemented easily*.

Example 4.5 We consider the formula $s_{0f} \models A_r E_{Ir} F (d \wedge u \wedge z)$.

In checking this formula from Fig. C of Appendix, when d_0 is reached in M_1, we have the information that (d_0 *anl-conf* s_1) where s_1 is a *cut-off state*. By *unfairness theorem*, neither d_0 nor its successors are reachable in the *unfair run* with s_1 as its *cut-off state*. So, the formula is false.

On the other hand, if we consider only *fair runs*, we ignore the above *anl-conf* between d_0 and s_1, which means that we disregard the unfair run that starves d_0. So, the formula is true, *when we assume that the unfair runs are disallowed/ruled out.*

Thus, the handling of unfairness is quite simple in ΣM. This is another result of distributed storing of only local states and their continuations as well as state-based causality through the synchronization points.

4.5.2 Non-monadic, Nested CML formulae and Labeling Algorithms

It is recalled that if a *monadic, third-order state formula* is treated in place of a *primitive state formula* and treated as an *operand* of a modal operator to get an interleaving formula followed by a run formula and so forth *inductively* according to the original definition of syntax of CML in Chapter-3, what results is a nested, non-monadic formula of higher-order than the third.

The model-checker discussed so far, for the monadic CML formulae does not use any *labeling procedures*, as the *nested, depth first search* does the checking of the *primitive conjunction* accounting for all the runs and/or interleavings with a *single pass algorithm*, without breaking-up /splitting the formula into sub-formulae as by the traditional procedures [1].

The extension of the model-checker for non-monadic formulae seems quite feasible if suitable labeling strategies are adopted. This is left for future-work, which we believe is a question of extending the implementation, without necessitating any drastic change in the theory.

4.6 System Invariants and Deadlocks Detection

Definition 4.1 *System Invariant* is defined as a property of the system, which is true *for all states* of the system, starting from its initiation. Conventionally, it is also defined as a condition which when satisfied (or true) is true for ever. It is a *safety* property of the system and is associated with a *stable predicate*, which often corresponds pragmatically to a condition when, there is no local progress of the system with respect to a subset of component Mpms after the condition is satisfied.

System invariant usually involves a basic proposition of *implication*. It takes the form of a proposition h with the modal operator G such that when h becomes true, it remains true for ever: a stable property.

$\models (h \Rightarrow E_r A_{Ir} G\, h), \ \forall s_f \in S_f$ of ΠF

The above is a specification of a *validity formula* as the implication operator makes it *state independent*. This is because, every state either satisfies or not satisfy h. The implication operator suffices to express that all the future states of *those states satisfying h* should also satisfy h.

Example 4.6 From Fig. 16 on page 161,

$\models (c_0 \Rightarrow E_r A_{Ir} G\, c_0), \ \forall s_m \in S_m$ of ΠM.

i.e., If c_0 is satisfied by some state, it is implied that it will be satisfied by all its future states belonging to all interleavings of some run.

The above is an expression of a *validity formula*, as it is state-independent.

4.6.1 Deadlocks

Deadlock is a condition when there is no progress in the system. *Deadlock freedom* is again a *safety* property.

Dead state is an Mpm-state which does not have any *successors* nor is a *cut-off* state.

4.6.1.1 Deadlocks Detection

Detecting a deadlock consists in reaching a global-state Fsv(C) in which, every component $Fsv_i(C)$ is a *dead state* or, does not have a *local continuation* that is *configurable* with C. It can be checked similar to any *primitive conjunctive proposition*, posed as the following *deadlock-detection formula*:

$s_{0m} \models E_r\, E_{Ir}\, F\, (\wedge_{i=1..n}\, p_{mi}(s_{mi}))$, where s_{mi} is a *dead-state* or *devoid of a local continuation*, for all i=1..n.

The formula above can be checked by the *model checker* algorithm , with a minimal variation as follows:

All Mpms M_i, i=1..n are traversed in some arbitrary order until a state s_{mi} which is a *dead state* or does not have a local continuation of the current configuration C is encountered. Upon reaching it, using its *handle-state* $s_{mj} = Fsv_i(C)$, j <> i, switching/branching off (in *space*) from M_i to M_j is made as the next *primary* Mpm, and traversal among the latter's local continuations carried out in the same manner as in M_i, for all j <> i.

During the traversal of M_i, i=1..n, if a *cut-off* state is reached, there is no dead-lock in the corresponding *run* and branching-off in *time* to another *run* (by visiting a sibling state) in the depth-first search is made within the Mpm-tree traversed.

It is interesting to observe the following result posed as a theorem:

Theorem 4.3 A deadlock detected for some interleaving is satisfied by all interleavings.

i.e., $s_{0m} \models E_r\, E_{Ir}\, F\, (\wedge_{i=1..n}\, p_{mi}(s_{mi}))$, where s_m is a dead-locked state.

$\Longleftrightarrow s_{0m} \models E_r\, A_{Ir}\, F\, (\wedge_{i=1..n}\, p_{mi}(s_{mi}))$

If s_{mi} can be reached in one interleaving, it is reachable by all interleavings, by the *interleaving insensitivity* property (Property 2.7) stated in Chapter-2. s_{mi},i=1..n is either a dead-state or does not have a local continuation, from the assumption that s_m is a deadlocked state. s_{mi} reached by some interleaving, carries its property to other interleavings.

This is because,

(i) Successors are independent of interleavings and so, if there is no successor for s_{mi} in one interleaving, it is the case when it is reached by other interleavings as well.

(ii) The conflicts are preserved across interleavings since they relate local states only, and so its configurability with a successor remains the same if there is one for s_{mi}.

Therefore once s_{mi} is reached, it remains there for ever and so cannot but wait for other components s_{mj}, $j<>i$ to be reached in every interleaving. Hence the result follows.

4.7 Sum machine Generator & Model-checker Algorithms

The model-checking involves *one-time generation* of the sum-machine ΣM that is used for checking/verifying all the CML formulae by the model-checker. A straightforward version of the *generator algorithm* is presented first, that has an *exponential complexity*. This is followed by the presentation of a modified algorithm that avoids the exponential complexity by eliminating the repeated visits of the already generated states. After analyzing these complexities, we present the *model-checker algorithm* and its complexity analysis.

The algorithms are presented in pseudo code, more or less in the style of those of [1] and [3].

4.7.1 ΣM Generator Algorithm (I)

Input	A set of n CFsms F_i,i =1..n with synchronous events and their partner Fsms as shown in Fig.A of Appendix .
Output	A set of n CMpms M_i,i=1..n (and hence ΣM) corresponding to the input CFsm system along with the functions B_i,i=1..n (and so B).

Main data-structures:

s_m	Global Mpm-state vector
Mpm_state, s_{mi}	{s_{fi}, occ#}, where s_{fi} is the Fsm-state and occ# is to make the Mpm-state unique, as generated by the *auxiliary functions* f_i, f_{syncij}.
$Mp_i(s_{mi})$	Minimal prefix vector of the state, s_{mi}.
$r_{tfi} \in R_{tfi}$	Transitions of Fsm, F_i.
$r_{tmi} \in R_{tmi}$	Transitions of Mpm, M_i.
wait_stack	This is a stack, that stores the pairs <i, r_{tfi}> that are waiting to synchronize with their partners. When M_i waits for M_j, it pushes its entry into wait_stack and inputs it to M_j. If M_j has to wait for M_k, it appends its entry <j, r_{tfj}> into the wait_stack and passes it on to M_k and so on. When wait_stack = Null, it means M_i the currently generated Mpm is not waiting for any partner.
partners_list	List of ordered pairs <s_{mj_in}, r_{tfj}> that form the input partner state and partner Fsm transition (to be simulated) of s_{mi_in}, r_{tfi} of the waiting Mpm M_i respectively, synchronizing on $r_{tfi}.e_i = r_{tfj}.e_j$.
visited($B(Mp_i(s_{mi}))$)	Boolean flag to keep track of visited Fsm-global-state corresponding to Minimal-prefixes of Mpm-states to detect the *cut-off* states. Set to true when first visiting an Mpm-state s_{mi} and remains true during the visits of all its

continuations (local and non-local alike). Set to false when s_{mi} is no more contained by the currently generated state's local configuration.

```
global Fi, Bi, i=1..n, ΣM, sm, wait_stack
procedure generate_all_Mpms( )
{
    s0mi := (s0fi, f0i(Null, r0tfi)); Bi(s0mi) = s0fi, for i=1..n; /*Bi for initial states generated*/
    for all i =1..n
    {
        Store s0mi ; store Mpi(s0mi) := (s0m1, s0m2,..., s0mn);
        C0i := Ci(s0mi) := {s0mk | k= 1..n};
         generate_Mpm(Mi, s0m);
    }
}
/*generate_all_Mpms*/

procedure generate_Mpm(Mi, sm)
{
    if visited(B(Mpi(smi)))
    {
        cutoff(smi) := true; /* Mark smi as a cut-off state since it has a basis vector as
            an ancestor in CFsm domain.*/
        return;
    }
    visited(B(Mpi(smi))) := true; /* B(sm) is derived as (B1(sm1), B2(sm2),...,Bn(smn)) */
    /* Generate all the successors of smi and their Mp-vectors in depth-first fashion */
    for (all rtfi = (sfi_in, efi, sfi_out) of Rtfi such that: Bi(smi) = sfi_in) do
    {
        sm_succ := sm; /*initialize successor state-vector to be used by the next level of
                        recursion*/
        if (efi is local event)
        {
            smi_out := (sfi_out, fi(smi_in, rtfi)) ;
            Bi(smi_out) = sfi_out; /* An element of Bi is generated */
            sm_succ(i) := smi_out; /*successor vector is thus updated*/
            compute_and_storeMp( smi_out, Mpi(smi)); /*generate and store Mpi(smi_out)
                                using Mpi(smi) as the boundary*/
            generate_Mpm(Mi, sm_succ); /* recursive call to generate its successor
             state*/
        }
        else if (efi is synchronous  between Fi and Fj)
```

```
{
    if <j, e_fj > is at wait-stack /*M_j is waiting to synch. with M_i*/
        add <s_mi_in, r_tfi> to partners_list; /* s_mj_in is input partner of s_mi_in */
    else if <j, e_fj> is at wait-stack such that: e_fi <> e_fj skip outer
            else loop; /* since M_j is waiting for some other synch. event */
    else /*M_i needs to invoke M_j to find the matching partners*/
    {
        push <i, e_fi> in wait-stack;
        partners_list := generate_Mpm(M_j, s_m);
            /*Invoke and wait for secondary Mpm M_j to generate
            all possible choices of partners of synchronous event e_fi.*/
        pop <i, e_fi> from wait-stack;
        for all <s_mj_in, r_tfj> in partners_list /*containing the matched pairs*/
        {
            s_mi_out :=  (s_fi_out, f_syncij(s_mi_in, r_tfi, s_mj_in, r_tfj));
            s_mj_out :=  (s_fj_out, f_syncij(s_mj_in, r_tfj, s_mi_in, r_tfi));
            Add (s_mi_out , s_mj_out) to sync_out relation;
            compute_and_storeMp( s_mi_out, Mp_i(s_mi)); /*generate and store*/
            compute_and_storeMp( s_mj_out, Mp_i(s_mj)); /* partner states*/
            s_m_succ(i) := s_mi_out; s_m_succ(j) := s_mj_out; /*update successor vector*/
            generate_Mpm(M_i, s_m_succ);
        }/*for*/
    }/*else*/
}
visited(B(Mp_i(s_mi))) := false;
    /*All successors of s_mi are generated and so reset the flag so that the same global
state traversed by other configurations in backward conflict is not mistaken as the cut-off
vector on the basis of B(Mp_i(s_mi)) as the ancestor, which is not the case. */
}/*generate_Mpm()*/

procedure compute_and_storeMp(s_mi_out,Mp_i(s_mi))
{
/* The Mp-vector of state s_mi_out is computed using the Mp of its predecessor*/
    if Mp_i(s_mi_out) is stored already skip;
    Backtrack from s_mi_out locally through state-tree M_i as well as non-locally through
     M_j, j<>i  across synch. points until Mp_i(s_mi)(i) , i=1..n are reached respectively;
    Mp_i(s_mi_out)(i) := Maximal state w.r.t R_mi ,i=1..n among all visited/backtracked;
    store Mp_i(s_mi_out);
```

/* The non-local states of $Mp_i(s_{mi\_out})$ will be synchronous output states generated just sufficient (and necessary) to enable M_i's progress*/

}/*compute_and_storeMp()*/

The above is a simple version of the algorithm to implement the generation of Mpms and hence the sum machine.

Every Mpm is generated as a primary one with the rest (n-1) of them making only a secondary progress to satisfy the former's synchronization constraints directly or indirectly.

4.7.2 Tools for Complexity Analysis

Lemma 4.1 The *maximum* size of a *maximal configuration* is O(n*logN), where N = $|S_{mi}|$, i=1..n, is the maximum number of states of the state-tree of any Mpm M_i i=1..n, with the assumption that each *tree* is *balanced.*

Proof: This follows from the *disjointness theorem* of Chapter-2 that any configuration is a *disjoint set of n paths* of n Mpm-trees respectively. The maximum length of any path is the height of the tree which is logN for a *balanced tree*. A maximal configuration has at least one *cut-off state* forming a *leaf* node of the tree and therefore has at least one path that is of maximal length equal to logN. There are n paths in any configuration. The non-local paths are not necessarily of maximal length i.e., the height of the tree, in their respective trees. Therefore the *maximum size* of a maximal configuration is O(nlogN). ■

In the case of *unbalanced trees*, logN is to be replaced by N in the above, and through out the analyses in the sequel.

Lemma 4.2 The *size of the sum machine* ΣM is (n*N) where N = $|S_{mi}|$, the maximum size of any one Mpm-tree M_i, which is the same as the cardinality of the *Minimal prefixes*, $|Mp_i|$ as well as that of *local configurations*, $|C_i|$ which vary *monotonically* with the cardinality of the *sync* relation |*sync*| *such that:*

$|Mp_i|$ and so N is free of the *exponential factor* associated with the enumeration of global-states due to all possible *runs* and *interleavings, as allowed by the specification.*

Proof:

The first part of the lemma follows from the definitions of Mp_i and local configuration C_i as *one-to-one functions* of Mpm-states of a *state-tree* (a tree has equal number of states and events), and the fact that every Mp-vector of a state is formed by the $sync_{out}$ states for all its non-local components. So, the more the *interaction* or the *degree of coupling* among Mpms, the more is the size of $|sync|$, the more the size of $|Mp_i|$ and the corresponding size of its domain, $|S_{mi}|$.

The '*such that' part* follows from the definition of the *equivalence relations*, RMp_i at Definition 2.27 and the *locality of conflicts & Summation Lemma* discussed in Chapter-2.

Each Mp-vector is a *representative* of all the global-states formed by the asynchronous local states reachable from the non-local synchronous components of an Mp-vector thus defining its *equivalence class*, $[Mp_i]_{RMp_i}$. It is these asynchronous combinations in all *possible* manner, that give rise to *non-deterministic interleaving* in an otherwise *total-order* model. Therefore, the set of Mp-vectors being the *minimal set of global-states* to generate the rest is interleaving exponential-free, in the above sense. The Mp vectors are to global-states as *local configurations* are to *general configurations/runs*.

The above fact is associated with locality of conflicts and Summation Lemma since the *general configurations* are generated as the union of *local configurations* and hence the former need not be enumerated, corresponding to all the runs.

Thus, non-enumeration of global states is associated with non-enumeration of both the runs as well as interleavings at once.

'*As allowed by the specification*' is explained as follows:

But, due to strong *degree of coupling*, there could be a *degenerate case* when every *equivalence class* is just a *singleton set* or in other words, the *causal order* $\leq$ degenerates to a *total-order* in which case, there is no possibility of avoiding the *enumeration of interleavings* using the *representative Mp-vectors*, since the latter do not make any difference.

When the above factor is combined with *non-deterministic synchronization* of local conflicts as explained in Section 2.12.4, the scope or the room to utilize the non-enumeration of runs also increasingly disappears with the mentioned factor.

The *degenerate cases* are discussed in the sequel at few more places.

4.7.2.1 Complexity of Generator Algorithm (i)

In the worst case, the generation of every *synchronous output state* s_{mi_out} of a *primary* Mpm M_i involves the generation of *all the paths of* the *secondary* Mpms M_j, j<>i starting from $Mp_i(s_{mi\_in})$ where s_{mi_in} is the corresponding *synchronous input state*. This is necessary in order to explore all possible partners from the synchronizing Mpm, by the recursive calls to *generate_Mpm*().

The generation of a *path* of the primary Mpm M_i involves the generation of *all the paths* of a secondary Mpm M_j, each of which may involve the generation of paths of another secondary Mpm and so on until all n Mpms are exhausted. This leads to an *exponential time complexity*, $O(n(N)^n)$, where N is the maximum number of Mpm-states in a *state-tree*.

4.7.3 An Efficient Alternative of Algorithm (i)

Fortunately, the fact that there are only a *fixed number of processes* with defined *identities* and that every *synchronous event* is associated with the *known identities of the partners* can be exploited to improve the algorithm. The data-structure requiring only a polynomial amount of additional space is improvised so that the algorithm operating on this data-structure is enriched with the advantage of the *polynomial time complexity*.

While exploring the *secondary* Mpm M_j for a partner state s_{mj} for the *synchronous input state* s_{mi} of the *primary* Mpm M_i, all the *synch. input states* s'_{mj} (synchronizing with different synch. events of M_i) encountered in many different paths before reaching s_{mj} are recorded. These states s'_{mj} are potential input partners of s'_{mi} to be visited and processed later in M_i. By maintaining s'_{mj}, we avoid repeated searching of the same path(s) during the processing of different synchronous transitions.

The concept of Minimal prefix (Mp) is applied again here. When s_{mi}, a synchronous input state is reached in M_i *waiting to synchronize* with M_j, $s_{mj} = Mp_i(s_{mi})(j)$ represents the *minimal state* from which M_j has to be explored in order to find a partner for s_{mi}. During this course, any synchronous input state s'_{mj} (synchronizing on a different event with M_i or any Mpm waiting in *wait_stack* defined similar to Algorithm (i)) encountered is a *potential* *partner state* that will be reached in the future by *re-exploring* the same path(s) from the state $Mp_i(s_{mi})(j)$) or its descendents such that: $Mp_i(s'_{mi})(j) \geq Mp_i(s_{mi})(j)$). Then, s'_{mi} and s'_{mj} are partner states and *unless* s'_{mj} *is recorded now while processing* s_{mi} *with reference to the state* $Mp_i(s_{mi})(j)$, s_{mj} would be a state of re-exploration at a later point of time.

4.7.3.1 Description of Modification in Algorithm (ii)

We assume that M_j is being explored to find the *partner input state* s_{mj} of s_{mi} synchronizing on event e_{fj}. All the other *synchronous input states* s'_{mj} synchronizing on event e'_{fj} encountered are recorded in the following data structure, which is a *list* to record *multiple partners* for the same synchronous event in the case of non-deterministic synchronization:

partners_list[e'_{fj}, $Mp_i(s'_{mi})(j)$] in which s'_{mj} is stored,

where e'_{fj} denotes the *synchronous event* on which s'_{mj} is ready to synchronize with nthe state s'_{mi} of M_i ,

$Mp_i(s'_{mi})(j)$ stands for the *synchronous output state* such that :

$s'_{mj} \geq Mp_i(s'_{mi})(j) \geq Mp_i(s_{mi})(j)$, from which re-exploration may be required later in M_j.

The condition $s'_{mj} \geq Mp_i(s'_{mi})(j) \geq Mp_i(s_{mi})(j)$ defines the *range* of the synchronization points $Mp_i(s'_{mi})(j)$ depending on, state s'_{mi} at which state M_i needs to synchronize with M_j.

Since s'_{mi} is not known *a priori*, for a given e'_{fj}, there is a *partners_list* created for every synchronization point within that *range*: ($Mp_i(s_{mi})(j)$, s'_{mj}), where s_{mi}, s'_{mj} are current states of M_i and M_j, being generated.

The *explored* states are marked by an array of boolean flags. There is one flag, *explored*[$Mp_i(s_{mi})(j)$] for every non-local component of the Mp-vector of the *synchronous input states* s_{mi} of the Mpms M_i, i=1..n (that are in *wait-stack*).

As a result of the above, all the paths of every Mpm are *explored only once*, without repeating the search of any path except in the following: to traverse at most the depth of the state-tree of M_j while backtracking from the current state s'_{mj} up to $Mp_i(s_{mi})(j)$ of the *range:* ($Mp_i(s_{mi})(j)$, s'_{mj}), to create the partners_list.

The algorithm with the additional data-structure follows. The exploitation of these data-structure in the algorithm are emphasized with an underline.

4.7.4 ΣM Generator Algorithm (ii)

Data-structures:

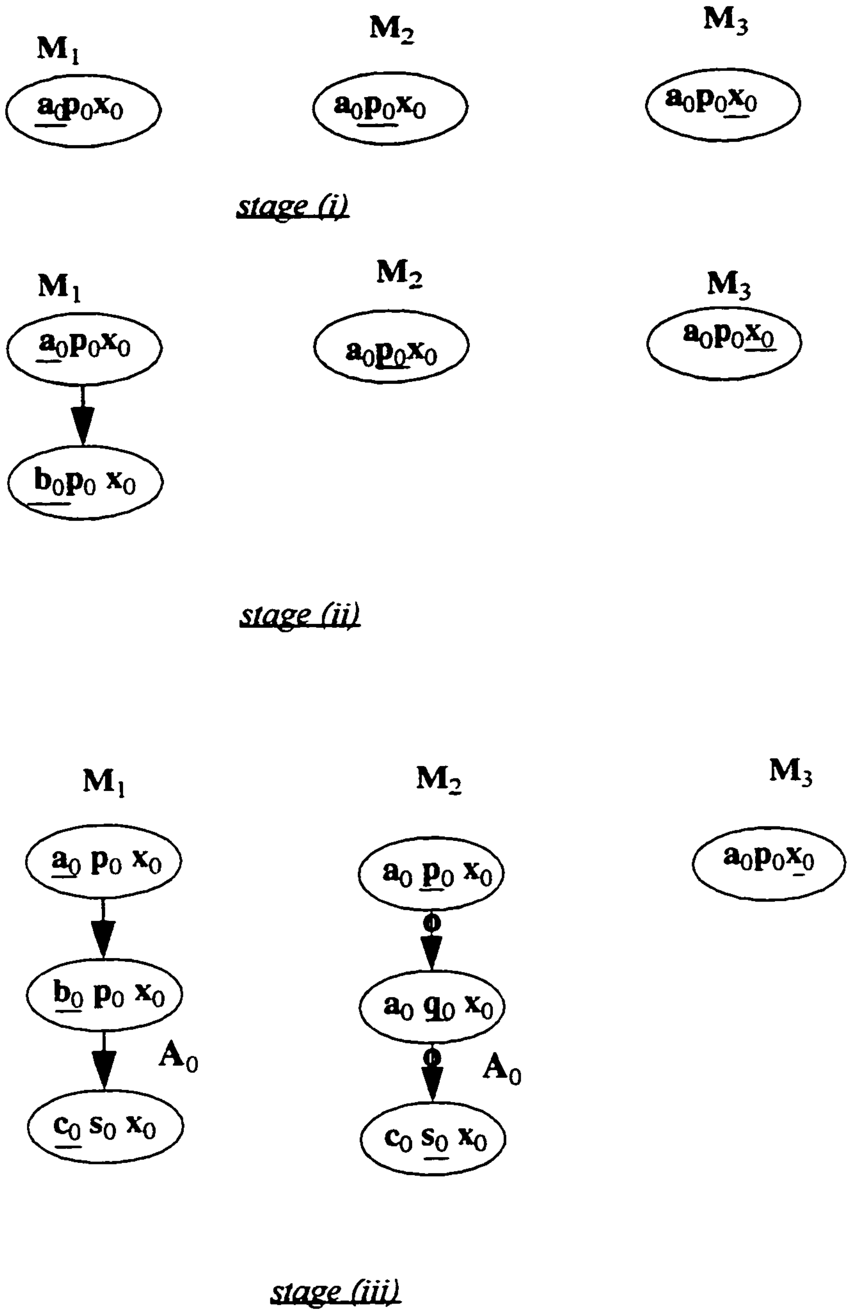
s_m	Global Mpm-state vector
s_{mi}	Mpm-state of M_i.
wait-stack	Stack of Mpm-ids waiting to synchronize with partners. The *event* waited for is not stacked as in the first version of the algorithm because, partners for *every synch. event* are going to be added to parners_list any way, for immediate or future reference.
explored[$Mp_i(s_{mi})(j)$]	Boolean flag to indicate if the paths of M_j reachable asynchronously of s_{mi} (i.e., when M_i is waiting in stack) from $s_{mj} = Mp_i(s_{mi})(j)$ are explored.
partners_list[e_{fj}, $Mp_i(s_{mi})(j)$]	List of the ordered pairs <partner input Mpm-states, Fsm-transition> reached in M_j asynchronous to state s_{mi} for the synchronous event e_{fj} while M_i is stacked.

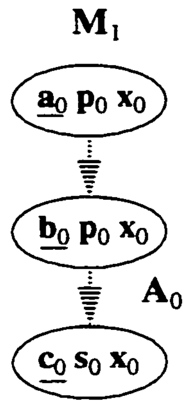
```
global ΣM, F_i, B_i, i =1..n, wait-stack;
procedure generate_all_Mpms()
{
    for i= 1..n
    {
        s_0mi := (s_0fi,f_0i(Null, r_0tfi)); B_i(s_0mi) := s_0fi; /*Now, s_0m = s_m */
        Store s_m0i, Mp_i(s_0mi) := (s_m01, s_m02,..., s_m0n);
        explored[ s_0mi] := false; /*initialize all flags and
                partner lists*/
         partners_list[ e_fi, s_0mi] := Null, for all synch. events e_fi in E_fi;
        for all s_mi stored where: explored[ s_mi] := false
                generate_Mpm(M_i, s_m ); /*only those leaf nodes of partial state-trees of
                    thus far secondary Mpms (not explored previously) are explored in this
                    algorithm*/
    }
```

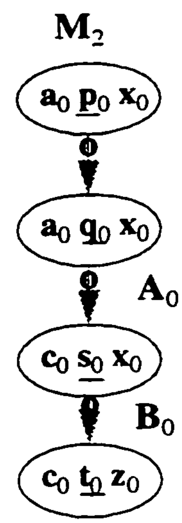
```
}/*generate_all_Mpms*/

procedure generate_Mpm(Mi, sm)
{
    if visited(B(Mpi(smi)))
    {
        cutoff(smi) := true; /*Mark smi as a cut-off state*/
        return;
    }
    visited(B(Mpi(smi))):= true;
    for (all rtfi = (sfi_in, efi, sfi_out) of Rtfi such that: Bi(smi) = sfi_in) do
    {
        sm_succ := sm;
        if (efi is local event of Fi)
        {
            smi_out := (sfi_out, fi(smi_in, rtfi) );
            Bi(smi_out) = sfi_out; /* Build the next level successor of state-tree of Mi */
            compute_and_ storeMp(smi_out, Mpi(smi));
            sm_succ(i) := smi_out;
            generate_Mpm(Mi, sm_succ);
        }
        else if (efi is synchronous between Fi and Fj)
        {
            if (j is at wait-stack ) add <smi, rtfi> to all the partners list[ efi, smi_out]
                such that: (smi ≥ smi_out ≥ Mpj(smj)(j) ) and (smi_out is synchronous
                output state) )
                /* If Mj is waiting, <smi, rtfi> is one of the partners (possibly more in case of
                non-deterministic synchronization) of the event efi = efj, possibly
                synchronizing with any of the states smi_out*/
            else /*Mj is not waiting in stack for any other Mpm */
            {
                    if not explored[ Mpi(smi)(j)]
                    {
                        Push <i> to wait-stack;
                        generate_Mpm(Mj, sm); /*explore secondary Mpm Mj*/
                        explored[ Mpi(smi)(j) := true;
                        pop <i> from wait-stack;
```

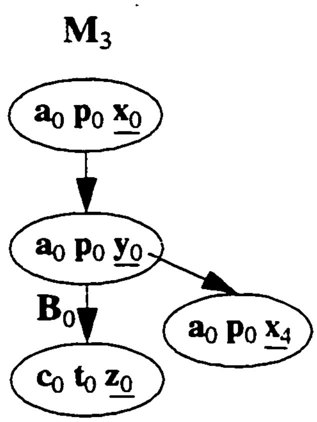
```
            }
            /*if M_j from Mp_i(s_mi)(j) is already explored, process the possibly
             multiple (due to non-determinism) partners for e_fi stored during
             exploration */
            for all <s_mj_in, r_tfj> in partners_list[ e_fj=e_fi, Mp_i(s_mi)(j)]
            {
                s_mi_out := (s_fi_out, f_syncij(s_mi_in, r_tfi, s_mj_in, r_tfj));
                s_mj_out := (s_fi_out, f_syncij(s_mj_in, r_tfj, s_mi_in, r_tfi));
                Add (s_mi_out , s_mj_out) to sync_out relation;
                /*we store and use only the sync_out relation */
                compute_and_storeMp( s_mi_out, Mp_i(s_mi));
                compute_and_storeMp( s_mj_out, Mp_j(s_mj));
                explored[s_mi_out] := explored[s_mj_out] := false; /*initialization*/
                s_m_succ(i) := s_mi_out; s_mj_succ(j) := s_mj_out;
                generate_Mpm(M_i, s_m_succ);
            }
        }/*else*/
    }/*outer else*/
  }/*for */
}/*generate_Mpm*/
```

Fig. 17 Stages of Generation of Mpms according to Generator Algorithm (ii)

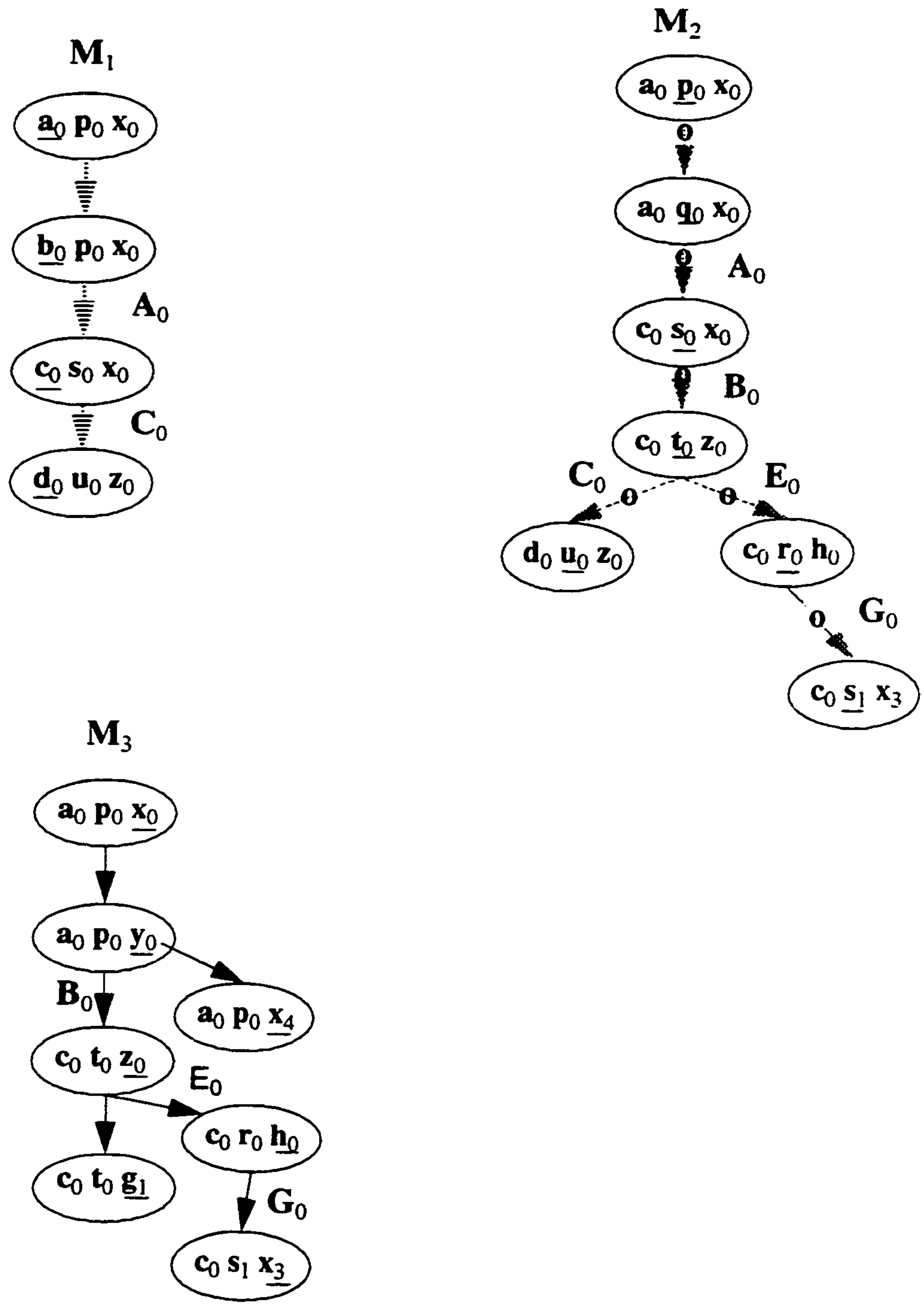



M1
a0 p0 x0
b0 p0 x0
A0
c0 s0 x0

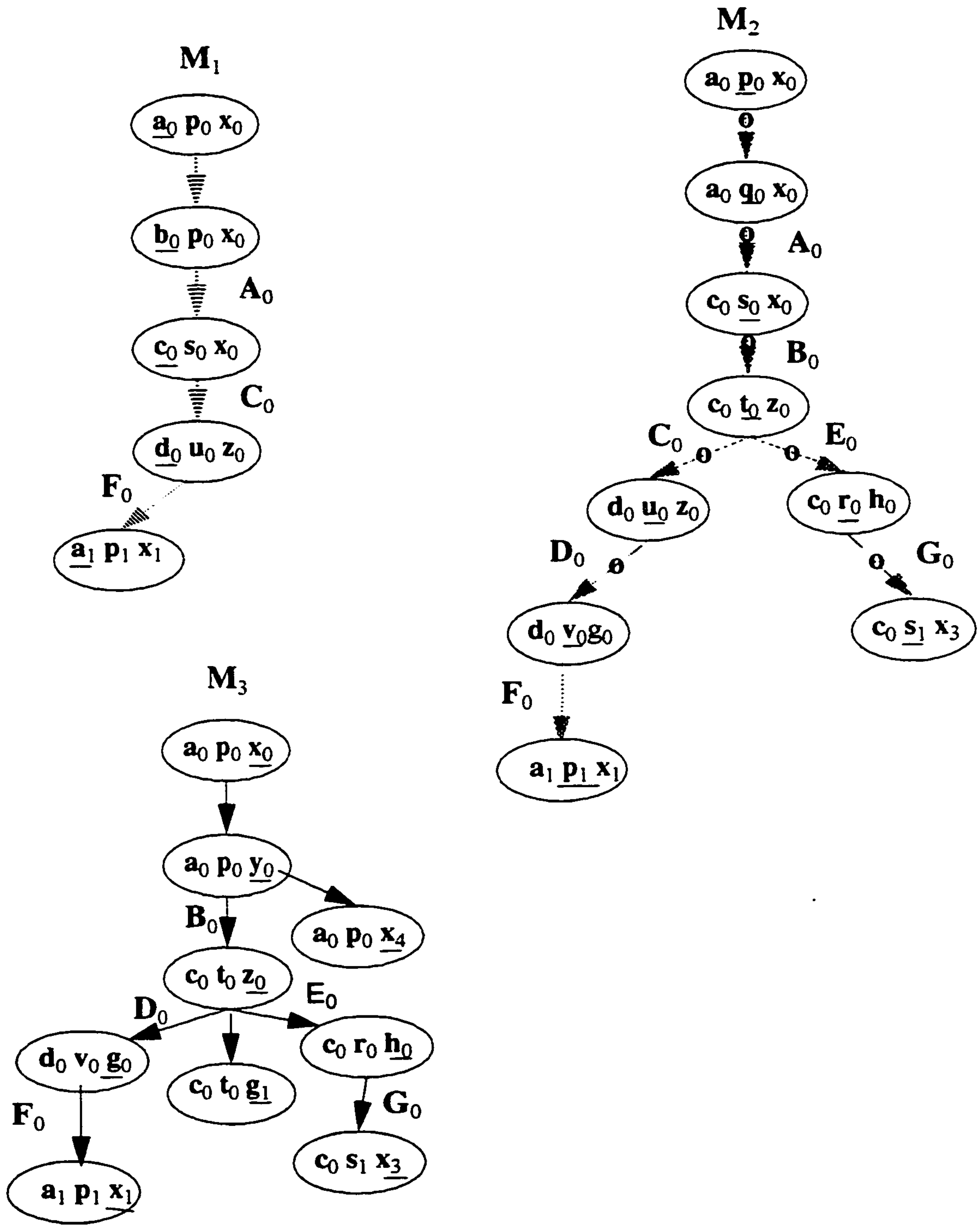

M2
a0 p0 x0
a0 q0 x0
A0
c0 s0 x0
B0
c0 t0 z0


M3
a0 p0 x0
a0 p0 y0
B0
c0 t0 z0
a0 p0 x4

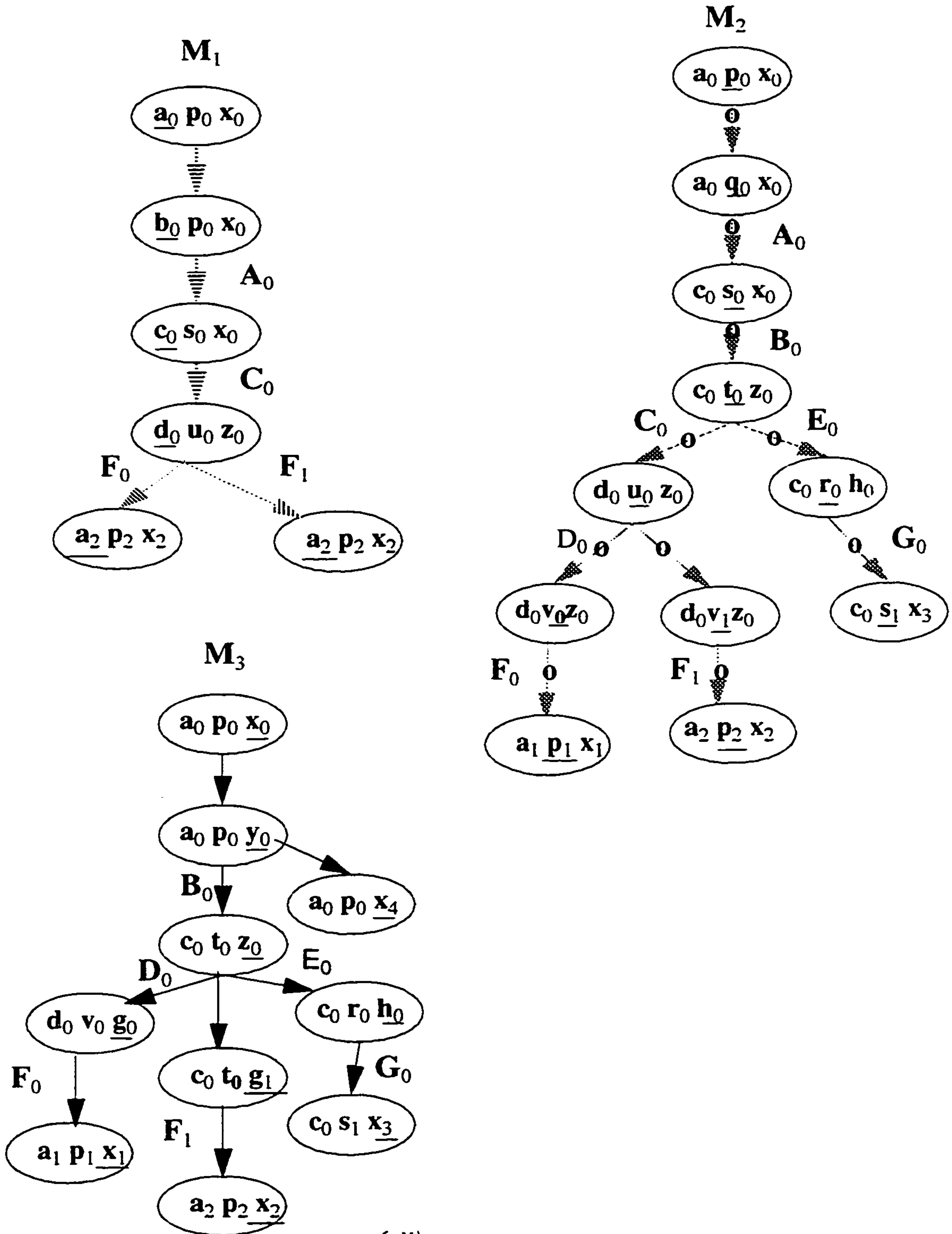
stage (iv)


M_1
$a_0\ p_0\ x_0$
$b_0\ p_0\ x_0$
A_0
$c_0\ s_0\ x_0$
C_0
$d_0\ u_0\ z_0$
M_2
$a_0\ p_0\ x_0$
$a_0\ q_0\ x_0$
A_0
$c_0\ s_0\ x_0$
B_0
$c_0\ t_0\ z_0$
C_0
E_0
$d_0\ u_0\ z_0$
$c_0\ r_0\ h_0$
G_0
$c_0\ s_1\ x_3$
M_3
$a_0\ p_0\ x_0$
$a_0\ p_0\ y_0$
B_0
$a_0\ p_0\ x_4$
$c_0\ t_0\ z_0$
E_0
$c_0\ r_0\ h_0$
$c_0\ t_0\ g_1$
G_0
$c_0\ s_1\ x_3$

stage (v)


M1
a0 p0 x0
b0 p0 x0
A0
c0 s0 x0
C0
d0 u0 z0
F0
a1 p1 x1
M2
a0 p0 x0
a0 q0 x0
A0
c0 s0 x0
B0
c0 t0 z0
C0
E0
d0 u0 z0
c0 r0 h0
D0
G0
d0 v0 g0
c0 s1 x3
F0
a1 p1 x1
M3
a0 p0 x0
a0 p0 y0
a0 p0 x4
B0
c0 t0 z0
D0
E0
d0 v0 g0
c0 t0 g1
c0 r0 h0
F0
G0
a1 p1 x1
c0 s1 x3

stage (vi)



M_1
$a_0\ p_0\ x_0$
$b_0\ p_0\ x_0$
A_0
$c_0\ s_0\ x_0$
C_0
$d_0\ u_0\ z_0$
F_0
F_1
$a_2\ p_2\ x_2$
$a_2\ p_2\ x_2$
M_2
$a_0\ p_0\ x_0$
$a_0\ q_0\ x_0$
A_0
$c_0\ s_0\ x_0$
B_0
$c_0\ t_0\ z_0$
C_0
E_0
$d_0\ u_0\ z_0$
$c_0\ r_0\ h_0$
D_0
G_0
$d_0 v_0 z_0$
$d_0 v_1 z_0$
$c_0\ s_1\ x_3$
F_0
F_1
$a_1\ p_1\ x_1$
$a_2\ p_2\ x_2$
M_3
$a_0\ p_0\ x_0$
$a_0\ p_0\ y_0$
$a_0\ p_0\ x_4$
B_0
$c_0\ t_0\ z_0$
D_0
E_0
$d_0\ v_0\ g_0$
$c_0\ t_0\ g_1$
$c_0\ r_0\ h_0$
F_0
G_0
F_1
$a_1\ p_1\ x_1$
$c_0\ s_1\ x_3$
$a_2\ p_2\ x_2$

stage (vii)

4.7.5 Steps of generation of Mpms from CFsms

The various stages of generation of CMpms from CFsms of Fig. A of *Appendix* are shown above in Fig. 17:

Stage (i): The three initial states of M_1, M_2 and M_3 are generated. The *primary Mpm* is chosen to be M_1 and so the states of M_1 are chosen to be generated first.

Stage (ii): M_1 waits for M_2 to simulate the synchronous Fsm transition (b, A, c) at b_0.

Stage (iii): In the process of exploring M_2 for a match of occurrence(s) of A, q_0 of M_2 (from p_0) is generated followed by the synchronous output states c_0 and s_0. There is no other path to be explored in M_2 for a match of an instance of A.

Stage (iv): M_1 waits to simulate the synchronous event C at c_0. M_2 in turn waits for M_3 to simulate occurrences of B. There is one path in M_3 from x_0 to x_4 which terminates in a *cut-off* and another path leading to the input partner y_0 followed by z_0, after performing B_0. M_2 reaches t_0 after performing B_0.

Stage (v): Having performed B_0, M_2 continues to match for an occurrence of C. States d_0, u_0 of M_1 and M_2 are reached respectively. The path of M_2 with E_0 followed by G_0 is generated (and hence a matching path in M_3) resulting in *cut-off states* s_1 and x_3 respectively of M_2 and M_3, in an attempt to find all matching occurrences of C in M_2 to synchronize with the primary Mpm M_1.

In the secondary progress, while looking for the match of E in M_3 to synchronize with M_2, there is a path with *synchronous event* D with M_2, but since M_2 is waiting in stack, <z_0, (z,D,g)> is put in the *partners_list*[D, z_0=$Mp_2(t_0)(3)$], to be used later on. The state g_1 of M_3 is generated following which there is a synchronous transition on event F synchronizing with M_1 and M_2 both of which are waiting on stack. The [1]*partners_list*[F, x_0=$Mp_1(c_0)(3)$][1] and *partners_list*[F, z_0][1] are created using the *range* (x_0, z_0) and <g_1, (g,F,x)> is added to both of them on account of the synchronization of F between M_3 and M_1. <g_1, (g,F,x)> is added also to *partners_list*[F, z_0=$Mp_2(t_0)(3)$][2], on account of the synchronization of F between M_3 and M_2.

---

[1] Note that, the algorithm assumes only two-way synchronization which can be in general n-way and so an additional element to denote the identity of partner_lists of M_i, i=1..n is necessary.

Stage (vi): M_1 is ready to synchronize on occurrences of F from d_0 with the partner Mpms both M_2 and M_3. u_0 is the *handle-state* which needs to synchronize on D with M_3 before simulating that on F. Hence the states v_0, g_0 of M_2 and M_3 are generated (by referring to the *partners_list*) followed by a_1, p_1, x_1 respectively of all three Mpms. While generating the synchronous event D_0 from u_0, z_0 *need not be re-explored* since the flag *explored*[z_0] is true. Instead, the *partners_list*[D,z_0] is referred to (there is only one element $<z_0, (z,D,g)>$ in this list corresponding to the single occurrence of D; in this case z_0 itself is the input partner state as well, though it is a descendent of z_0 thus saving us the re-visit of multiple states) to generate v_0 and g_0 in M_2 and M_3 respectively after performing D_0.

Stage (vii): The rest of the states of the three Mpms are generated upon performing the occurrence F_1 of F, again referring to the *partners_list*[F, x_0] and *partners_list*[F, z_0] respectively.

4.7.6 Complexity of Algorithm (ii)

Input Parameter: Though the CMpms are generated from CFsms, because the generation is recursive, the algorithm is analyzed with N, the size of the generated tree itself as a primary parameter. The worst case size of sum machine is analysed in terms of those of CFsms.

Space Complexity:

Space complexity is due to array of *partners_list* and *explored* flags:

There are as many flags as there are number of *synchronous output states*. So,

Upper bound of space complexity of *explored* flags = $|S_{mi}| = N$;

Similarly, there is a *partners_list* for every synchronous output state of an Mpm, (exploring which a set of *synchronous input state* and event pairs can be reached) corresponding to every *synchronous event* of every one of partner Mpms.

Upper bound of space complexity of *partners_list* array =

(Size of the array)* (Length of each cell) .....(i)

Size of the array := $(n*|E_{fi}|*n*N)$....(ii)

Length of each cell :=

Size of list of ordered pairs <state, event> of an Mpm := $N^*|E_{fi}|$....(iii)

Therefore, complexity of (i) := $(n^*|E_{fi}|^*n^*N) * (N^*|E_{fi}|) = n^{2*}N^{2*}|E_{fi}|^2$ from (ii) and (iii).

Even though *partners_list* is stored on the basis of *pairwise synchrony*,when more than two partner Mpms synchronize, the *size of the array* will increase by an order of n to distinguish the partners of different Mpms synchronizing on the same event. So in the above complexity , n^2 will be replaced by n^3.

The above is really a pessimistic upper bound since it assumes all the states and events of an Mpm to be synchronous.

Time Complexity:

Every state is visited *exactly once* during *its* generation , and if the state is a synchronous input state, the *partners_list* array is created for at most all the *synchronous output states* between the initial state and the generated state in its tree. *At most*, the entire depth of the tree may be back-tracked with a worst case complexity of (logN), the height of the balanced tree.

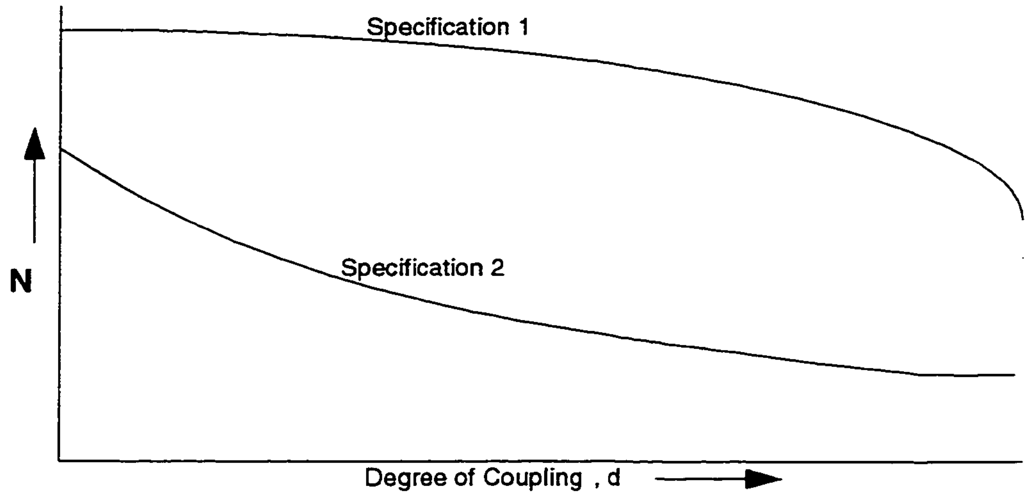
At most, every state could be a *synchronous input state* in the worst case.

Therefore, the *time complexity* is : O(nNlogN).

4.7.6.1 Size of Sum machine as a Primary Parameter

Though we started with CFsms, since the model-checker works on the sum-machine, the size of the latter becomes the primary parameter. It plays a dominant role after transformation and is comparable to the *size of the Net models* of propositional logics [35] and the *size of the unfolding* [3]. Hence it serves as a standard parameter as in other PO models. The size of the sum machine itself depends on the structure of the given specification, especially the *basic three relations* (*sequence, choice and concurrency*), and strongly on the synchronization/*degree of coupling* among the Fsms as it is given by the size of Minimal prefixes which depend on the degree of coupling. The figure below gives an idea of two different variations expected for two different specification structures of CFsms with respect to the degree of coupling. N varies directly with the degree of coupling. But the actual pattern of change depends also on other parameters viz., *choice, sequence* etc. making up the structure of the given CFsms. The illustrated ones are two possible samples.

Fig. 18 Variation of the size of sum-machine with degree of coupling for two different spec. structures



4.7.7 Size of sum machine in terms of Size of CFsms

If N_f is the size of an Fsm and $N_{fsync} \subseteq N_f$ is the size of synchronous states, $(N_{fsync})^n$ is the only *worst case exponential factor* in the size of the sum machine (number of Minimal prefix vectors) due to *all possible synchronous combinations of local states due to non-deterministic, tight synchronization of true choices.* This is to be contrasted with the *worst case size* of $(N_f)^n$ in the case of the product machine. Following are the two phenomena that hinder the applicability of the sum machine:

(i) As the *degree of coupling* increases, N_{fsync} , N increases and the number of Minimal-prefix vectors increase. Both the number of synchronous local states and the number of the participants of each synchronization (tightness) contribute to the degree of coupling. The size of *equivalence class* of every Mp-vector and so the applicability of the interleaving enumeration becomes less and less due to the above phenomenon.

(ii) When the *non-determinism in the synchronizations* increases, the non-enumeration of runs gradually cease to apply as explained in Section 2.12.4 on page 88 in Chapter-2.

As the upper limit of both the phenomena, *the equivalence class of every Mp-vector shrinks i.e., degenerates to a singleton* with $|Mp_i, i=1..n| = N = (N_{fsync})^n$ due to all possible combinations of synchronous local states forming Mp-vectors constituting all the global-states. This is due to the combination of *highest degree of coupling* that is *most non-deter-*

ministic; this is when the *non-enumerative aspect of both the interleavings and runs coincidentally drop to null*. This is the *degenerate case* when the *causality* relation $\leq$, that is in general a *partial-order degenerates to a total-order* along with an exponential number of true local choices exist due to non-deterministic synchronizations that are tight.

One consolation to the above situation is that when the asynchrony is absent and the processes progress in lock-step synchrony, the number of states will not be as explosive as in the case of high *degree of asynchrony* with no remedial measure adopted to control the state size.

If the absence of the *degenerate case* is guaranteed and the cases tending to it i.e., when the specification lends itself to utilize the advantages of sum machine, the complexity of the model-checker for the polynomial sized formulae does not seem to be NP-complete as opposed to at least one popular, rival approach of the *set methods* [44], [9], [10], [13], [17].

4.7.7.1 Non-determinism in Specification, Property Checked and the Checking Procedure

(i) Non-determinism in Specification: Since the *true choices* are restricted to local Mpms, the only source of exponential number of local states within an Mpm is the one mentioned in the last few subsections due to non-deterministic/combinatorial choices of synchronization among *true choice* states of multiple Mpms.

(ii) Non-determinism in the Property Checked: The *conjunctive normal form* of the propositional element in the CML formula is considered *non-deterministic* due to the variety of possible local propositions in each *disjunction* resulting in an exponential number of *primitive conjunctive propositions* to be checked. Consequently, there are exponential number of *global-states* whose *reachabilities* are to be checked, one corresponding to every *primitive conjunctive proposition*, upon transformation of the given *conjunctive normal form* to the *disjunctive normal form*.

(iii) Non-determinism in the Verification Procedure: Even given the absence of the above two categories of non-determinism, the verification procedure itself could be non-deterministic. This is manifested as the *non-deterministic interleavings* in the global state-graph of the traditional model-checking procedures. In the case of some of the popular modern procedures working on the *reduced-state graph*, the mentioned non-determinism is apparently got rid of, by generating only certain *representative interleavings* instead of *all*. But in the process, the *non-determinism is only shifted* to the procedure of *selecting* the representative global-states in order to maintain the *equivalence* between the original and reduced state-graphs.

The non-deterministic interleavings are alleviated in our work, by a *deterministic procedure* that traverses a set of *deterministic Mpms* whose *local states* and their global-vector labels are used to dynamically *generate or reason* about *every required* portion of the global-state graph *selectively* as demanded by the property checked, without any restriction. This way, the non-determinism is completely avoided without making any compromise or assumptions on the states required.

Given the absence of non-determinism in categories (i) and (ii), using our verification procedure, we can eliminate every source of *non-determinism* and hence conclude that the problem at hand is not in NP but in P.

4.7.8 The Easter-Egg Hunt Algorithm for Model-checking

The following algorithm is to check a *monadic third-order* $CML_{\Pi F}$ formula with a *primitive conjunctive proposition*: $g_f := A_r\, A_{Ir}\, F\, h_f$ where $h_f := (ap_{fi} \wedge ap_{fj} \wedge \ldots \wedge ap_{fk})$.

The above formula is transformed to a corresponding $CML^*_{\Sigma M}/CML_{\Pi M}$ formula g_m such that $B(g_m) = g_f$, that can be checked over ΣM as discussed in *Section* 4.2.2 at the beginning of this chapter.

Searching for the Mpm-state containing/satisfying a specific atomic proposition/conjunct in an *Mpm-tree* is likened to hunting for the *golden egg* containing the *desired Easter treat*. There are golden eggs/Mpm-states in every Mpm-tree, as many as the number of required conjuncts in total. If we could hunt the golden-eggs/Mpm-states, from all the required Mpm-trees in which the respective conjuncts belong, one each from *every interleaving* of *every run* of ΠM or equivalently every maximal configuration C_{rmax} of Σ^*M, we are considered to be *successful*.

While checking a formula in *disjunctive normal form*, the algorithm is repeated once for every disjunct of the formula g_m above. The algorithm is recursive and starts with checking for the local conjunct ap_{mi}. The current configuration is set to the *initial configuration* $C_0 = \{s_{0m1}, s_{0m2},\ldots,s_{0mn}\}$. It checks for the conjunct by traversing the local continuations of C_0 within state-tree of Mpm M_i, in a distributed, nested depth-first manner. *Chk_all_-runs*() is a recursive function, performing the recursive search.

The *distributed, nested depth-first traversal* is a pragmatic strategy as discussed in a preceding section to cover the *maximal*, finite configuration (i.e., C_{rmax} up to cut-off) corresponding to each *run* $\Pi r \subseteq \Pi M$ at a time. The *current configuration* $C \subseteq C_{rmax}$ keeps track of the *partial run* corresponding to Πr traversed thus far, the former being the subset of Σr.

If the local conjunct ap_{mi} is found, we switch to M_j as the *primary* Mpm to check through its local continuations of C for the next local conjunct ap_{mj}, using Fsv(C)(j) as the *handle* state. C here is the switching configuration. For every switching configuration reached at a local Mpm, there are multiple continuations to be traversed/checked at the next Mpm from the handle-state and so on until the last Mpm is reached and its conjunct checked. This procedure is referred to as the *distributed nesting* of the depth-first traversal, as the entire depth of one single time continuum is broken up across multiple Mpm-trees, as mentioned in a preceding section.

After *branching (in space)*, not every successor of the *handle-state* is guaranteed to configure with the *switching configuration* as explained in Section 4.4.1, with an example. The configurability of the successor of the handle-state (and its descendents) is tested by applying the *configurability theorem*, by the function *configurable*() of the algorithm.

During the search of any Mpm if it is either a *cut-off* state or in *asynchronous, non-local conflict* with a cut-off state (allowing the possibility of *unfair runs*), the required conjunct can not be found. This follows from the *unfairness theorem*.

Otherwise, the recursive procedure is continued by traversing all possible successors until:

(i) *cutoff* state is encountered which means failure of satisfiability of the formula, according to the *cyclicity-theorem*, or

(ii) there is no more continuation possible which again means failure, or

(iii) the local conjunct is found and switching (branching in space) to next Mpm made.

This procedure is continued until all conjuncts and hence the conjunction is satisfied which means success, the cases of failure being handled as in (i) and (ii) above.

Input of Model-checker: F_i, i=1..n and g_f a $CML_{\Pi F}$ formula.

Output of Model-checker: The result of satisfiability of g_f in the model $CML_{\Pi F}$: *true* or *false*.

It is assumed that ΣM corresponding to F_i,i=1..n has been generated as discussed and g_m such that $B(g_m)=g_f$ is a $CML^*_{\Sigma M}$/ $CML_{\Pi M}$ formula .

Data-structures for model-checker:

F_i, i=1..n	State-graph of Fsm F_i, i=1..n
M_i , i = 1..n	State-tree of Mpm M_i, i=1..n
B_i	The mapping of states, events of M_i *onto* F_i
B	The mapping of states, events of ΠM *onto* ΠF.
g_m	$CML^*_{\Sigma M}$ and $CML_{\Pi M}$ formula such that $B(g_m) = g_f$
h_m	A *primitive conjunctive proposition* of g_m.
Mp_i, i =1..n	Set of n Minimal prefix functions, i =1..n.
C_0	initial Configuration of Σ^*M
$C_i(s_{mi})$	Local Configuration of Mpm-state s_{mi}
C	Current Configuration
s_m = Fsv(C)	Final state vector of C
s_{mi} *anl_conf* s_{mj}	boolean binary operator indicating if s_{mi} is in *asynchronous local conflict* with s_{mj}.
local_successors	List of local Mpm-state successors of s_m in M_i (s_{mi}) that need to be checked.
failed_successors	List of all Mpm-state successors of s_m that need to be continued again when s_m does not satisfy g_m for *all interleavings*.

global C, s_m, h_m, ΣM;

function CML_check($U_{i=1..n}F_i$, g_f) /*The main model-checker function */

{

g_m := $A_\tau\ A_{Ir}\ F\ h_m$ where $h_m := (ap_{mk} \wedge ap_{mk+1} \wedge \ldots \wedge ap_{mk+j})$

such that: $B(g_m) = g_f$, $B_i(h_{mi}) = h_{fi}$ and $h_{mi} = ap_{mi}$, i= k..(k+j), j≤ n ;

$s_m := s_{0m} := (s_{0m1}, s_{0m2}, \ldots, s_{0mn})$; $C := C_0 := \{s_{0mi} | i=1..n\}$; /*initialization of C and

Fsv(C) = s_m*/

for *some conjunct* h_{mi} in h_m

return(chk_all_runs(M_i, s_{mi}, h_{mi})); /*which invokes checking of all conjuncts */

}/*CML_check()*/

function Chk_all_runs(M_i, s_{mi}, h_{mi})

{

If (s_{mi} is a *cut-off state*) return(false);

if (s_{mi} *anl_conf* s_{mj} s.t: s_{mj} is *cut-off state*) return(false); /* recorded during

generation of Mpms*/

If ($p_{mi}(s_{mi})$ =h_{mi} such that: $B_i(h_{mi}) = h_{fi}$) /*local conjunct is found*/

{

if ($p_{mj}(s_{mj})$ <> h_{mj} for some j <> i)

{

success := chk_all_runs(M_j, s_{mj}, h_{mj}); /*nested search of next Mpm */

if (not success) return(false);

}

/* s_m of current configuration/run satisfying h_m in one interleaving is reached

which has to be checked for truth for *all* interleavings before going to next

run*/

failed_successors := chk_all_interleavings(s_m);

if (failed_successors is Null) return(true);

```
    }
    else /* local conjunct is not yet reached and so searched among local successors
        of s_mi that are configurable with C*/
    {
            if ((local_successors := configurable(C, s_mi)) is Null) return(false);
            for all (s_mi_nxt in {local_successors U failed_successors} )
            /* all successors take care of all runs*/
            {
                C := C U C_i(s_mi_nxt); s_m := Fsv(C); /*updating C and s_m to their
                    successors*/
                if (chk_all_runs(M_i, s_mi_nxt, h_mi) is false) return(false); /*recursive
                    call to its successor in the current Mpm*/
            }
            return(true);
    }
}/*chk_all_runs()*/
function configurable(C, s_mi)
{
    local_successors := Null;
    for all s_mi_nxt = successor(s_mi) in M_i do
    if (is_confign(C_i(s_mi_nxt) , C ) /*Checks if (C_i(s_mi_nxt) U C is a configuration */
        add s_mi_nxt to local_successors;
        return(local_successors);
}/* configurable()*/
function is_confign(C_i(s_mi) , C)
{
    for k= 1..n do
    {
```

```
        if (not Mp_i(s_mi)(k) is reachable from Fsv_k(C), k =1..n or vice versa) in state-tree of
            M_k /*Follows by configurability theorem of Chapter-2; incurs visiting at most
                (nlogN ) states.*/
              return(false);
    }
    return(true);
} /*is_confign()*/
function Chk_all_interleavings(s_m) /*checks all the interleavings of a run */
{
    failed_successors := Null;
    Repeat for every pair of successors s_mi_nxt, s_mj_nxt
    {
        for every pair s_mi, s_mj such that: s_mi_nxt, s_mj_nxt configure with C
        {
                if (not (must-co-wait(s_mi, s_mj, s_mi_nxt, s_mj_nxt)))
                    add s_mi_nxt, s_mj_nxt to failed-successors;
        }/*for loop exhaustive of all pairs of i and j */
    }/*repeat loop exhausting all successors of each pair of s_mi, s_mj */
    return(failed-successors);
}/*Chk_all_interleavings()*/
function must-co-wait(s_mi, s_mj, s_mi_nxt, s_mj_nxt)
/*implement the checking of 'must-co-wait' condition among Mpm-states expressed in
the inference rule interleaving theorem stated as of Theorem 3.2 Chapter-3.*/
{
    if is_dependent(s_mi, s_mj_nxt) ∧ is_dependent(s_mj, s_mi_nxt))
        return(true);
    else return(false);
}/*must-co-wait()*/
```

```
function is_dependent(s_mj, s_mi)
{
    /* This function checks if s_mj ≤ s_mi with Mpm-trees M_i, M_j visiting at most 2*logN
     states*/
    Backtrack s_mi along its unique path of predecessors in M_i until s_mi_sync
        such that: (s_mi_sync sync_out s_mj_sync) is reached;
    Backtrack s_mj_sync along its unique path of predecessors in M_j
        until s_mj or s_0mj is reached;
    If (s_mj is not reachable) return(false); /*i.e., when s_0mj is reached */
    return(true);
} /*is_dependent()*/
```

4.7.9 Analysis of the Model-checker Algorithm

4.7.9.1 Upper bound of Complexity

The maximum number of *continuations traversed* or synonymously the *states visited* per Mpm-tree is the same as the maximum number of occurrences of an Fsm state in the corresponding Mpm. So, if there are multiple conjuncts in a conjunction of *local atomic propositions*, (the number of such conjuncts being at most n) the complexity of *evaluating the satisfiability of such a primitive conjunction* is corresponds to that of *deciding the reachability of a global state* and hence is focussed below.

Parameters of Complexity:

Primary Parameters:

- N – the maximum number of Mpm-states per any Mpm-tree, i=1..n.
- n – Number of Mpms, which is also the maximum number of conjuncts as well as the *maximum depth of nesting* of the traversal.
- d - Degree of *coupling/interaction/synchrony* among Mpms contributed by the number of synchronous transitions and their tightness.

Secondary Parameters:

There are two secondary parameters m and k that depend on d; m varies *directly* with d and k varies *inversely* with d as explained below.

- m-- the *maximum number of successful local runs/configurations satisfying* the *local proposition*, ap_{fi}, which is the *number of occurrences of any* one Fsm-state, s_{fi}, i=1..n, *in conflict*. If d is large, then the *size* of |*sync*| and the size of $|\Sigma Mp_i|$ (and N) are large. This implies that *the number of occurrences* of given s_{fi} and m are also large. Similarly, if the value of d is small, the other extreme can be argued: i.e., the less the value of d, so will be that of m. The *upper bound of m* is N but in general, m<<N as it represents occurrences of only one Fsm-state. *Lower bound of m* is 1 (0 is possible as well but excluded).
- k -- the *degree of distributed nesting of local configurations*: This parameter is a *variable* as opposed to the *depth of nesting* which is a constant. Its *maximum* value is equal to the maximum number of conjuncts, which is *n* and its *minimum* value is 1. The value of k is related to the *degree of coupling* d. If *each one of the m successful* local configurations of one Mpm has *all the m* local configurations of the next Mpm as their continuations for every pair of current and next Mpms, it will imply that the *asynchrony* among Mpms is *maximum* (i.e., d is *minimum*). At the other extreme, when each one of the m local configurations has *exactly one* distinct successful continuation in the next Mpm, it will imply that the *asynchrony* among Mpms is *minimum*, which corresponds to *d* at its *maximum*. Thus we establish the maximum and minimum values of k as n and 1 respectively.

The Reasoning of the Upper Bound :

From every Mpm-tree, up to m *successful local configurations in conflict* correspond to m local runs satisfying the *atomic proposition*. With the *nesting degree* k varying from 1 to n *for every local configuration*, there could be up to m successful local continuations in the next Mpm-tree. Accounting for all n Mpms we get a total of m^k local configurations, that is *apparently exponential*.

The term '*apparently*' is explained as follows:

When the degree of coupling d is *high*, the cardinality of *sync* relation, |*sync*| is high which means that $|\Sigma Mp_i|$ and m are also *high*, the *upper bound* of which is N. When that happens, each local configuration has a *unique successful continuation* in the next Mpm, making the *nesting degree a minimum, k=1*. This means m^k tends to become N^1.

On the other extreme, when d is lowest, |*sync*| is lowest, and due to large asynchrony, *all the m successful local configurations* of one Mpm are reachable totally *asynchronous* of the *each of m successful local configurations* of the previous Mpm leading to the *maximum degree of nesting, k = n*. Interestingly, the maximum number of occurrences of an Fsm-state m is almost *unity* in this case since $|\Sigma Mp_i|$ is at a *minimum* and so m^k reduces to 1^n in this case. That is,

$$\lim_{d \to \min} (m^k) = 1^n \text{ and,}$$

$$\lim_{d \to \max} (m^k) = N^1$$

The maximum value of the function m^k occurs for an *optimum value of d*, when k is between 1 and n and m is between N and 1. As k is higher, m is lower, thus *nullifying each other and hence diffusing the explosive effect of the factor, m^k*. Therefore, even though it is an exponential factor, it droops after an optimal peak where m <<N (since m in general is the number of occurrences of one Fsm-state only) and k < n.

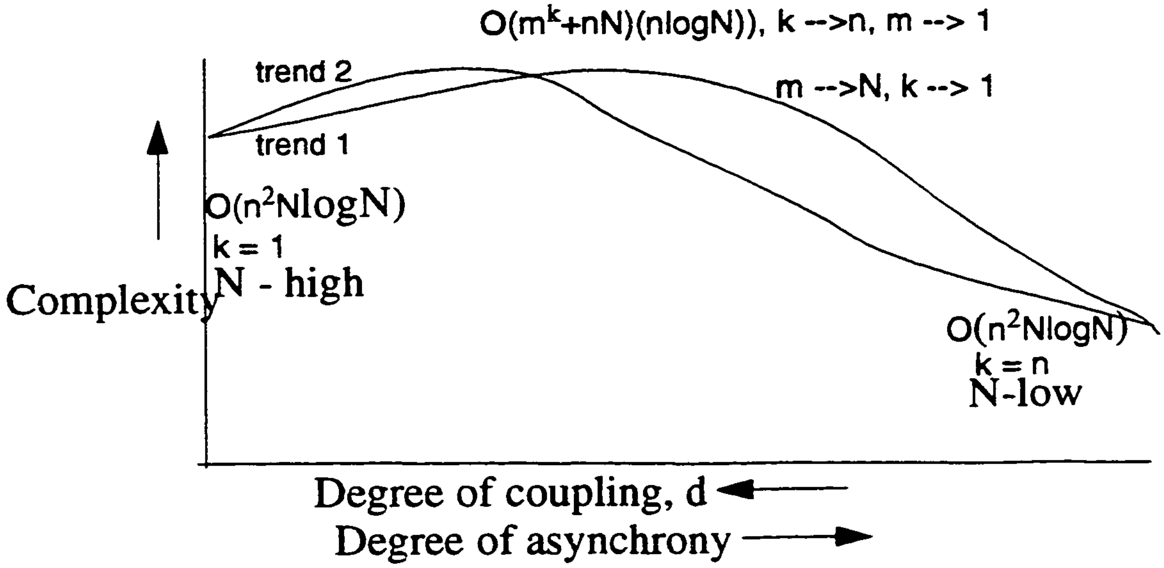
Since the size of a *maximal configuration* is O(nlogN) (assuming the balanced Mpm-tree) according to Lemma 4.1, as traversed by *configurable*() function, the complexity of traversing all *successful* runs is: m^k(nlogN).

Including also the *failed* local configurations of each Mpm, we account for all the local configurations of all n Mpms, i.e., (nN*nlogN) in the complexity.

Thus the total complexity of *check_all_runs*() is: (nN + m^k)nlogN.

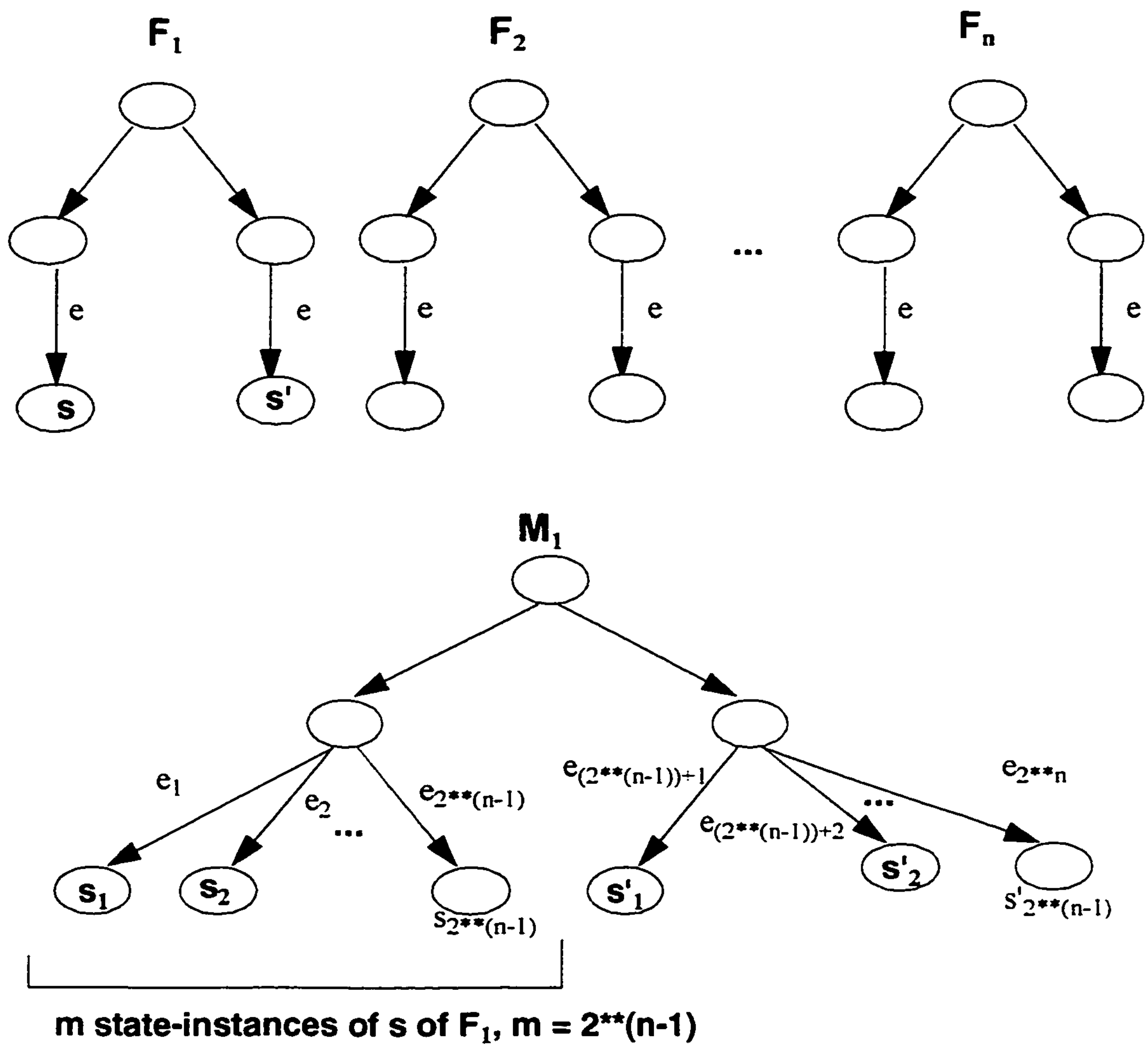
The figure below shows the trend of the upper bound complexity of checking all runs with the degree of coupling. There are two different trends (labeled as trend 1 and trend 2) corresponding to two different primitive conjunctions checked (which changes m and k) in a given specification. The complexity is expressed in terms of *N* *which itself varies with the* *degree of coupling as shown in Fig. 18.* That is s why for the same complexity in terms of N, the actual value is high when d is high, and is low when d is low. When m and k are optimal, the factor (m **k) dominates and increases the complexity beyond the extreme cases when the above factor loses its effect.

Fig. 19 Variation of complexity of chk_all_runs() with the degree of coupling



4.7.9.2 Size of the parameter m and Non-deterministic Synchronization

Fig. 20 Non-deterministic Synchronization and Enumeration of Runs (*induced local conflicts*)



As illustrated in the figure above, that is reproduced from Chapter-2 with additional state information, we note that the states s_1 through $s_{2^{**}(n-1)}$ in M_1 are all the instances of the single Fsm-state s of Fsm F_1 and the corresponding instances of the local predicate of s. The growth of m with the degree of coupling is caused mainly by this phenomenon of non-deterministic synchronization culminating in m tending to the order of N. Fortunately the size of k tends to 1 then although N itself has an exponential size which is caused by all possible synchronous combinations of local states due to the phenomenon discussed. When the degree of coupling is low, k may be high but the size of m and N are not exponential. Depending on the extent of non-deterministic synchronization (m ** k) may dominate instead of N in this case.

Thus we see that either the growth of N or domination of (m**k) is a result of the necessary enumeration of local runs inherited from the specification. By carefully avoiding the combination of non-determinism and tightness of synchronous transitions wherever possible the above growth can be controlled.

4.7.9.3 Upper bound Complexity of chk_all_interleavings()

This is contributed by the *is_dependent*() function .

There are at most ($n^C 2$) comparisons/calls for every pair of states of s_m, the final state of the configuration; each comparison makes *at most* O(logN) visits (height of the *balanced tree*) while backtracking the concerned states in their paths. Thus, the *upper bound of complexity* of checking all interleavings within each run is:

$O(n^2 * \log N)$ since $n^C 2 = O(n^2)$.

The *over all complexity* is given by the *product* of:

complexity of *chk_all_runs*() and of *chk_all_interleavings*(): $O(Nn^3\log^2 N)$.

4.7.9.4 Total Upper bound Complexity

The total upper bound complexity of the model checker algorithm to check *all interleavings of every run* is given by the *product* of the complexity of *chk_all_interleavings*(): $O(n^2 \log N)$ and that of *chk_all_runs*() of last section and the product is:

$(m^k + nN)\ (n^3 \log^2 N)$.

When the Mpm-trees are *not balanced*, logN is replaced by N in the above figure.

We see that the upper bound complexity is *polynomial* both in *the size of sum machine N* and n, considering the fact that *m and k diffuse each other out*.

If there are q disjuncts in the *disjunctive normal form* of the proposition checked in the formula, the complexity is only q times the above figure.

The important result here is that the complexity is *neither exponential in the size* of ΣM nor in the *length of the formula* (conjunctions and disjunctions) as opposed to the conventional *propositional logics* over Nets [35].

4.7.9.5 Worst case Complexity

The above claim, $(m^k+nN)\ (n^3\log^2 N)$ of the upper bound complexity does not necessitate any assumption about the *specification structure*. The size of N itself can be exponential. Typically, when m tends to N (and k to 1) it is also the case that N is exponential equal to $(N_{fsync})^n$ due to the combination of *non-deterministic and tight coupling* among Mpms, resulting in exponential *induced local conflicts* within an Mpm. This issue was explained in a couple of contexts in Section 4.7.7 and in Section 4.7.10.

If we assume the specification to be *free of non-deterministic synchronization*, then we can realize the non-enumerative advantages of the sum machine usefully without any exponential complexity *absolutely*, for the formulae in *disjunctive normal form*.

Conclusive note of Model-checker and its Complexity:

The contribution of model-checker is that, only the successful, specific local continuations depending on the formula checked are considered for traversal in practice across one Mpm to the next, which in practice droops down more and more as the degree of nesting increases.

Searching for a global-state across a single global state graph exhaustively is a random approach and hence *nondeterministic* as opposed to the *deterministic, goal-oriented localized search* across individual local state graphs that are linked by the Minimal prefix vectors. The *global-state* and the corresponding *global proposition* are constructed by *concurrent holding* of specific local Mpm-states and the *conjunction* of their atomic propositions respectively during the *local search* of Mpm-trees as contrasted to *global search* of enumerated global-states checking for the *satisfaction* of the whole conjunction directly, at once.

4.7.10 Examples

Model-checker with Universal Versus Existential interleaving Operator:

The model-checker corresponding to *existential interleaving operator* checks for one interleaving by traversing the configuration as guided by the proposition. It is equivalent to checking for *pos-co-wait* condition among Mpm-states, as stated in *Axiom 3.4* of Chapter-3. The algorithm does not invoke the *chk_all_interleavings*() function at all within the *chk_all_runs*(), which requires a minimal simplification of the model-checker algorithm presented.

The model-checker with *universal interleaving operator* is the one presented in the last section. It is the same as the one with existential operator except that the conjunction checked is quantified by the ***universal interleaving operator*** (A_{Ir}) in place of E_{Ir}; in which case, once a conjunction is satisfied by reaching the required configuration C by some arbitrary interleaving, we check if the individual components of the conjunction *must/necessarily co-wait* according to the *interleaving theorem* of Chapter-3. If so, it implies that the conjunction is true for all interleavings. Following section explains how to implement the checking of '*must-co-wait*' condition among Mpm-states expressed in the *inference rule interleaving theorem* as stated at Theorem 3.2 of Chapter-3. So, there is additional function *chk_all_interleavings*() , to implement the following in this algorithm.

Example 4.7 The algorithm for the formula $s_{m0} \models E_r\, A_{Ir}\, F\, (p_{m1}(d_0) \wedge p_{m2}(v_1) \wedge p_{m3}(g_1))$ works the same way as for existential operator, till the configuration C with Fsv(C) = (d_0, v_1, g_1) is reached.

After that, we check if every pair (and so their *atomic propositions*) of the components of Fsv(C) = (d_0, v_1, g_1) reached by some arbitrary interleaving *must-co-wait*.

Considering d_0 and v_1, $d_0 < v_1$ and so (d_0 *must-wait-for* v_1) since d_0 is present in the local configuration of v_1. Next thing to be checked is whether *succ*(d_0) > v_1, where *succ*(d_0) is a *successor* of d_0.

This is done by : back-tracking v_1 to u_0 which forms a synchronization point with d_0 itself. (In general, the synchronous state has to be back-tracked in M_i to check if the desired state is reachable). Therefore, $p_{m1}(d_0)$ and $p_{m2}(v_1)$ *must co-wait*.

Similarly, v_1 and g_1 can be shown to *wait for each other* and so do g_1 and d_0.

Thus the conjunction is satisfied by all interleavings of the run containing C in its conflict-free sum-machine. Hence the formula.

Example 4.8 The formula, $s_{0f} \models A_r E_{Ir} F (p_{f1}(d) \wedge p_{f2}(v) \wedge p_{f3}(g))$ can not be true as follows:

When M_1 is traversed, at state d_0 we find that (d_0 *anl-conf* x_3) where x_3 is a cut-off state. Therefore by *unfairness theorem*, we can conclude that there is a run where d_0 or any of its local descendents (with respect to $\leq$) in M_1 will never be reached in the future. Thus there will not be an Mpm-state mapping onto d in F_1 in the future and so the conjunction will not be satisfied in the future as well.

Hence the result.

Example 4.9 Let us walk through the checking of the formula,

$s_{f0} \models E_r E_{Ir} F (p_{f1}(d) \wedge p_{f2}(v) \wedge p_{f3}(g))$:

The checking of this formula in ΠF domain consists in finding the satisfiability of the following formula in ΠM domain:

$s_{0m} \models E_r E_{Ir} F(p_{m1}(d_{num1}) \wedge p_{m2}(v_{num2}) \wedge p_{m3}(g_{num2}))$ such that:

$B_1(d_{num1}) = d$, $B_2(v_{num2}) = v$, $B_3(g_{num3}) = g$.

The initial configuration is set to $C_0 = \{a_0, p_0, x_0\}$.We traverse the state-tree of M_1 first until we reach state d_0 visiting and traversing the local configurations of all the states enroute. When we reach d_0 which maps onto Fsm-state d, $C = C_1(d_0)$ is the *switching configuration* with $Fsv(C) = Mp_1(d_0) = (d_0, u_0, z_0)$. We switch to M_2 and traverse from the *handle-state* u_0, using the *Mp-vector label* of d_0. The successor v_1 of u_0 *configures* with the switching configuration. The current configuration C now is the local configuration of v_1 with $Fsv(C) = (d_0, v_1, z_0)$. Since the local conjunct of M_2 is satisfied, we now switch to M_3. The successor of z_0 viz., g_1 configures with the switching configuration to give rise to its new current configuration C, with $Fsv(C) = (d_0, v_1, g_1)$.

Therefore, $s_{0m} \models E_r E_{Ir} F (ap_{d0} \wedge ap_{v1} \wedge ap_{g1})$ and

mapping it from ΠM *onto* ΠF through B, we get:

$s_{0f} \models E_r E_{Ir} F (ap_d \wedge ap_v \wedge ap_g)$

Example 4.10 As an alternative example, consider the formula $s_{0m} \models E_r E_{Ir} F (ap_t \wedge ap_g)$:

M_2 is traversed until t_0 is reached with $Mp_2(t_0) = (c_0, t_0, z_0)$; Using $z_0 = Mp_2(t_0)(3)$ as the handle-state, branching-off to M_3 is made. Suppose g_0 is visited in M_3, arbitrarily chosen as the first descendent of z_0. Now the current configuration C is such that, Fsv(C) = (d_0, v_0, g_0). Interestingly, this vector satisfies ap_{g0} but not an instance of ap_t. So, M_2 is traversed again from v_0, the handle-state. But its only descendent p_1 is a cut-off state meaning failure of the local conjunct and hence the formula in the current run.

g_1 is another successor of z_0 which upon visitation, gives rise to C with Fsv(C) = (c_0, t_0, g_1) satisfying $(ap_{t0} \wedge ap_{g1})$, and hence the given formula, since this conjunction holds in the future of a run/configuration just traversed, by an arbitrary interleaving.

Example 4.11 Let us consider the formula, $s_{0m} \models A_r E_{lr} F(ap_v \wedge ap_g)$.

In this example, M_2 is traversed till the state v_1 is reached; C such that Fsv(C) = (d_0,v_1,z_0) is the switching configuration. Using z_0 as the handle state, M_3 is traversed from z_0. g_1 configures with C but not other successors g_0 and h_0 of z_0 because: (g_0 *conf* v_1) and (h_0 *conf* v_1), inherited from (v_1 $conf_2$ v_0) where (v_0=g_0) and (v_1 *conf* r_0) where ($r_0 = h_0$) respectively.

Thus we get C' as the only local continuation of C in M_3, such that Fsv(C') = (d_0,v_1,g_1) satisfying $ap_{v1} \wedge ap_{g1}$.

C" = (d_0,v_0,g_0) is another continuation in M_2 satisfying $(ap_{v0} \wedge ap_{g0})$ already.

C''' = (c_0,s_1,x_3) is yet another continuation which is a cut-off vector corresponding to the basis vector, (c_0, s_0, x_0) as explained in Chapter-2. Since a cut-off vector is reached before finding the targeted conjunct (and hence the conjunction), the run corresponding to C''' does not satisfy the formula, by the *cyclicity theorem*.

Hence the given formula is false.

4.7.11 Sketch of Proof of Correctness of Model-checker

The proofs essentially follow from many definitions, theorems of Chapter-2 and particularly the *axioms and inference rule* of Chapter-3.

Some of the salient points that contribute to the sketch of the proof, using which the proof can be phrased without any lack of rigour, are listed below:

- There is a *one-to-one correspondence* between configurations and reachable Mpm-state vectors (as formers' Fsvs), and every *general configuration* and its Fsv can be reached by the traversal of *local configurations* by ΠM *Generator Theorem* viz., Theorem 2.8 that follows from the *Summation Lemma* of Chapter-2.

- Every *reachable state-vector* also corresponds *one-to-one*, to a *primitive conjunction* of atomic propositions of the component local Mpm-states, by the definition of the input bijection p_{fi} and the generated bijection p_{mi}, by the *Mpm-propositions* stated as *Definition 3.1* of Chapter-3.

- By *Equivalence Theorem III* stated at Theorem 3.2 of Chapter-3, the given $CML_{\Pi F}$ formula (with respect to ΠF) can be transformed to an equivalent $CML_{\Pi M}$ formula in ΠM domain and checked on Σ^*M domain, as a $CML^*_{\Sigma M}$ formula.

- The recursive processing of successive states ensure the *depth-first search*, to cover all the *required* configurations corresponding to all required runs, *one* configuration *(run) at a time*.

- The notion of Mp ensures *branching of space* from one Mpm to the other and continue the traversal of the latter's states, maintaining the *current branch of time*, using the notion of *handle-state*.

- The Axiom 3.2 of Chapter-3 supports the implementation of existential interleaving operator as (*pos-co-wait*) or *concurrency* among states of the final state vector reached by a configuration.

- The *cyclicity theorem* and *unfairness theorem* ensure termination of the algorithms due to failure, as proved and explained in Section 4.4.5 and Section 4.5.1 respectively.

- The purpose of the *chk_all_interleavings*() function is to check if every pair of components s_{mi}, s_{mj} of s_m= Fsv(C) where C is the destination configuration are related by the *must-co-wait* operator. The proof-sketch follows from the inference rule at Rule 3.2 and *interleaving theorem* (of Rule 3.4) Chapter-3.

▧

4.8 Complexity Theorem II

The overall complexity of verification of the properties of a given CFsm system includes the *one time generation* of the *sum machine* ΣM, followed by its traversal for model-checking.

Theorem 4.4 The *language* restricted to *monadic third-order formulae* with propositions in *disjunctive normal form* of the *primitive conjunctions* of the logic $\text{CML}_{\Pi\text{F}}$ is *recognizable* by the *sum machine* ΣM, with a complexity ,

(i) which is neither *exponential* in the *size* of ΣM nor in the *length of the formulae* without any assumption made.

(ii) which is *non-exponential absolutely,* assuming the *absence of non-deterministic synchronizations in the specification.*

(iii) which is non-exponential again for *any* CML formulae, *assuming the absence of non-determinism in the property checked* as well as in the *specification* mentioned in (ii).

Proof: The proof of (i) follows conceptually from the *complexity theorem I* of Chapter-2 and concretely from the *model-checker algorithm* along with their proof sketch of correctness in the last section above.

Using ΣM, the properties of ΠF expressed as CML which include both *safety* and *liveness* can be verified. The algorithm covers one of the most complex formulae as a representative, like the approach of [1], with a complexity that is *neither exponential in N,* nor in the *length of conjunction n , nor again the disjunction, q.*

Proof of (ii) : The *alleviation of non-deterministic specification* of synchronization guarantees that N, and so the *size of the sum machine* ΣM will not be *exponential,* since it is the combination of tight coupling and non-determinism that leads to combinatorially *induced local conflicts*, as explained in Section 2.12.4 on page 88 of Chapter-2 and Section 4.7.10.

Proof of (iii) : With (ii) guaranteed and with the *absence of non-determinism* in the property checked typically contributed by the *conjunctive normal form,* we avoid the exponential number of global-states whose reachabilities are to be checked.

As explained in Section 4.7.7.1, with all the three sources of *non-determinism* removed, the problem is in P (Polynomial) rather than in NP and hence the result follows.

■

Discussion of the Theorem:

Depending on the *specification structure*, the size N varies as discussed in Section 4.7.7 on page 187. If the *degree of coupling* is not high and the *non-deterministic synchronization* is minimal in the given input system of CFsms, there is ample room for the exploitation of non-enumeration of *interleavings* and *runs* in which cases, the size of N will not be exponential in the size of N_f.

But when $N = (N_f = N_{fsync})^n$ (due to the combination of tight and non-deterministic coupling), we can only be rest assured that the stringent synchronous paths naturally are regulated by the high degree of coupling in a self-stabilizing manner and the resulting N will not be as high as the size of $(N_f)^n$ when there is large degree of asynchrony when $N_f >> N_{fsync}$.

Thus the algorithm is not inherently NP-complete even though NP-cases are not ruled out depending on the *specification structure* and the *property checked* as discussed in Section 4.7.7.1 categorically. This result is better than the one where the method may fail to apply for some *degenerate cases* as mentioned and in addition, the complexity is exponential regardless of the specification structure and the property checked. The contrast in this regard with a popular, existing approach will be made in Chapter-5.

The *primitive conjunctions* correspond to *global-state reachability*; the latter corresponds to what is aimed by formulae of full logics of CTL [1] etc. The CML formulae with *until since* operators can be reduced to the ones with primitive conjunctions as shown by the axioms of Chapter-3. Even all the other formulae allowed by full logic CML can be checked as well by the algorithm with the presumption, if the input formulae size is combinatorially large the same order of complexity is to be expected/tolerated. The polynomial and exponential sized formulae were discussed in Section 4.2.1.1 on page 151.

4.9 Summary of Verification/Model checking

A CML formula with respect to the *product machine* ΠF of a given CFsm system is viewed as a *surjection* of a corresponding formula with respect to ΠM of the generated CMpm system. The generation of the CMpm system and the associated *sum machine* ΣM from the given CFsm system has been described rigorously in Chapter-2. The CML formula transformed with respect to ΠM is in turn viewed *virtually*, upon the *sum machine* ΣM that is real. The reachability of the global-states of the former is asserted as the reachability of the Final-state-vectors of the configurations of the latter, qualified and quantified by the *modal* and *branching* operators respectively of *branching space-time* logic CML that expresses *monadic, third-order global-state formulae*.

The model-checking algorithm describes the verification process, adopting a distributed, *recursive, depth-first traversal* of some or all of the *n* state-trees of M_i, i=1..n depending on the given predicate to be verified. *Primitive conjunctive propositions* involve reachability of global-states and incur typically, non-trivial complexity among all the assumed propositions calling for the *branching off in space* from one Mpm to the other, multiple number of times depending on the number of conjuncts.

Primitive conjunctive propositions qualified by modal operators and quantified by branching operators constitute the CML formulae primarily checked. Upon reaching the Mpm-state satisfying a conjunct, *switching* of the *primary* Mpm corresponding to *branching-off in space* takes place maintaining the current *branch of time*. Different branches of time are decided only by local conflicts inherited by non-local (Mpm-)states and monitored by Minimal prefix vectors, stored along with every associated Mpm-state in the representation of Fig. C in *Appendix*.

Configurability theorem of Chapter-2 is applied to check if a *successor* of the current state visited in the primary Mpm is a *continuation* of the current configuration or not. *Cyclicity Theorem* and *Unfairness theorem* are applied to detect the termination upon failure.

Deadlocks and *System invariants* are defined as CML formulae and the detection of deadlocks shown to be feasible by the listed *model checker*, where Mpm-states satisfying given atomic propositions are replaced by *dead-states*. Accommodating the *fairness assumption* is shown to be simple.

TABLE 3 Model-checking with CTL/Net logics Versus. CML

CTL model is a TM (*Total-order Model*).	Even though $CML_{\Pi F}$ is a TM, by virtue of its equivalence with $CML_{\Pi M}$ and $CML^{*}_{\Sigma M}$, it derives all the advantages of EPM (*Extended Partial order Model*).
CTL's *monadic second-order* formulae are implemented in [1].	CML's *monadic third-order* formulae are checked in ours.
CTL formulae are checked on a single tree with recursive DFS (Depth First Search).	CML formulae are checked on a set of n trees with recursive, *distributed, nested* DFS over them individually.
For the most complex formula viz., A(g *until* h) in [1], *labeling algorithm* is needed to check for g and h, *a priori* and *label* the states accordingly.	For the corresponding CML formula, there is no labeling of subformulae necessary. In particular, A_r(g *until* h) can be implemented as: $A_r\ E_{Ir}/A_{Ir}$ (g $\wedge$ F (g $\wedge$ h)), with a *conjunctive predicate* as above.
The algorithm is exponential which is the size of the state-graph. For the *propositional logics*, any model-checker based on Nets is exponential either in the size of the net or in the length of the formula, given by the number of conjuncts in the case of a monadic formula.	The algorithm of model-checker, though has an apparent exponential factor m^k, the base and the exponent have a diffusing effect on each other. As a result, after an optimal growth up to a moderate value of the exponent, the factor droops down. The important result is that it is neither exponential in the size of N nor the length of the formula, which is the number of conjuncts and disjuncts in the *disjunctive normal form*.

Chapter 5
Summary and Conclusion

5.1 What is accomplished?

We assume an input specification that is a fixed set of *n communicating sequential processes* (CFsms). The set of (local) states of each process is associated with a respective set of *atomic propositions* given, mapped as a *bijection* from the states of each process. The processes communicate by synchronization.

5.1.1 The Problem

If the given CFsms were composed into a conventional *product machine*, there is *state-space explosion* due to the *non-deterministic interleaving* inherent in the *total-order* models. If the CFsms were transformed into an *elementary net* model which is quite straightforward, even though *true-concurrency* is modeled by its partial-order (PO), *true-choice* can not be modeled as a basic process relation and this is the reason why many partial-order computational models are based on *linear-time* logics/semantics only. This is due to the inadequacy of the *flow relation* of the Net models that will be explained in the sequel.

Partial-order reduction methods form a very popular alternative approach that represent a single interleaving instead of enumerating all, to cope with the explosion due to non-deterministic interleaving of total-order models. This representation works fine for *safety* properties which check for the *negation* of certain predicates qualified by the implicit *existential interleaving* qualifier alone. But there could be some interesting *liveness* properties where it might be useful to *assert* the concurrent holding of a given set of conditions at least in *some* or *all* interleavings *eventually*.

The main drawback of the above reduction methods in the complexity domain is that, apart from the *degenerate case* of the specification due to high degree of coupling where the methodology can not be applied, even in cases where the specification lends itself to the application, the method which involves finding an optimal set of successors of every state visited is NP-complete in general. So, implementation of these methods often involve heuristics.

5.1.2 A Solution

We have shown that, by assuming a concrete input specification of *n communicating sequential processes* modeled as n CFsms, it can be transformed to a *state-based partially-ordered* model of *n communicating processes* modeled as n CMpms constituting the so-called *sum machine*. It is *novel* in the sense that it not only represents *true concurrency* but also *true choice* in combination, without one being sacrificed for the sake of the other as in the research of many of the peers surveyed in the literature and listed by the table of [2] in Chapter-1. *True causality* and *true sequence* are the distinct features of our model also.

The states of the sum machine are related by a global, *dependency-order/causality relation* denoted $\leq$, that is *partial* (PO). Through this order, *sequence, conflict* and *concurrency* are all defined at the same basic computational level such that their union is a *total* relation among all Mpm-states. From a given input system of a fixed set of *n* CFsms, we show that a corresponding set of *n* CMpms whose disjoint union and the (inter and intra) dependency-order, i.e., the partial causality relation $\leq$ among them, constitute a sum-machine, through a surjective mapping, B.

In summary, *simultaneity* due to *synchronization points* is the basis of global *causality* as well as *concurrency* among Mpm-states which results in the notion of *Minimal prefixes*. These synchronization points also escalate the *conflict points*, *local* to state-trees of Mpms to a *global* level. These notions enable:

- Implementation of a *concrete model-checker* for a *branching-time temporal logic over partially-ordered structure* (resulting in *branching-space*, *branching-time* logic or in short, *branching space-time* logic CML), to verify the properties of the input CFsms using the sum-machine of CMpms introduced.
- Distributed traversal of local configurations to *dynamically generate* the *required runs* among the set of all general runs, as guided by CML formula checked. In other words, generation of all the required, reachable global-states of the CFsm system using a minimal subset corresponding to the Mp-vectors.
- Introduction of interleaving operator that takes away *non-determinism* in the interpretation of *reachability* of global-states checked and aids the reasoning about

varying *degrees of concurrency*: specifically, deduction of the *property of all interleavings from one*.

- Excepting the *degenerate cases* of the specification due to *high degree* and *non-determinism* in the *coupling* of input processes, which take away the applicability of the sum machine (in particular the non-enumerative aspect of interleavings and runs), the model-checking over CMpms of *deterministic property/polynomial sized CML formulae* as discussed, is *deterministic* and is of *polynomial complexity* as opposed to the quoted NP-completeness of PO-*set-methods* for the same specification and same property checked.

5.2 Comparison & Contrast with Related Work in a Pragmatic Perspective

5.2.1 CML Versus Partial order Reduction Methods

The research results of [9], [10], [13], [17]report *approximated* partial-order models with representatives of interleavings instead of all, but they work with the assumption that if a property is true for some interleaving, it is also true for all. As reported in [36], what could be perhaps described collectively as '*the set methods*' (*stubborn sets, sleep sets, persistent sets, ample sets* etc.) of [44], [9], [10], [13], [17] are essentially based on the exploitation of the *conditional commutativity* of actions rather than on *partial orders*, and thus calling them '*partial order methods*' may be partially misleading. Nevertheless, they have gained a lot of popularity over the last decade since the experimental results show a substantial reduction of the otherwise full state-graph by a variety of such reduction methods for real-life industrial applications [35],[39] necessitating a detailed comparison and contrast with our work.

These PO reduction methods or the '*commutativity based*' methods [40] as mentioned, share the goal of alleviating the state space explosion by exploiting the fact: many properties are insensitive to the order in which concurrent actions are executed. All of these methods are aimed at constructing a *reduced state-graph*, based on exploring for each visited state, only a *subset of the enabled operations* so that *only some of the successors of that state are expanded*. They differ only in the details of selecting the above subset referred to, and the properties preserved by the reduction. Thus a family of reduction methods possible are reported in the literature.

The techniques of these set methods are integrated into tools such as SPIN, VFSM-valid (cross-referred to, in the survey article of [35]) and so they inherit all the characteristics of these methods. SPIN includes an '*on-the-fly*' model checking algorithm using a Buchi automaton that corresponds to the complement of a specification. It also uses a more efficient *double DFS* (depth-first search). But since the automaton recognizes the *sequences*, (that are disallowed by the specification), only LTL (Linear time logic) formulas can be recognized. The rest of the points of comparison of the general set methods i.e., PO-reduction methods to be listed in the following, hold good for these tools as well.

The idea of Minimal prefixes of CMpms model is comparable to the representative interleavings of these methods but the difference is: with Mp-vectors, there is a possibility of generating any interleaving and so any global-state required by the formula by allowing the conjunctive predicate to guide and simulate the path of interleaving or the order in which the union of certain local configurations is performed; on the other hand, the representative global states are statically decided in the rival approach and hence there are global states that can not be generated at all (dynamically or otherwise). Also, the procedure to choose the representative set is in general NP-complete as mentioned before and reiterated as follows:

The *degenerate case* when the methodology loses its application is shared by these PO-*set reduction methods* (in choosing selected interleavings, that represent others). But in their case, irrespective of the influence of the specification, the implementation of the algorithm to find an optimal number of successors of every state (in the reduced state-graph) according to *equivalent robustness* condition is inherently NP-complete, and heuristics are adopted in the implementation. While in ours, there is no question of choosing any set of representatives, since we allow the flexibility to generate every interleaving/run as required by the formula dynamically with local states and their Mp-vectors. *Generation of all local states and their Mp-vectors is mechanical as opposed to making a prudent choice of optimal set of successors of every state when visited, according to certain equivalence robusmess criteria.* In addition to being mechanical, we buy the saving due to non-enumeration of runs (that comes from storing the local states alone) as well as the flexibility to build any interleaving possible and deduce the rest. There is no restriction of properties/ formulae checked since there is no equivalence robustness to be met in our case.

CML shares the disadvantage with these set-methods that in the *degenerate cases* of the specification when the degree of coupling is high the methodology can not be applied successfully as there is no scope to apply the non-enumerative aspect of interleavings. In our case, the non-enumerative aspect of runs can not be applied also due to ***non-deterministic coupling*** which gets worse with ***high degree*** of coupling. The degree of coupling is high when the number and the tightness (number of participants) of synchronous transitions are high.

But the consolation in both the approaches is that, in the degenerate cases the combinatorial explosion that takes place is not as bad as the cases when there is a high degree of asynchrony but no approach is adopted to apply the non-enumeration. This is because, with lock-step synchrony that is the result of high degree of coupling, the paths of global-states are relatively limited and regulated by the synchrony itself as opposed to having a high degree of asynchrony with no measure taken to control the explosive possibilities of paths of global-states. Thus the state-explosion becomes adaptive to the degenerate condition in a self-stabilizing manner.

Minimizing the non-determinism in the synchronous transitions in the given CFsms is a way of avoiding the explosive, *induced, local conflicts* inherited from other processes thus sustaining the savings due to non-enumeration of runs. Tapering off the number of participants from such non-deterministic synchronizations is another way in particular, and keeping the degree of coupling low, in general. This sustains the savings due to non-enumeration of runs and interleavings.

5.2.1.1 Disadvantages of PO-reduction

The limitations of PO-reduction/set methods listed below are in contrast to CML backed by sum-machine:

- Most of these methods model the executions of programs as computation sequences, in particular with the underlying logic LTL (Linear-time Temporal Logic) and not the branching-time one. There is an exception to this which will be compared in the sequel.

- The expressiveness of properties verifiable is rather low: Many properties in LTL are restricted to *deadlock-freedom* or *safety* properties alone, i.e., the ones that are preserved by or insensitive to the reduction often referred to as *equivalence robust* properties.
- The work reported in [40] is claimed to be the *first approach* that is most recent, combining *partial-orders* with *branching-time semantics*. But even in *non-degenerate cases* when the system specification to be checked is amenable to the methodology, finding optimal *ample sets* is NP-complete and only heuristics are used for the implementation in this work.

Specific comparison with the most recent logic CTL-X based on the set-method is discussed below.

5.2.1.2 CML versus CTL-X

Partial-order reduction methods have been implemented for assertional languages that model the logic LTL. The approach of [40] claims to show, for the first time, that PO reductions can be applied to *branching-time logics*. CTL-X is the result of this approach. The following are the main points of comparison/contrast:

(i) Even though this logic has model-checking algorithm that is *linear* rather than exponential in the *size of the checked property* and the experimental results show substantial reduction in the size of state-graph, in general the reduction problem is reported to be PSPACE-*hard, in the number of program operations*.

(ii) As its name suggests, the *next-time operator* is disallowed in the logic.

CML clearly is in contrast to the above demerits and it is needless to reiterate them here.

5.2.2 Comparison with Net based Models

5.2.2.1 Comparison with Petrinet based analysis tools like PEP Etc.

The report in [36] discusses the verification test bed called *Programming Environment based on Petrinets* (PEP), in particular, its highlights and shortcomings, and so makes an ideal candidate to compare and contrast with our work.

The system accepts two types of input: a parallel program written in a simple language called $B(PN)^2$ (*Basic Petri Net Programming Notation*), and a property expressed in a temporal language called ***BL*** (*Branching time Logic*). Through a sequence of compilation and verification steps, PEP allows the property to be checked against the program.

The logics BL Versus CML:

The logic BL is *propositional over places* quite similar to our basic partial order model of $CML_{\Sigma M}$. In particular, the subformulae such as $X(l_1 \wedge l_2 \wedge .. \wedge l_m)$ is the same as our *primitive conjunctive* formulae, where l_i is a *literal* i.e., an *atomic proposition* (or its complement) over s_i, the set of *places*, a *place* of the Petrinet being comparable to Mpm-state of ours.

- The complexity of model-checking the formula of above type is *exponential in the length of the formula conjunction* in BL. Though in CML, an exponential factor m^k is present in the *upper bound*, fortunately m tends to be very small when the exponent k approaches n, the maximum length of the conjunction, thanks to the concept of Minimal prefix and its link with the degree of synchrony.
- The *eventuality properties* can not be specified in BL. In fact it is admitted that PEP does not support a strong logical system as yet.
- It is also admitted that among the Petrinet based verification systems such as PEP, INA and PROD (that are cross referenced in [36]), every one has its strengths and weaknesses and a combination of different tools seems to be desirable. There is no specific weakness of CML that is really undesirable.

5.2.3 Linear Algebra based model-checking

This is a *semidecision verification method*, based on a linear upper approximation of the state space[36]. The method extracts from the description of a net, a set of linear constraints L that every reachable marking must satisfy. Thus, the solutions of L are a superset of the reachable markings. In order to make use of L for verification, a new set LP of linear constraints is added to it, which specify the markings that do not satisfy a desirable property P. Then, linear programming is used to solve the system $L \cup L_P$; if the system has no solution, every reachable marking satisfies P.

Currently, there are semidecision-algorithms for deadlock-freedom and for the reachability of a marking but not supported by a logic in general. Semidecision algorithms are only a compromise between the inherent algorithmic complexity of fully automated verification and the aid of computer-assistance during validation.

5.2.4 Tableau Constructions in Model-checking

A *tableau* T is a directed-graph. The tableau-construction in general, translates a temporal-logic formula to an automaton that usually accepts the *set of linearizations* satisfying the formula. Thus it is a popular methodology for checking *linear-time versions* of logics (as opposed to our goal of branching-time ones) such as LTL. This allows both checking the validity of formulae and model-checking of program properties[34]. The asynchronous *Buchi automaton, Streett automaton* are generated by tableau-constructions from LTL specifications.

Several logics [11], [12], [21], [34], [50] allow specifying properties over partial-order executions. All these are linear-time based, as *finite acceptors* for PO executions/event-structures with conflicts are not known. For instance, TLC reported in [34] is one such. For model-checking of a TLC specification g of a concurrent program P, first an automaton $M_{\wedge g}$ for the negated property followed by M_P that generates the program executions, are constructed. Then it is checked if the intersection of the languages of these two automata is empty. Since g does not distinguish among linearizations of the same partial-order, M_P may generate only one representative per equivalence class, thus admitting the benefits and drawbacks of partial-order reduction methods.

Thus, tableau-construction is associated with generating automaton (often in exponential time) that are *recognizers of traces or linearizations of PO executions* only without *conflicts* being taken into account. Comparison/contrast with CML and sum machine is quite obvious here.

The rest of the allied recent work in the literature, just as the ones compared and contrasted above, either are based on *linear-time semantics only with PO-semantics (branching space)* or *branching-time semantics with linear space only (without PO-semantics)* alone and hence are not repeated.

5.3 Comparison & Contrast with Peers in an Abstract, Modeling Perspective

5.3.1 CML Versus CTL and *F(B)*

F(B) was developed along CTL* and CML is developed along *F(B)*. CTL is a *total-order* based logic. Though it is claimed to be 'branching-time', the paths treated as runs in this logic include *non-deterministic, interleaved paths* as well as the genuine runs originating due to conflicts alone. Though *F(B)* is a *partial-order* based logic along with branching-time feature, it does not support a model-checker and the reason is elaborated in the following subsection.

$CML_{\Pi M}$ is a total-order model (TM), that is built by the partial-order model (PM) of $CML_{\Sigma M}$, with the runs of the former configured by the local states of the latter that form a global partial-order. So, $CML_{\Pi M}$ enjoys the advantages of the operational-semantics of a TM (total-order model) as well as those of a PM resulting in the alleviation of the state-explosion problem of CTL. Viewing the runs as partial-ordered configurations that form a set of *n* respective paths of the *n* input processes has multiple advantages in contrast to CTL: *Path interleaving formulae* in addition to and distinct from *run formulae* improve the expressiveness of the specification of reachability of states. Since the *runs* correspond to *configurations* which can be formed as the *union* of local configurations according to Summation Lemma of Chapter-2, enumeration of all possible configurations and so runs is not necessary. *Causality* based on *simultaneity* of state entries enables the checking of *universality* property of *interleavings* without their enumeration as well.

More specifically, the sum-machine ΣM comprises a set of concurrent automata that comes with the property of state-based partial order, that is not yet put forth by any related work, to our knowledge. The concept of *Minimal prefix* that issues out from the state-based causality supports $CML_{\Pi M}$, a total-order model (TM) that is equivalent to the extended-partial-order model (EPM) $CML^*_{\Sigma M}$, as proved in Chapter-3.

5.3.2 Comparison with Traditional Event-Oriented PO Structures

In the case of sum-machine, the entities ordered are Mpm-states (even though the ordering is based on state-entry which can be considered as an event) as opposed to the events themselves. The main advantage of relating the states (i.e., by their entries) lies in the capturing of synchrony that gives rise to *simultaneity of two or more distinct states* each belonging to different processes. In the case of events and the causal ordering among them, synchronization or simultaneity can only be modeled by a single identical event, one each from multiple processes. So, there is no additional information captured pertaining to the simultaneity of the synchronous events that is not present in the asynchronous events.

By modeling *simultaneity of states* of processes which are *distinct*, as an *equality relation*, we capture an additional information that is not present among asynchronous states. We are then able to define *global causality* by extending *equality/simultaneity* transitively with the local reachability relations. *Concurrency*, defined as the complement of the union of *sequence* and *conflict* also exhibits a semblance of *causality*: *A state is related by concurrency to all the states that are equal/simultaneous to it as well as to those that are reached asynchronous of it.* Since *simultaneity* is a relation (equality) rather than *unrelation* or *independence* among states, *concurrency* and *causality* relations are allowed to overlap in the sense that two states can be related by both the latter relations. A global-state can have all its n components *causally related* to each other in addition to being *concurrent*, by their definition. This result allows the deduction of *universality property* of interleavings from the configurations. In other words, without enumerating global-states, we can reach all of them as far as the expressibility of the logic requires, *using the local states and their concurrency and causal relationships.*

This we claim, as the advantage of choosing the *state-entries* as entities for causal ordering as opposed to *event-occurrences*. *This does not mean that the causal ordering among events is incomplete. The causal order among events is as much representable as that of states within the framework of sum machine, without contradicting any of the results. It is just that the event-ordering is not needed in this application.*

In this regard, a comparison/contrast needs to be done with the Petrinets/Occurrence nets that can model both *synchrony* and *asynchrony*, as well as with the behaviour models such as *prime event structures* in the paradigm of *asynchronous communication*.

5.3.2.1 Comparison with Petri/Occurrence Net Models

The models based on Petrinets and their derivatives such as Occurrence nets define the *causality* among events primarily which makes the ordering *inadequate* in the following sense:

The *causality* (dependency-order) in Petrinets (and its derivatives) comes from the *flow relation* defined as: $F := (S \times T) \cup (T \times S)$ where S denotes the *places* and T the *transitions* comparable to Mpm-states and events respectively of ΣM. Then, causality is defined as: $\leq := F^{+}$.

F forces the dependency-order among places in such a way that for two places in S to be dependent, they need to be sandwiched i.e., punctuated by at least one transition. This essentially defines only the *sequential* relation among places and not in general, the *causal* relationship among them; that is, one *entering before* the other, not necessarily in a *sequence*. If two states are *sequential,* one has to *exit* before the other's *entry*. In general, one *place* can hold a token before another place of a different process progressing *concurrently,* and continue to have the token even after the other place gets its token. Thus, the latter place can be dependent on the former *causally,* without particularly being *sequential*. This general dependency-ordering among places is not captured by F and hence the *inadequacy*.

5.3.2.2 Lacunae of Net models

The lacunae mentioned in the last subsection stems from the following facts:

(i) *Simultaneity* is not part of the event-structure of the model, since one can not capture any non-trivial information by modeling simultaneity among events as explained in the last section. Equality subset of the causality-relation $\leq$ comes solely from the *id* function and not from $sync_{out}$/*simultaneity* relation even though Nets include synchronous transitions.

(ii) *Concurrency* is rather treated as the *complement* of *causality relation* $\leq$, with the following significant implications that are detrimental to both modeling and implementation domains:

(a) Because *causality* and *concurrency* relations are complement of each other, their union forms the *universe* and therefore, *conflict* relation is automatically pushed out of the universe. This is why the *basic process structure* of nets can not consider all the three entities viz., *sequence, conflict and concurrency* at the same computational level. Consequently, to come up with a *finite acceptor* and a concrete *model checker* that handles both *conflict* and so *branching-time semantics*, with *concurrency* and so *partial-order semantics* at once are hard in this paradigm.

(b) Since *causality* and *concurrency* are mutually exclusive, the benefits of the former reinforcing the latter to derive the *universality* properties of interleavings can not be utilized.

The sum-machine ΣM models *true concurrency* and *true choice* as explained in the content of the previous chapters as well as *true sequence* and *true causality*, all in the same layer of execution. Consequently, we have a *finite acceptor* viz., ΣM for branching-time, branching space CML structures, i.e., for a *spatial, temporal* logic. Hence a concrete model-checking algorithm is a reality, with the above acceptor as the platform for $CML^*_{\Sigma M}$, the extended PM, which simulates $CML_{\Pi M}$, the TM.

5.3.2.3 Lacunae of Prime event structures with Conflicts

Pratt's [18] and Winskel's [15] event structures and their derivatives come under this category. As opposed to Net models, the *synchronous communication* is completely *hidden* in these models, and the event-structure represents only the asynchronous communication. In this sense, the *physical communication mechanism* is abstracted out and the causal order among events denoted as $\leq$ is assumed to be a *granted* rather than a *derived* notion unlike ours.

Here also, as in the Net models, *concurrency is defined as unorder*, i.e., as a complement of $\leq$. So, all the demerits mentioned in the last subsection for Nets model apply to this model as well, centered around the fact that all the three relations namely, *sequence, conflict* and *concurrency* can not be expressed in the same layer/level of program execution. So, we either have *causality and concurrency* based logics in *linear time* only or if branching-time semantics is incorporated with *conflicts*, only the *linearizations of PO structures* are recognized. The *asynchronous Buchi automata* and *Streett automaton* [34] are recognizers of *traces* or the *set of linearizations* of a PO execution and not the PO structure as such. As a result, modal operators over *causal structure (PO)* involve reinterpretation using certain *auxiliary operators* over the linearizations of PO.

Following is the claim: The physical communication mechanism viz., synchrony/simultaneity needs to be represented rather than abstracted out from the model. When the causality-relation is *derived* from the concrete notion, the advantage propagates in all directions; particularly, it seems to be possible to extract the *concrete set of paths* from the *abstract labelled partial orders/configurations* and link them to *finite automata* as well as to a logical system and its algorithmic implementation, by the model-checker.

5.3.2.4 Comparison with Reisig's work

The fundamental work of Reisig [2] reports a *mapping* between an *en-system* (elementary-net system) and an *occurrence-net* to model a *run* of the former. This mapping is similar to and very much instrumental in the composition of the mapping B_i, i=1..n and B in our work between CMpm and CFsm systems. There is also the logic *F(B)* proposed in this work. But Reisig's model has the lacunae mentioned in a previous subsection that manifests as the following drawback:

There is a discrepancy between the logic and the model of occurrence net. Even though the former accommodates conflicts, the latter does not represent it as a basic entity like concurrency and sequence. So, the *tree of infinite occurrence nets*, perceived by the logic is difficult to implement at a concrete level. This is consistent with the conclusion that there is no *finite acceptor* for the branching, *prime event structures* [34].

In other words, processes are *linear-time* structures only, i.e., the ones free of conflicts. due to the *inadequate* representation of the *flow-relation* as already explained before. This is the reason why the mapping in [2] could be used only to define a logic *F(B)* and its set of axioms and inference rules without an implementable, concrete model-checker to support the logic.

5.3.2.5 Comparison with McMillan's work

The work by McMillan[3] again uses the *finite prefices* of an occurrence net generated as an *unfolding* of a Petrinet specification. Our computational model is similar to this work since the truncated CMpms can be considered as the *unfolding* of CFsms specification. But in contrast to our mathematical functions B_i,i=1..n and B, the former [3] is not formally backed by a *mathematical mapping* of *occurrence-net* to Petrinet entities. It defines an *event-oriented configuration* that is inspirational and dualistic to our *state-oriented configurations*. The former fits naturally with the net's transitive closure of flow-relation, F^+. As explained already, dependency-order derived from this flow-relation models only *sequence* and not *causality* among the local states of the processes, with its consequential drawbacks mentioned already.

Dynamic birth and death of processes without fixed identity to them, makes the complexity of generation of the occurrence-net unfolding, *exponential*; all subsets of places have to be exhaustively searched in a nondeterministic fashion before generating every input transition since the determinism due to *identity* of processes and the knowledge of partners of every synchronous transition are missing in this model.

Following are the draw-backs in [3]:

(i) There is *no general logic* for specifying the properties in this work and consequently, lack of expressiveness. Only basic *existential safety predicates* can be checked using the *unfolding*.

(ii) The unfolding needs to be generated *once for every such predicate* verified which involves in the worst case, a complexity of $O(N^n)$ where N is the size of unfolding and n is the maximum number of partners of a synchronous transition.

(iii) The property of *deadlock freedom* can not be posed as a *predicate* but instead, a more complex *constraint satisfaction* approach is used in a dedicated deadlock-detection algorithm.

All the above factors favour our work in which:

(i) Richer predicates supported by a logic with modal operators enriched with *branching time* as well as *branching space*, can be specified, thus filling the void entry of the table of survey in [2].

(ii) CMpm system with respect to a given CFsm system needs to be generated just *once* for all the formulae checked.

(iii) Neither the generation nor the checking of formulae(model-checking) incurs the exponential complexity due to the enumeration of runs or interleavings of a run except in the *degenerate cases* where there is no scope for the applicability of the sum-machine.

5.4 Classical Framework Provided by Sum machine and CML

5.4.1 Finite Automata Over Partial-orders

From the recent literature [37], it is quite clear that the area of '*finite automata over PO*' is not yet an established one. The possible reason and remedy are quoted in the following :

Possible reason: "Many properties of finite automata which are essential in *logical* or *algorithmic* applications fail to hold when *partial orders* are considered as inputs to automata rather than *strings* or *trees*.

Possible remedy: To take a 'narrower view' by extracting sets of paths from partial orders which brings back the framework of classical formal language theory."

Our Claim: In our case of sum machine, we did not have to take any narrower view to extract a PO. They simply turn out to be the case, resulting from a PO structure of a configuration of Mpm-states being a set of *n unique paths* of *n* respective Mpms of the sum machine, which can be considered as a collection of *finite*, *deterministic synchronous automata* with respect to given set of *n* CFsms.

Every *point* or every Mpm-state of a configuration is assigned a *representative global-state* (using Minimal prefixes) of the communicating finite automata (CFsms) deterministically, thus forming a *labelled PO structure*. Finite prefices of these structures suffice to verify the properties of the given automata. So, we extract *finite, labelled partial orders* as configurations from essentially infinite automata, whose set of *local paths* are connected into a *state-tree*, each accounting for *branching-time* semantics, the source of *branching points* being local to each of *n* such trees.

Logics: We claim that the above set of finite, labelled, partial-order structures are *finite automata-recognizable* as well, since they are generated from the latter in the first place. Another reason to support this argument comes from the fact that these PO structures are definable in what is defined as a '*monadic third-order logic*', with the *propositions* operated by the *first-order modal qualifiers, second-order interleaving quantifiers* and the *third-order configuration/run quantifiers* to define *monadic, third-order state formula.*

Recognizability mentioned above is accomplished by viewing the *product machine* with respect to the *sum machine* and viewing the corresponding logical system that is a *Total-Order model* (TM) $CML_{\Pi M}$, with respect to $CML^*_{\Sigma M}$ that is an *extended PO model* (EPM). *The addition of the third order formulae is possible because of the fact that $CML_{\Pi M}$ which essentially is a TM can be viewed and generated as an EPM. This combines the advantages of the latter derived from a PM (purely partial-order model) and at the same time, maintaining its connection with the operational semantics of the former to aid the concrete implementation of the model-checker.*

Algorithmics: We scan only the 'local neighborhood' of every Mpm's state-tree during the localized depth first search till the required local property is satisfied. Only upon local success, branching to the next non-local tree is made to continue just from *those* 'non-local neighbourhoods' as indicated by the *labels* of the former. This is how the *labelled* PO of the individual points/Mpm-states are exploited. The algorithm indicates that the *language of finite labelled PO is recognizable* without the state-explosion of the traditional TM.

5.4.1.1 Scope of Work in Automata/Language Theory

We believe with the above foundational support, many open questions posed in [37] are answerable in $CML^*_{\Sigma M}$ as listed below:

- The theorem by *Buchi* and *Elgot* quoted in [37] is for a class of *acyclic graphs* that are *strings/words,* definable in *monadic second-order* logic. This can be extended to a class of *labelled partial-orders,* definable in the *monadic third-order language* of $CML^*_{\Sigma M}$ and equivalently, in that of $CML_{\Pi M}$.
- Proof of the *closure* properties such as union, complement and projection of the language above in classical formal language theory is another possibility.

5.5 Conclusion

The conclusion must be quite clear from the comparison and contrast with the peers' work discussed in the sections above.

We have shown that, a *partial-order* version of the *state-oriented* model is indeed possible that is supported by an expressive, ***branching space-time temporal logic*** CML whose formulae over *labelled PO structures* specifying both *safety* and a variety of *liveness* properties can be checked *efficiently* as allowed by minimal *non-determinism* in the input *specification structure* and the ***property checked***. In doing so, we believe to have filled the void entry of Reisig's table in [2] meaningfully, which was illustrated in Fig. 1 and Fig. 2 of Chapter-1.

By relating the *local state-entries* as opposed to *event-occurrences*, in particular, the *equality of the distinct synchronous ones*, *simultaneity* is accounted for, which becomes the controlling agent for both *concurrency* as well as *causality*, that are not mutually exclusive/complementary among their related states. This aspect, combined with the *sufficiency of local conflicts* to account for *global ones* is exploited in the model checker. The *events are truly modeled* as well (although, they are not exploited and hence not demonstrated in this application). This is why the *size of the sum-machine* increases with the *degree of coupling* as in the event-oriented models as opposed to the increase with *asynchrony* as in state-oriented ones.

5.6 True Conclusion

Just as both *magnetism & electricity*, *voltage & current* and *inductance & capacitance* are dual to each other in electromagnetic theory, both *state* and *event* are duals of each other and both entities must symmetrically be represented as the primary entities in the model of a concurrent/communications system without either one of them being compromised for representing the other. They do co-exist and both are necessary; except that, depending on the application, one entity or the other may be emphasized and projected in the computation, rather than its dual.

If a specification is first represented in the *concrete domain* and then escalated to the *abstract domain* of a model in a bottom-up fashion, it seems to be easy to bring down the results of the abstractions back to the concrete world, in the reverse direction where they are put to use. The set of Mpms and their communication by *sync* relation represent the physical communication mechanism in the *concrete* domain, and the *theory of extended sum machine* with its configurations as *labelled PO structures* constitutes the *abstract* domain of the model. The relationship between the *configurations* and their association with the *fixed set of paths* of Mpms is the link bridging the two domains, which enables a tractable implementation of the decidability of CML formulae.

5.7 Scope of Future Work

The algorithms for checking *non-monadic higher-order formulae* with nesting, as explained in Chapter-4 are not handled here, which is hoped to be feasible with suitable labeling algorithm for plugging in the results of the intermediate, inner state-formulae between levels of nesting.

It is true that we have not reported any experimental results for industrial sized examples nor the results for *bench-mark* examples such as Milner's scheduler described in [23] etc. But it is our earnest belief, as is also common-place in the literature, that the experimental results are often quoted when the algorithm is in general PSPACE-hard/NP-complete and in lieu of which, heuristics are adopted. But in the case of *sum-machine* and *CML pair*, the model-checker is *polynomial*, for a polynomial sized formulae and non-determinism-free communication of the input specification. All said and done, it would be worthwhile to reinforce the theoretical results with experimental ones by actually implementing our model-checker and checking it with a wide range of examples it is expected to cover. This is left for the future work.

The complete axiomatization of the logic CML, proving its *soundness* and *completeness* in a classical manner, could be another possible direction of extending this work.

Towards another classical result, as already mentioned, an extension of the *Buchi, Elgot theorem* stated in [37] can be made as: the class of *language over partial-orders* (as opposed to *strings*) is *recognizable* by a finite set of Mpms *iff* it is *monadic third-order* (MTO) *definable*. The forward direction of the proof is more or less the *complexity theorem II* proved in Chapter-4, but the converse direction from MTO-formulae to the sum machine requires standard closure properties under union, complementation and projection that need to be proved. This may not be difficult, given the result of *equivalence* of *non-deterministic model* of CFsms and the *deterministic model* of finite CMpms, proved in Theorem 2.10 of Chapter-2.

With the inherently infinite Mpms, some more unexplored direction could be towards the theory of recognizable sets of *infinite PO structures* such as: the meaning of '*regularity*' of PO structures, the *non-emptiness problem* over the classes of these infinite PO etc., as raised in [37] again.

We believe to have laid a reasonably and sufficiently classical foundation to answer all the above open questions.

References

[1] E.M. Clarke, E.A.Emerson, and A.P. Sistla, "Automatic verification of finite state concurrent systems using temporal logic specification: A practical approach", ACM TOPLAS, 1983.

[2] W. Reisig, "Towards a Temporal Logic for Causality and Choice in Distributed Systems", Lecture Notes in Computer Science: LNCS 354, pp 603-628, 1989.

[3] K. L. McMillan, "Using unfolding to avoid the state space explosion problem in the verification of asynchronous circuits", Proc. 4th Workshop on Computer Aided Verification, 1992.

[4] W.Reisig, "Temporal Logic and Causality in Concurrent Systems", LNCS 224, 1988.

[5] E.A. Emerson and J.Y.Halpern, "Decision Procedures and Expressiveness in the Temporal Logic of Branching Time," Journal of Computer and System Sciences, 1985.

[6] K.H. Rosen, "Descrete Mathematics and its Applications", Random House, New York, 1987.

[7] W.Reisig, "Elementary Net systems - Part-I, condition/event systems".

[8] Norio Shiratori et al., "A Verification Method for LOTOS specifications and its application", Protocol Specification, Testing and Verification IX, IFIP, 1990.

[9] Patrice Godefroid, "Using Partial-orders to improve automatic verification methods", In workshop on Computer aided Verification, 1990.

[10] P.Godefroid and P.Wolper, "Using partial orders for the efficient verification of deadlock freedom and safety properties", Formal Methods in System Design, 2(2): 149-164, 1993.

[11] S.Pinter and P.Wolper, "A temporal logic for reasoning about partially ordered computations", Proc. 3rd ACM Principles of Distributed Cimputing, 28-37, 1984.

[12] S.Katz, D.Peled, "Interleaving Set Temporal Logic", Theoretical Computer Science 75 (3), 21-43, 1992.

[13] S. Katz and D. Peled, "Defining conditional independence using collapses", Theoretical Computer Science, 101: 337-359, 1992.

[14] G. Boudol, I.Castellani, "Concurrency and Atomicity", Theoretical Computer Science 59, 1988, pp 25-84.

[15] G.Winskel, "Event structure semantics for CCS and related languages", LNCS 140.

[16] A. Mazurkiewicz, "Trace Theory: in Petrinets: Applications and relationships to other models of concurrency", LNCS 255, pp 270-324, 1986.

[17] D.Peled, "All from one, one for all: on model-checking using representatives", Proc. 5th workshop on CAV, 1993.

[18] V. R. Pratt, "Modelling Concurrency with Partial Orders", International Journal of Parallel Programming", Vol. 15, No.1, 1986.

[19] E.Najm, "A Verification oriented specification in LOTOS of the Transport protocol," IFIP 87.

[20] J.L. Richier et.al, "Verification in XESAR of the sliding window protocol," Protocol Specification, Testing and verification, IFIP 1987.

[21] K. Lodaya, P.S.Thyagarajan, "A modal logic for subclass of event structures", 14th ICALP, LNCS 267.

[22] C.A.R. Hoare, "Communicating Sequential Processes", Prentice Hall, 1984.

[23] R. Milner, "Calculus of Communicating Systems", LNCS 92.

[24] Z.Manna, A.Pneuli, "The anchored version of the temporal framework", LNCS 354, pp. 201-285.

[25] C.Stirling, "Temporal logics for CCS", LNCS 354, pp660-673.

[26] T. Bolognesi, S.A. Smolka, "Fundamental results for verification of observational equivalence - a survey", Protocol specification, testing and verification VII, IFIP 1987, pp 165-181.

[27] D.K.Probst, H.F.Li, "Abstract specification, composition and proof of correctness of DI circuits," Technical Report, Dept. of CS, Concordia University, 1989.

[28] E. Brinksma, "An Introduction to LOTOS," Protocol specification, testing and verification VII, IFIP 1987.

[29] S.D. Brookes, C.A.R. Hoare, A.D. Roscoe, "A Theory of Communicating Sequential Processes", J. ACM 31, pp 560-599, 1984.

[30] G.V. Bochmann, C.A. Sunshine, " Formal Methods in Communication Protocol Design", IEEE Transactions on Communication, vol.COM-28, no.4, 1980.

[31] R.P. Kurshan, "Analysis of Discrete Event Co-ordination", Technical Report, AT&T Labs, New Jersey, 1990.

[32] Burns, M. Gouda, and Miller. "On relaxing Interleaving Assumptions," IFIP 1990.

[33] R. Ramanujam, "Locally Linear Time Temporal Logic", Proc. of Logic in Computer Science, 1996.

[34] R.Alur, D. Peled, W.Penczek, "Model-checking of Causality Properties", Proc. of 10^{th} symposium on Logic in Computer Science, IEEE, 1995.

[35] D. Peled, "Partial Order Reduction: Linear and Branching Temporal Logics and Process Algebras", DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 29, 1997.

[36] E. Best, "Partial order Verification with PEP", DIMACS Series, vol 29, 1997.

[37] Wolfgang Thomas, "Elements of an Automata Theory Over Partial Orders", DIMACS Series, Vol 29, 1997.

[38] U.Montanari, M.Pistore, "History Dependent Verification for Partial Order Systems", DIMACS, Vol 29, 1997.

[39] P. Godefroid, "On the Costs and Benefits of using Partial-order Methods for the Verification of Concurrent Systems" (Invited Paper), DIMACS Series, Vol 29, 1997.

[40] R. Gerth, R.Kuiper and D.Peled, "A Partial Order Approach to Branching Time Logic Model Checking", Third Israel Symposium on Theory of computing, 1995.

[41] G.Plotkin, V.R. Pratt, "Teams Can See Pomsets"(Preliminary Version), DIMACS Series, Vol 29, 1997.

[42] W. Reisig, "Interleaved Progress, Concurrent Progress and Local Progress", DIMACS Series, Vol 29, 1997.

[43] D. Mandrioli, C. Ghezzi, Theoretical Foundations of Computer Science, Wiley, 1987.

[44] A. Valmari, "Stubborn Attack on state explosion", CAV'90, DIMACS, vol 3, 1991.

[45] E. A. Emerson, J. Y. Halpern, "'Sometime' and 'not never' revisited: On Branching versus Linear Time", CACM 1983.

[46] D. Peled, "Sometimes 'sometime' is as good as 'always'", CONCUR '92, Aug 1992.

[47] G. J. Holzman, P. Godefroid and D. Pirottin, "Coverage Preserving Reduction Strategies for Reachability Analysis", In Proc. 12th IFIP, June 1992.

[48] P. Godefroid, P.Wolper, "Using Partial order for efficient verification of deadlock freedom and Safety Properties", In Proc. Third Workshop on CAV 91.

[49] P. Godefroid, D. Pirottin, "Refining Dependencies Improve Partial-order Verification Methods" In Proc. Fifth Workshop on CAV '93.

[50] D. Peled, A. Pneuli, "Proving Partial Order Liveness Properties" LNCS 443, 1990.

[51] R. Milner, "Calculi for Synchrony and Asynchrony", Theoretical Computer Science 25, pp. 267-310, 1983.

[52] J.R. Burch, E.M. Clark et al., "Symbolic Model Checking : 10^{20} states and beyond", Proc. Fifth annual Symposium on Logic in Computer Science", June 1990.

[53] V. Sassone, M. Nielson, G.Winskel, "A Classification of Models for Concurrency", Technical Report DAIMI, Computer Science Dept, Aarhus University, 1993.

[54] G. Winskel, "Event structure Semantics of CCS and Relataed Languages", Proc. of ICALP 1982, LNCS 140.

[55] G. Winskel, "Synchronization Trees", Theoretical Computer Science, no:34, pp.33-82, 1985.

[56] Stephen A.Cook, "Soundness and Completeness of an Axiom system for program Verification", SIAM Journal of Computing, Vol. 7, No. 1, Feb 1978.

[57] M. Nielson, G.Plotkin, G.Winskel, "Petrinets, Event structures and Domains, Part I", Theoretical Computer Science 13, NO. 1, pp.85-108, 1980.

[58] M. Hennessy, R. Milner, "Algebraic Laws for Non-determinism and Concurrency", J. ACM, 32, 137-161, 1985.

[59] S. Katz, D. Peled, "Verification of Distributed Programs using Representative Interleaving Sequences", Distributed Computing 6, pp. 107-120, 1992.

[60] R. M. Keller, "Formal Verification of Parallel Programs", CACM 19, Vol. 7, 1976.

[61] A. Pneuli, "The Temporal Logic of Programs", Proc. of 18th symposium on Foundations of Computer Science, Nov 1977.

[62] Van Nguyen, Alan Demers, D.Gries & S.Owicki, "A Model and Temporal Proof system for Networks of processes", Distributed Computing, 1986.

[63] T. Murata, "PetriNets: Properties, Analysis and Applications", Proc. of IEEE, Vol 77, April 89.

[64] J. L. Peterson, "Petrinets", ACM Computing Surveys, Vol. 9, 1977.

[65] Shai Ben-David, "The global time assumption and Semantics for Concurrent Systems", CACM 1988.

[66] Vijay K. Garg, Craig M. Chase, "Distributed Algorithms for detecting Conjunctive Predicates", Technical Report ECE-PDS-94-03, Parallel and Distributed Systems Group, University of Texas, Austin, June 1994.

[67] O. Babaoglu, M. Raynal, "Specification and Verification of Dynamic Properties in Distributed Computations", Technical Report UBLCS-93-11, Univ. of Bologna, Italy, May 1994.

[68] R. de Nicola, "Extensional Equivalences for Transition System", Acta Informatica 24, pp. 211-237, Springer-Verlag 1987.

[69] D. Harel, A.Pneuli, "On the development of Reactive Systems", NATO ASI series, Vol. F13, Logics and Models of Concurrent Systems, 1985.

[70] W. Reisig, "Petrinets with Individual Tokens", Theoretical Computer Science, 41, pp. 185-213, North-Holland, 1985.

[71] L & M. Duponcheel, "Acceptable Functional Programming System", Acta Informatica 23, 1986.

[72] G.L. Peterson, "Concurrent Reading while Writing", ACM Transactions on Programming Languages and Systems", Vol. 5, No. 1, Jan 1983.

[73] P.Wolper, P. Godefroid, "Partial-order Methods for temporal verification", In Proc. CONCUR '93, LNCS 715.

[74] M. Raynal, "Distributed Algorithms and Protocols", John Wiley, 1988.

[75] K.M. Chandy, J. Misra, "Parallel Program Design", Addison-Wesley.

[76] W.H. Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems", ACM Computing Surveys, Vol. 13, N0.2, June 1981.

[77] L. Lamport, " A simple approach to Specifying Concurrent Systems", Communications of ACM, Jan 1989.

[78] L. Lamport, "Reasoning about Nonatomic Operations", Communications of ACM, 1983.

[79] L. Lamport, "On Interprocess Communication, Part I: Basic Formalism; Part II: Algorithms", Distributed Computing, 1, pp 77-101, 1986.

[80] S. Owicki, L. Lamport, "Proving Liveness Properties of Concurrent Programs", ACM Transactions on Programming Languages and Systems", July 82.

[81] G. D. Plotkin, "A Power Domain Construction", Siam Journal of Distributed Computing, Vol. 5, Sep 1976.

[82] B. Alpern, F.B. Schneider, "Defining Liveness", Information Processing Letters, 21, 1985.

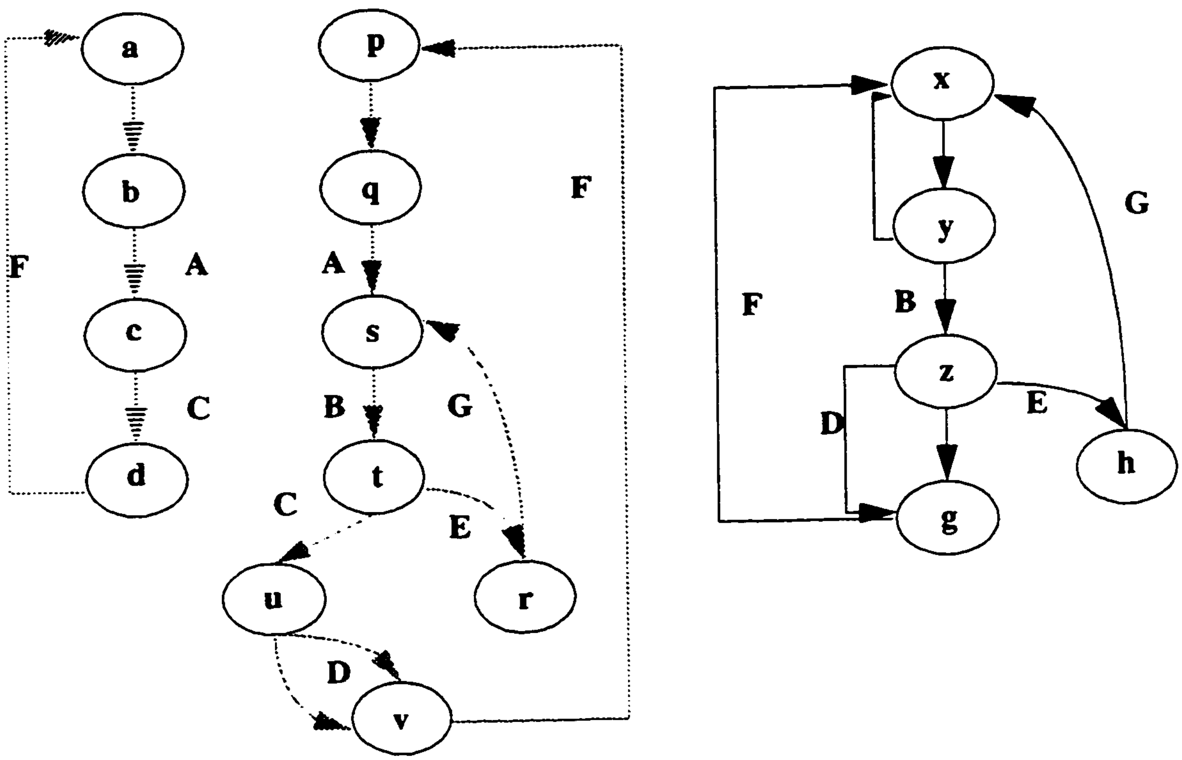
[83] G.R. Andrews, F.B. Schneider, "Concepts and Notations for Concurrent Programming" ACM Computing Surveys, 1983.

[84] W. Reisig, "Petri Nets and Algebraic Specifications , Fundamental Study", Theoretical Computer Science 80, 1991.

[85] L.Lamport, "Solved Problems, Unsolved Problems and Non-Problems in Concurrency" Proc. of Third ACM Conference on Principles of Distributed Computing", 1984.

Appendix

Fig. A. Set of CFsms



$$\mathbf{F = \{F_1, F_2, F_3\}}$$

Synchronization specification:

A, C - $\{F_1, F_2\}$,

B,G, D, E - $\{F_2, F_3\}$,

F - $\{F_1, F_2, F_3\}$.

Atomic-Proposition sets:

$Ap_{f1} = \{ a, b, c, d \}$,

$Ap_{f2} = \{p, q, r, s, t, u, v\}$,

$Ap_{f3} = \{x, y, z, g, h\}$

Fig. B The sum machine, ΣM corresponding to CFsms of Fig. A

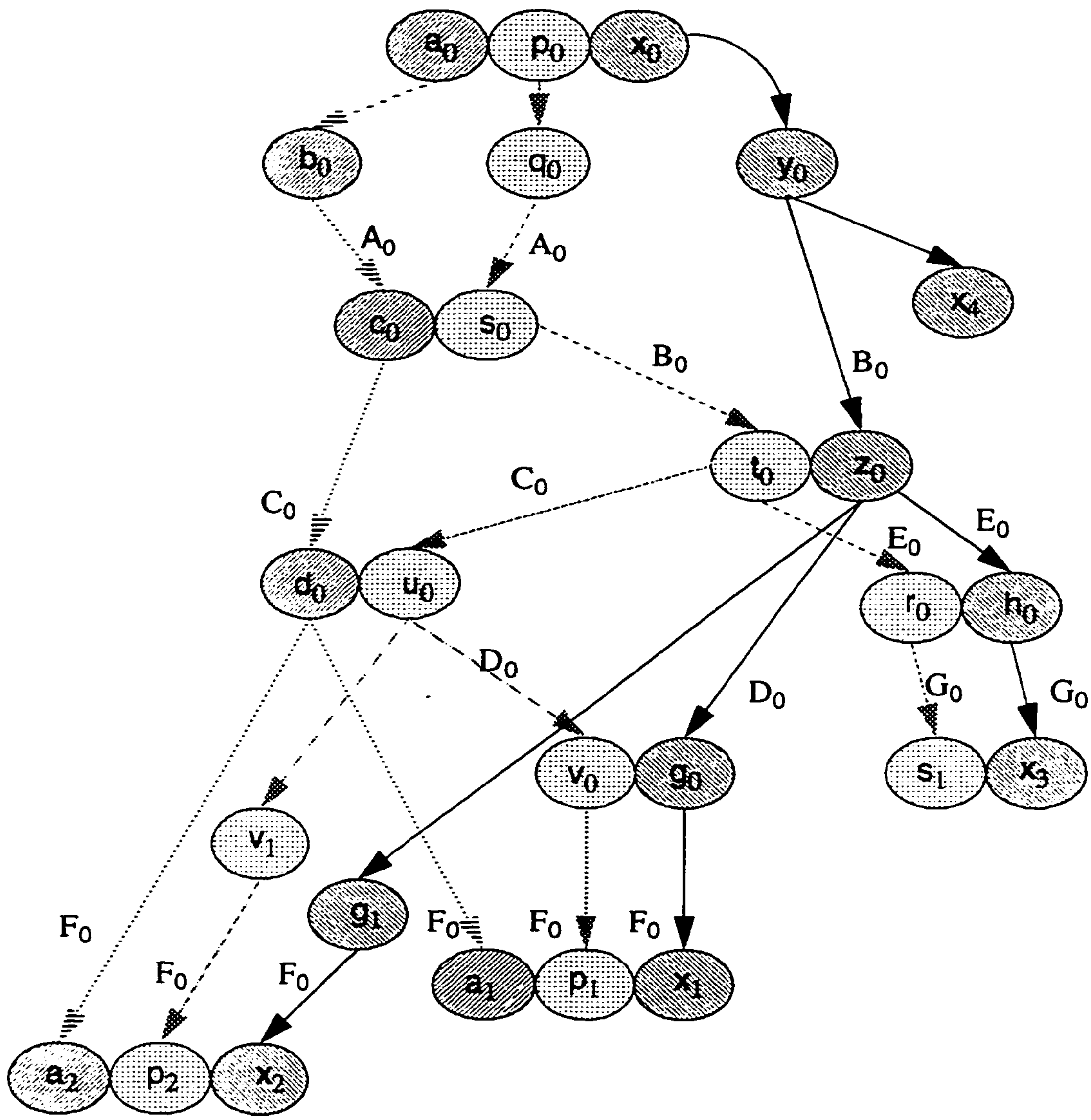


Fig. C ΣM with synch. points replaced by Mp-vectors of every state

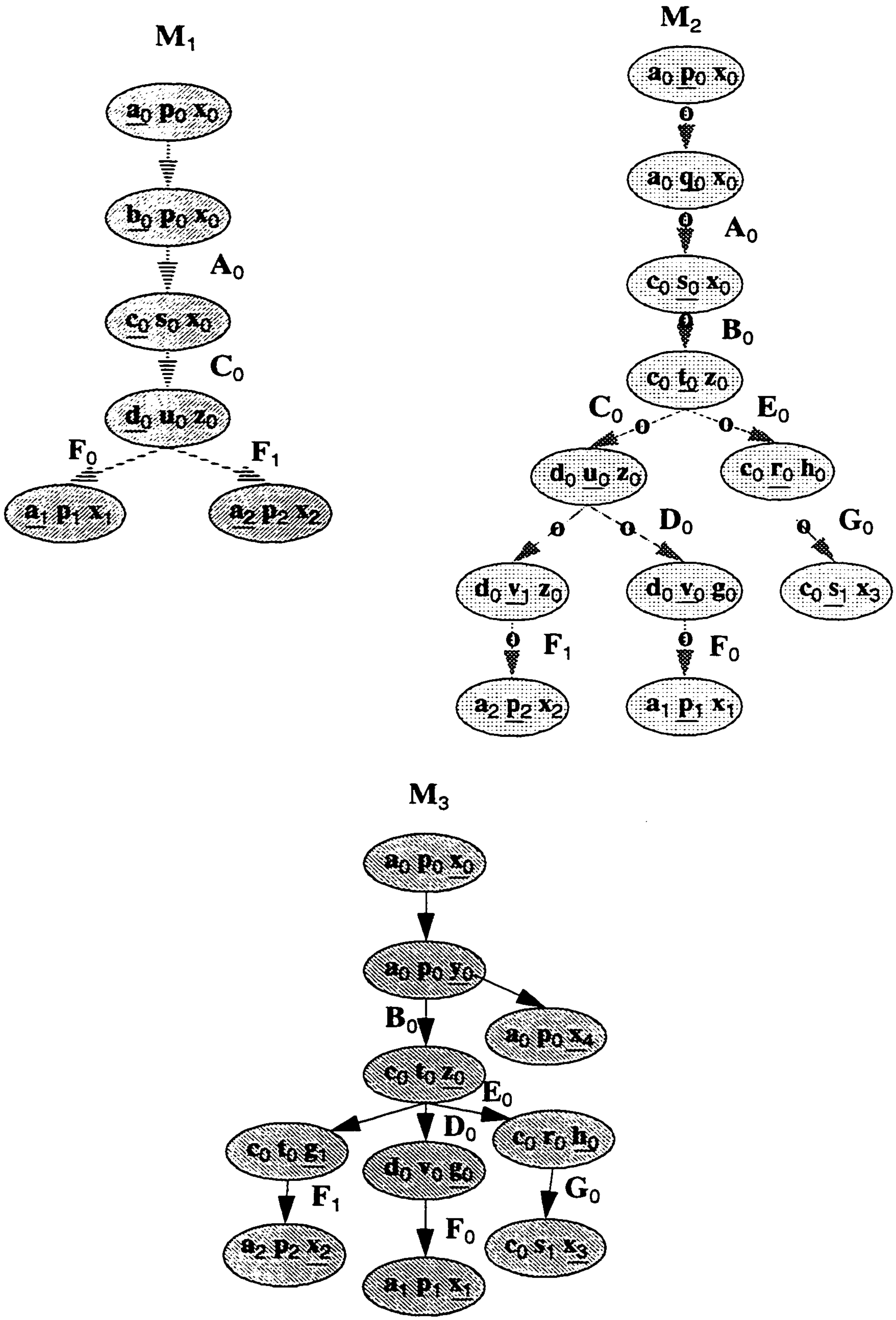
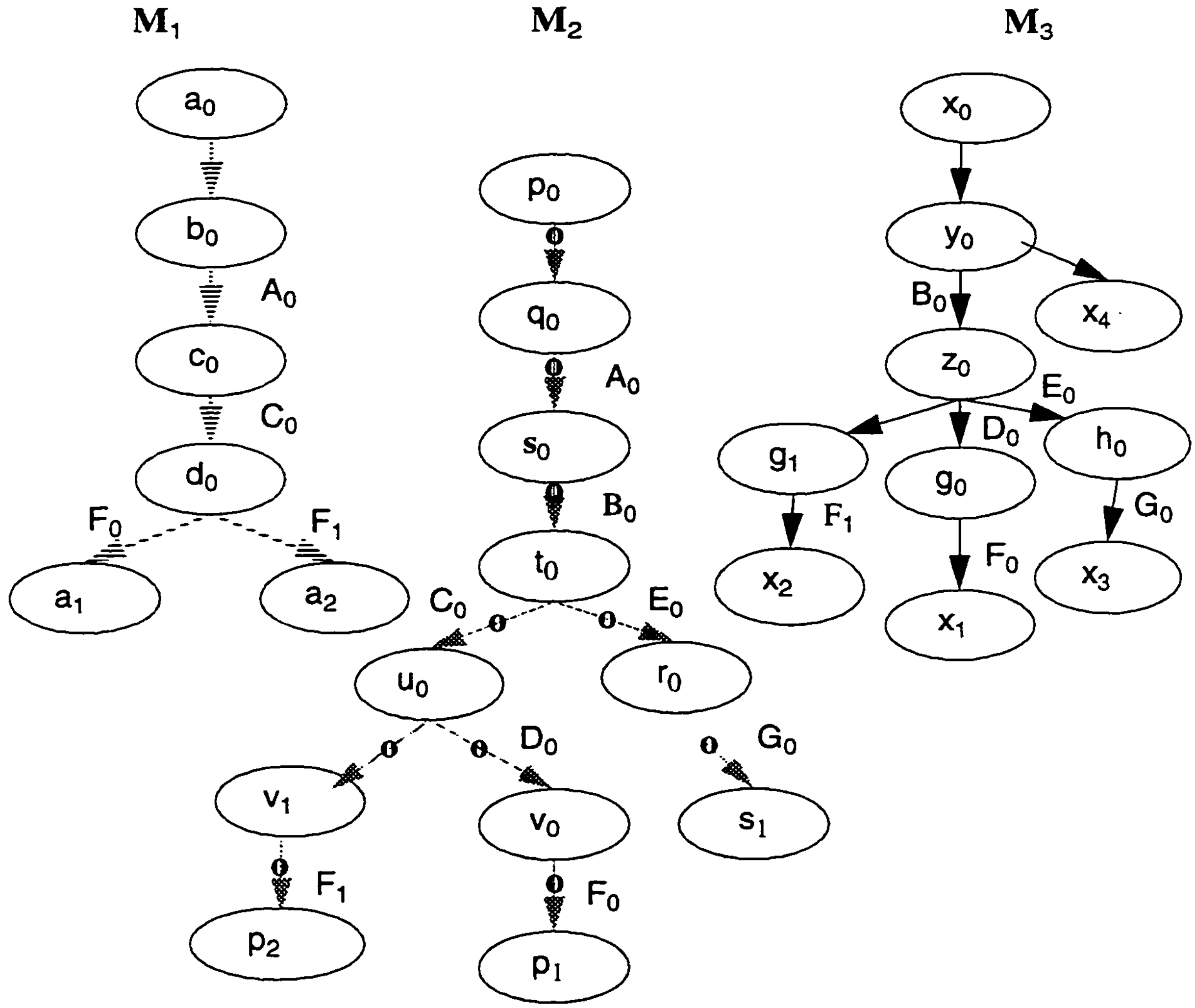


Fig. D ΣM with Mp-vectors tabulated separately



State	Mp
a_1	$a_0\ p_0\ x_0$
b_0	$b_0\ p_0\ x_0$
c_0	$c_0\ s_0\ x_0$
d_0	$d_0\ u_0\ z_0$
a_1	$a_1\ p_1\ x_1$
a_2	$a_2\ p_2\ x_2$

$Mp_1(S_{m1})$

p_0	$a_0\ p_0\ x_0$
q_0	$a_0\ q_0\ x_0$
s_0	$c_0\ s_0\ x_0$
t_0	$c_0\ t_0\ z_0$
u_0	$d_0\ u_0\ z_0$
r_0	$c_0\ r_0\ h_0$
v_1	$d_0v_1z_0$
v_0	$d_0v_0g_0$
s_1	$c_0s_1x_3$
p_2	$a_2p_2x_2$
p_1	$a_1p_1x_1$

$Mp_2(S_{m2})$

x_0	$a_0p_0x_0$
y_0	$a_0p_0y_0$
x_4	$a_0p_0x_4$
z_0	$c_0t_0z_0$
g_1	$c_0t_0g_1$
g_0	$d_0v_0g_0$
h_0	$c_0r_0h_0$
x_2	$a_2p_2x_2$
x_1	$a_1p_1x_1$
x_3	$c_0s_1x_3$

$Mp_3(S_{m3})$