

Semantics and Analysis of Scenario-Aware Dataflow Specifications

Hao Wu
Diploma Thesis at the
Software Modeling and Verification
(MOVES) Group
Prof. Dr. Ir. Joost-Pieter Katoen
Computer Science Department
RWTH Aachen University

Thesis supervisor:
Prof. Dr. Ir. Joost-Pieter Katoen

Second examiner:
Dr. Ir. Marc Geilen

at **TU/e** Technische Universiteit
Eindhoven
University of Technology

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, May 2011
Hao Wu

Contents

Acknowledgements	ix
1 Introduction	1
1.1 Thesis Contribution	2
1.2 Thesis Outline	3
2 Interactive Markov Chains	5
2.1 Classical Process Algebras	5
2.1.1 Syntax and Semantics of CCS	6
2.1.2 Stochastic Process Algebra	9
2.2 Interactive Markov Chains (IMCs)	10
2.2.1 Essentials behind IMC	10
2.2.2 Algebra of Interactive Markov Chains – IML	12
2.2.3 Interactive Markov Chains	15
2.3 The Equivalence Notions on IMC	18
2.3.1 Strong Bisimulation	18

2.3.2	Weak bisimulation	18
2.3.3	Stochastic Branching Bisimulation	19
2.3.4	Analysis of IMC	23
3	Scenario-Aware Dataflow (SADF)	25
3.1	Introduction	25
3.1.1	Scenario-Aware Data Flow(SADF)	26
	Terminology	26
3.2	Formal Definition of SADF	28
3.2.1	Operational Semantics of SADF	32
3.3	Important Properties	40
3.3.1	Boundedness	40
3.3.2	Determinacy	41
3.3.3	Long-run equivalence	42
4	The IMC Semantics of SADF	43
4.1	The elementary IMC models for SADF	45
4.1.1	The IMC Semantics of Data Channels	47
4.1.2	The Semantics of Control Channels	48
4.1.3	The IMC Semantics of Detectors	49
	The Semantics of Subscenario-specification Module	52
	The Semantics of Function Module	54

4.1.4	The Semantics of Kernels	58
4.2	The IMC Semantics of an SADF Graph	61
4.3	The Non-determinism and Properties in the resulting IMC	65
4.3.1	Infinite System caused by Non-determinism	71
5	State Space Generation, Analysis of IMC and Case Studies	73
5.1	Overview of CADP toolbox	73
5.2	Generation of IMC models	76
5.2.1	Definition of data types	76
5.2.2	Definition of Control Part	77
	Concept and Terminologies in LOTOS	79
	From IML to LOTOS	81
5.2.3	State space Generation	81
	Semi-Composition	83
5.2.4	State Space Generation	89
5.2.5	Apply Action Urgency on Generated IMC	90
5.2.6	State Space Reduction	90
5.2.7	Adding Probabilistic Information	92
5.3	Model Checking on the resulting IMC	93
5.3.1	Functional Verification	94
5.3.2	Performance Evaluation	95

5.4	Case Studies	99
5.4.1	1. Case Study	100
5.4.2	2. Case Study	102
	Functional Properties	105
	Temporal Properties	107
6	Conclusion	111
6.1	Future Work	112
	Bibliography	113

Acknowledgements

First, I would like to give my thanks to my supervisor Prof. Joost-Pieter Katoen for his advises and patience during this work. He always enlightens me by explaining the complex theories with easy understanding examples.

I also appreciate Dr. Bart Theelen at ESI institute, who has developed the SADF model, for his support and suggestions. I was very impressed by his rigorous and exhausted attitude to scientific research. Without his suggestions I could not have improved my thesis.

My appreciation also goes to the developers of CADP toolset in VASY team at INRIA Rhone-Alpes, who answered my questions very detailed in the forum of CADP toolset. Finally, a debt of gratitude is owed to my lovely parents, who always support me financially and psychologically.

Chapter 1

Introduction

The Scenario-Aware Dataflow (SADF) model is an extended Synchronous Dataflow (SDF) model, which allows to capture the dynamism in modern streaming applications by incorporating the concept of scenarios [41]. The main task of this diploma thesis is to give a framework to define the SADF specification in terms of a semantic model called interactive Markov chain (IMC). The IMC model provides a formal way to build the system in a compositional manner and to abstract the time passing by Markovian transitions. Two major adaptations are made in order to define the SADF specification in terms of IMC in our thesis: The generic discrete execution time distribution per (sub)scenario of each process in SADF is replaced by a single execution time per (sub)scenario, which is equal to the weighted mean of the execution times (discrete random variables) in the sample space of the distribution. Since the obtained execution time per (sub)scenario is a fixed (constant) value, we adapt this constant execution time to an exponentially distributed delay. The exponentially distributed delay is characterized by a rate λ , which is equal to the reciprocal of the obtained weighted mean value of execution time. Concisely, we describe the steps of rough solution as follows.

1. *Modeling.* Give each component in an SADF graph a corresponding IMC.
2. *State space generation.* Construct the whole system by using *parallel composition* and *hiding* on such resulting IMCs. Two approaches are discussed during the state space generation: in a straightforward manner or using compositional aggregation approach.
3. *Analysis of the resulting system.* In this step, we elaborate the kinds of transitions in the resulting system after state space generation and discuss the non-determinism

in the system. If inherent non-determinism is absent in our system, after stochastic branching bisimulation (reduction) we should get a IMC with only probabilistic transitions and Markovian transitions for further analysis of temporal properties. If inherent non-determinism exists, the system is much more complex.

4. *Verification.* Based on the resulting system with only probabilistic transitions and Markovian transitions, after we have eliminated the probabilistic transitions in the system, a CTMC is obtained. Functional properties can be verified by using ACTL logic on the resulting IMC and performance (temporal) properties can be evaluated by using steady and transient analysis on the CTMC.

1.1 Thesis Contribution

Because of the two major adaptations mentioned above, there is no comparable results from the IMC model with the results from original semantics model Timed Probabilistic System (TPS). But we still have some new aspects in IMCs.

- We define a compositional semantics of SADF specification in terms of IMC. This compositional approach allows us to construct large IMC of an SADF graph based on the IMCs representing each component in the SADF graph.
- We allow the unbounded FIFO channels in SADF also be modeled as unbounded IMCs and implemented with CADP toolset. We express the IMC semantics of kernels as a special case of detectors in SADF.
- We provide another aspect of SADF specification by using a stochastic time model, while the original one defined by TPS is a probabilistic one.
- Some functional properties in SADF specification are checked by using ACTL Logic.
- For performance evaluation, we extend the IMC to a special IMC automata with accepting states. By parallel composing the IMC automata, the property like the probability of first execution time until one process's finishes its first execution can be determined.
- We discuss the problems we encountered when we were trying to apply compositional aggregation approach during the IMC state space generation of SADF.

1.2 Thesis Outline

In Chapter 2., we introduce our target model interactive Markov chains and its algebra IML. We will see how to construct large system based on two operators: parallel composition and hiding. Three different equivalence notions on IMCs are then defined. Since these equivalence notions are congruences w.r.t. parallel composition and hiding, an IMC can be reduced to its quotient IMC w.r.t these equivalences. Chapter 3. describes the syntax and the operational semantics based on Timed Probabilistic System (TPS) of the Scenario-Aware Dataflow model. Based on the adapted SADF model and IMC model, we represent a framework to define SADF graphs in terms of IMCs. We give each components in a SADF graph a corresponding IMC, and then to construct the whole SADF graph by using parallel composition and hiding. In Chapter 4., the implementation of the framework is presented by using the toolset CADP developed by the VASY team at INRIA Rhone-Alpes. We studied two examples of SADF models by discussing the issues encountered during the state space generation, checking the functional properties and evaluating some performance properties. In the last chapter, we summarize our work and point out some challenges in future, which may happen when inherent non-determinism is present in SADF spification.

Chapter 2

Interactive Markov Chains

The process algebra (PA) transformed from the theory of algebra in mathematics to computer science finds a sound and effective framework to formally model and analyze of concurrent systems. In this chapter, we will introduce a stochastic process algebra called IML and its mapping semantic model interactive Markov chain (IMC). Beyond the classical process algebras like CCS, CSP, or LOTOS, the big advantage using IMC as semantic models is the combination of classical process algebras and continuous time Markov chains. This combination allows us not only to check the functional properties of systems, but also evaluate some performance metrics. We first give an overview of the classical process algebra CCS to see how it can be used to model the interactive concurrent systems, and then we will introduce the interactive Markov chains, which extends on the essential ideas of classical process algebras. The Figure 2.1 shows the basic concept of stochastic process algebra used in modeling and analyzing [23] phases.

2.1 Classical Process Algebras

For a better understanding of the idea behind process algebra, we give a short introduction of a classical process algebra CCS. The CCS was developed by Robin Milner and he published his research on CCS in his book in 1980 [33]. Here, we assume there is a countable set N of action names and let $\overline{N} := \{\overline{a} \mid a \in N\}$ denote the set of co-names. A special action is called the *silent* or *unobservable* action denoted as τ . We defined the set of *actions* as $Act := N \cup \overline{N} \cup \{\tau\}$ and let P_{id} be the set of *process identifiers*.

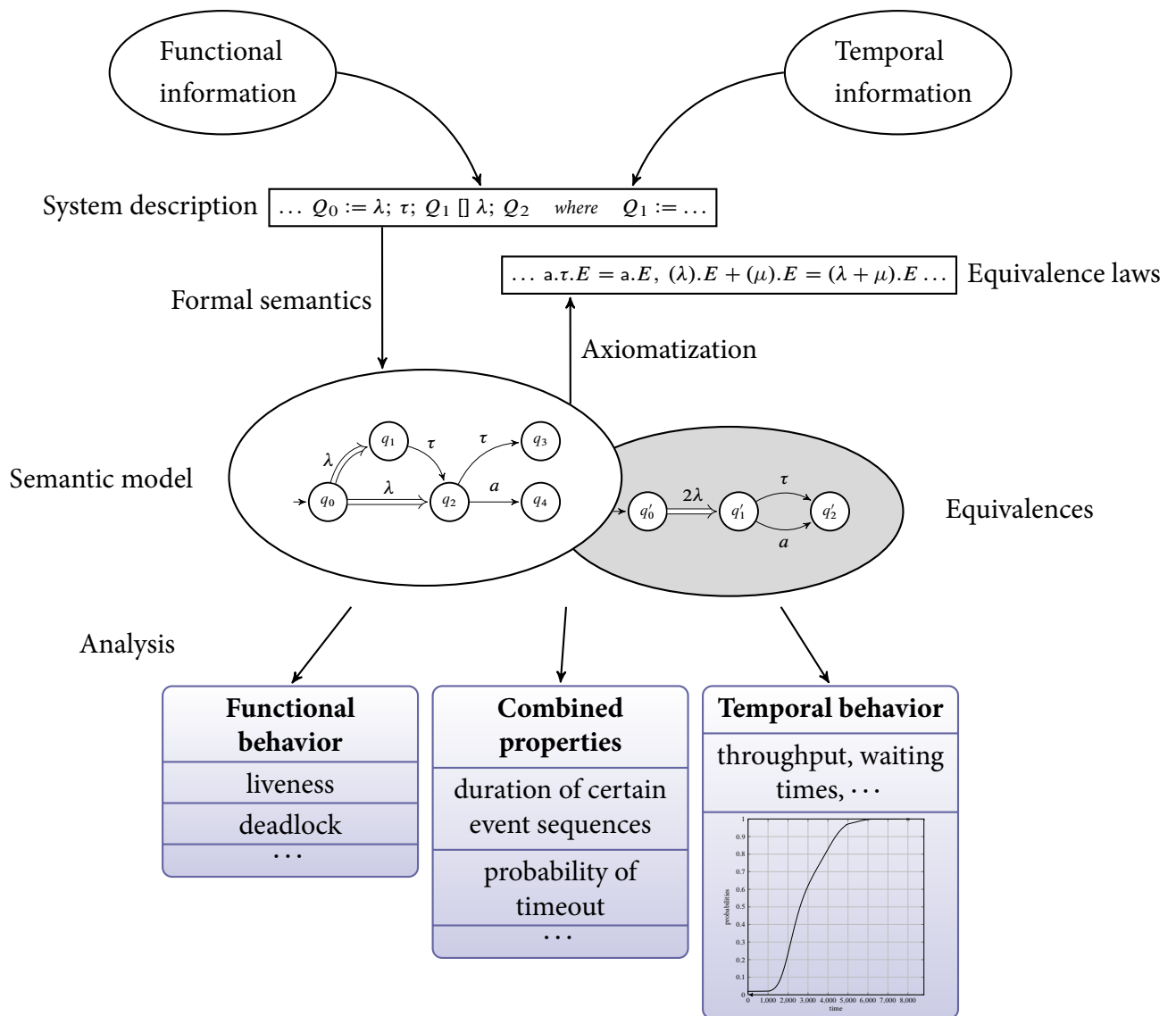


Figure 2.1: The basic concept of stochastic process algebra in modeling and analysis. [23]

2.1.1 Syntax and Semantics of CCS

Definition 2.1 Syntax of CCS Let $\alpha \in Act$, $a, a_i \in N$ and $A \in P_{id}$. We define the set *Prc* of (concurrent) process expressions by following syntax:

$$P ::= \text{nil} \mid \alpha.P \mid P_1 + P_2 \mid P_1 \parallel P_2 \mid \text{new } a P \mid A(a_1, \dots, a_n)$$

The intuitive meaning of CCS constructs are listed as follows:

Inaction	The symbol nil describes a process that can do nothing. The process can neither interact with other process nor perform internal actions.
Action prefixing	The expression $\alpha.P$ means the process can perform a α action and then behaves as process P . The operator “.” between α and P is a binary operator and called the <i>prefix</i> operator.
Choice	The expression $P_1 + P_2$ describes a non-deterministic choice between P_1 and P_2 . The symbol “+” is called the <i>choice</i> operator.
Parallel composition	The execution of parallel composition of process P_1 and P_2 ($P_1 \parallel P_2$) can involve interleaving or synchronization (communication). The semantics of this operation depends on the next execution possibility of these two processes. In the action set of process, actions are refined into three classes. The actions $a \in A$ are interpreted as input actions and its complementary action $\bar{a} \in \bar{N}$ is called output actions. If one process can perform an a -action and the other process can perform the complementary action \bar{a} of a , then the parallel composition of the two processes perform an synchronization action τ based on the merge of the a and \bar{a} .
Restriction	The new a declares that the action a is not visible outside the process P .
Process call	A (recursive) process definition is an defining equation system of the form $A_i(\vec{a}_i) = P_i \mid 1 \leq i \leq k$ where $k \geq 1$, $\vec{a}_i = a_{i1}, \dots, a_{in_i}$, $A_i \in P_{id}$ ($A_i = A_j$, if only if $i = j$), $a_{ij} \in N$, and $P_i \in Prc$. So the process call of $A(a_1, \dots, a_n)$ means the process behaves as the right-hand side of the corresponding equation system with replacing the formal name parameter with a_1, \dots, a_n .

The semantics of CCS is defined by using Structural Operational Semantics (SOS) developed by Gordon D. Plotkin [37, 38].

<p>Act $\frac{}{\alpha.P \xrightarrow{\alpha} P}$</p> <p>Sum₁ $\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$</p> <p>Par₁ $\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$</p> <p>New $\frac{P \xrightarrow{\alpha} P' \quad \alpha \notin \{a, \bar{a}\}}{\text{new } a P \xrightarrow{\alpha} \text{new } a P'}$</p>	<p>Com $\frac{P \xrightarrow{\varphi} P' \quad Q \xrightarrow{\bar{\varphi}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$</p> <p>Sum₂ $\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$</p> <p>Par₂ $\frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$</p> <p>Call $\frac{A(\vec{a}) = P \quad P[\vec{a} \mapsto \vec{b}] \xrightarrow{\alpha} P'}{A(\vec{b}) \xrightarrow{\alpha} P'}$</p>
---	--

This formal semantics of CCS provides us a way to interpret the process of CCS in terms of LTS, i.e., we can map every expression in CCS to its “equivalent” LTS.

Definition 2.2 (Labelled transition system). A labelled transition system is a triple $\mathcal{I} = (S, Act, \longrightarrow)$ where

- S is a nonempty set of states,
- Act is a set of actions, and
- $\longrightarrow \subseteq S \times Act \times S$ is a set of a transition relation (sometimes also called interactive transitions).

Conventionally, we abbreviate $(s, \alpha, s') \in \longrightarrow$ as $s \xrightarrow{\alpha} s'$. Note that an LTS does not specify any initial state(s) or a final (accepting) state in comparison with an automaton. The reason for dropping the final states is: in interactive systems, it is not important whether a sequence of actions can make the system to reach the final states but whether the system can perform the sequence of actions successfully or not [33]. Usually we also do not specify the initial state(s), and we can assume any state as initial state and get the whole image of the model.

Practically, we can also specify the initial state as s_0 and then the LTS is called a *rooted* LTS [26] defined as a quadruple $\mathcal{L} = (S, Act, \longrightarrow, s_0)$.

Example 2.1 (Parallel composition of two one-place buffers). An one-place buffer and the parallel composition of two one-place buffers can be defined in CCS as:

$$\begin{aligned} B(in, out) &= in.\overline{out}.B(in, out) \\ B_{\parallel}(in, out) &= new\ com\ (B(in, com) \parallel B(com, out)) \end{aligned}$$

The corresponding (rooted) LTS to represent the parallel composition of two one-place buffers is shown in Figure 2.2.

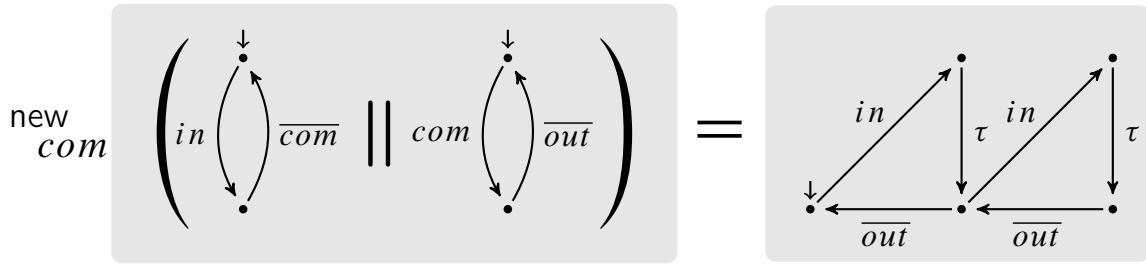


Figure 2.2: The (rooted) LTS representing the parallel composition of two one-place buffers

2.1.2 Stochastic Process Algebra

Classical process algebras are insufficient sometimes to model the systems in real world, and the models generated by classical process algebras are more suitable for functional verification. For performance evaluation, to add time information into the modeling is needed. One way is to use special time transitions to represent the advancement of time, e.g., timed probabilistic system (TPS). Another way is use the continuous distributions (general distributions or non-negative exponential distribution) to model the time passing in the system. Further, besides the interactive transitions and the transitions to represent the time passing using in the modeling, sometimes, probabilistic transitions are involved into the semantic models (like TPS, probabilistic and Markovian transition system (PMTS) [39] and probabilistic automata in continuous time [16]) to model the stage of doing probabilistic decision. On the other hand, the particular important class of stochastic model Markov chains lacks of some functional capabilities, especially to glue the subsystems together to construct a large system. As a result, many approaches are presented to form an especially kind of process algebra: stochastic process algebra (SPA). Further, the SPA can be divides into two categories: probabilistic process algebra and stochastic process algebra.

2.2 Interactive Markov Chains (IMCs)

Holger Hermanns introduced a new stochastic semantic model called interactive Markov chains (IMCs) with its corresponding algebra IML in his book [19]. This semantic framework integrates Markov Chains and LTS in an orthogonal manner, and provides not only the possibility to check the functional correctness, but also some temporal aspects of the system. We first look at some essential ideas behind the integration of Markov Chains and LTS.

2.2.1 Essentials behind IMC

The advantages of Exponential distribution. There are many kinds of distribution functions used for modeling the stochastic systems. But the more precisely the distribution is used the more complex the model will be during generation and analysis. One distribution (function) receives more attention than the others is called *exponential distribution*. The reason is that exponential distribution is the only continuous distribution function which fulfills the memoryless property and allows us relatively simple to construct the model and analyze the model systemically [25]. This memoryless nature of time in the Markovian approach makes it possible to avoid the explicit representation of time passage in system specifications [9].

Definition 2.3 (Distribution Function). *The distribution function F_x of a random variable X is defined to be the function*

$$F_X(x) = Pr(X \leq x)$$

If the random variable X is continuous, then we have

$$F_x(x) = \int_0^x f(t)dt$$

where $f(x)$, the derivative of $F_x(x)$, is called the probability density function for the random variable X .

Definition 2.4 (Exponential distribution) *The distribution and density functions of ex-*

ponential distribution are given by following relations:

$$\begin{aligned} F_X(x) &= Pr(X \leq x) = 1 - e^{-\lambda x} : (x \geq 0) \\ f_X(x) &= \frac{d}{dx} F_X(x) = \lambda \cdot e^{-\lambda x} \\ E[X] &= \int_0^{\infty} t \cdot f_X(t) dt = \frac{1}{\lambda} \end{aligned}$$

The memoryless property holds for the exponential distribution is expressed by the following equation:

$$f_X(x + y \mid x \geq y) = f_X(x)$$

Separation of delays and actions. When both interaction and delay (governed by a exponential distribution) information are include in one action in the Markovian process algebras (with the transition form (a, λ)), the issue reveals when we need to know the time consumption in case of interaction [27]. The delay of synchronization between action (a, λ) and (a, μ) is no longer exponentially distributed. A natural solution here in IMC is to separate the interaction and delay information in actions in order to let the synchronizations happen only via immediate actions. The semantics of action prefixing operator in the classic process algebras will only extended to the case of delay actions (cf. Section 2.2.2).

Exponential distributions and process algebra. To integrate the exponential distributions into process algebra is feasible, since the exponential distributions and interleaving semantics in process algebra fit well together [27]. In classic process algebras, the parallel composition operator “ \parallel ” can be reduced by using choice operator “ $+$ ” and prefix operator “ \cdot ” according to the *expansion law*. Assume $a, b \in Act$ and $P, Q \in Prc$, then according to expansion law we have:

$$a.P \parallel b.Q = a.(P \parallel b.Q) + b.(a.P \parallel Q).$$

Now, the problem is how the situation changes when we replace these normal actions with the delay actions, and is the expansion law still valid? We demonstrate with a simple example: $\lambda.P \parallel \mu.Q$. Here, $\lambda.P$ means the process with behavior like P after an exponential delay with mean $1/\lambda$ time units, and $\mu.Q$ analogously. Let U be the random variable before process Q can start, then obviously U is exponentially distributed with rate μ , and assume the delay of the process $\lambda.P$ finishes earlier than $\mu.Q$ after y time units. Then, the probability that the process Q has to delay for at most another x time units is

$$Pr[U \leq y + x \mid U > y].$$

Since the exponential distribution is the only continuous distribution function which fulfills the memoryless property, we have

$$Pr[U \leq y + x \mid U > y] = Pr[U \leq x].$$

This equation tells us the remaining duration (residual sojourn time) until the initial delay of process $\mu.Q$ is independent of how the other process (i.e., the process $\lambda.P$) behaves. The advance of time governed by memoryless distribution is independent [27]. Analogously, for the delay of $\lambda.P$ first finishes its execution has also no impact on the remaining finishing time of process $\mu.Q$. The conclusion is the expansion law holds also for the parallel composition of Markovian transitions (this property is illustrated in Figure 4.13(a)). But for general distribution, the law is generally invalid [27].

Internal actions and hiding. From the view point of an observer outside the system, all the communications (synchronization) in the system should be invisible. The only possibility to interact with the system is only the interfaces provided by the system to communicate with outside, and the inside of the system should be seen as a “black box”. As a result, if we want to investigate the execution of system only from an observer’s point of view, we need to specify the actions happened inside the system into internal actions. On the other hand, the synchronization should cost no time, it finishes immediately. This assumption is achieved by hiding these actions in process algebra using a specific operator (this operator turns the transitions labelled with specified actions into τ -transitions). In IMC, the interactive transitions that can not be delayed by the synchronization with other actions should be turned into τ -transition.

Maximal progress assumption. The transitions for synchronization (synchronization between subsystems inside the system) cost no time, while the probability of Markovian transitions to finish immediately is $(1 - e^{-\lambda \cdot 0} = 0)$. Therefore, the assumption known as *maximal progress* is made on the IMCs, which says if both internal transition (τ -transition) and Markovian transition can emanated from one state, the Markovian transition is simply blocked, since it can not finish immediately.

2.2.2 Algebra of Interactive Markov Chains – IML

In this section we introduce the algebra IML of IMC [19]. The syntactic descriptions of IMC is a set of expressions (language) called IML. To define the language IML, we assume \mathcal{V} denotes a countable set of variables expressing repetitive behavior, the set Act denotes a set of actions including a distinguished internal actions τ . Traditionally, we use λ, μ, \dots to range over \mathbb{R}^+ (the set of nonnegative reals), and a, b, \dots for elements of Act .

Definition 2.5 (The language IML of IMC) Let $\lambda \in \mathbb{R}^+$, $a \in Act$ and $X \in \mathcal{V}$, the language IML is defined as the set of expressions by following syntax:

$$\mathcal{E} ::= 0 \quad | \quad a.\mathcal{E} \quad | \quad (\lambda).\mathcal{E} \quad | \quad \mathcal{E} + \mathcal{E} \quad | \quad X \quad | \quad \underline{x ::= \mathcal{E}} \quad | \quad \perp$$

The intuitive meaning of constructs in the expressions in IML are listed as follows:

- Inaction (0)** The terminal symbol 0 describes a *terminated* behavior. Neither engagement in any interaction nor to perform internal actions are possible.
- Action prefixing (a.E)** The expression $a.E$ can perform an interaction on action a and then behaves as expression E . The operator “.” between a and E is a binary operator and called the *prefix* operator.
- Delay prefixing ($\lambda.E$)** The expression $\lambda.E$ describes a behavior that will behaves as expression E after a delay governed by an exponential distributions with a mean duration of $1/\lambda$ time units.
- Choice ($E + F$)** The expression $E + F$ describes a non-deterministic or probabilistic choice (if E and F are delay prefixed) between expressions E and F . The symbol “+” is called the *choice* operator.
- Recursion ($x ::= E$)** The expression $\underline{x ::= E}$ describes a recursively defined behavior: if the variable X appears somewhere inside the expression E , and during the evolution of the expression, whenever we encounter the variable X , we reinitialise its behavior to $\underline{x ::= E}$.
- Ill-defined behavior (\perp)** The symbol \perp represents an ill-defined behavior.

Example 2.2 An example of an IML expression and its equivalent IMC are

$$E_1 ::= a.(\lambda).b.(\mu).0$$



To map the complete language IML onto IMCs in a compositional manner, the semantic of each expression of IML is again defined by means of SOS.

$\frac{}{a.E \xrightarrow{a} E}$ $\frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'}$ $\frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'}$ $\frac{E\{x ::= E / X\} \xrightarrow{a} E'}{x ::= E \xrightarrow{a} E'}$	$\frac{}{(\lambda).E \xRightarrow{\lambda} E}$ $\frac{E \xRightarrow{\lambda} E'}{E + F \xRightarrow{\lambda} E'}$ $\frac{F \xRightarrow{\lambda} F'}{E + F \xRightarrow{\lambda} F'}$ $\frac{E\{x ::= E / X\} \xRightarrow{\lambda} E'}{x ::= E \xRightarrow{\lambda} E'}$
---	---

Table 2.1: Operational semantic rules of IML

Definition 2.6 *The action transition relation $\longrightarrow \subset \text{IML} \times \text{Act} \times \text{IML}$ is the least relation and the Markovian transition relation $\Longrightarrow \subset \text{IML} \times \mathbb{R}^+ \times \text{IML}$ is the least multi-relation given by the SOS rules above, where $E\{F/X\}$ denotes simultaneous substitution of each occurrence of variable X inside expression E by expression F .*

Definition 2.7 (Closed and open expressions of IML). *A variable X occurs free w.r.t. an expression E if it occurs inside E outside the scope of any binding $\underline{x} ::= \dots$. Let $\mathcal{V}(E) = \emptyset$ denote the set of variables that occur free in an expression E . If $\mathcal{V}(E) = \emptyset$, then we say the expression E is closed, otherwise it is open.*

Definition 2.8 *The set of well-defined expressions IML_\downarrow is the smallest subset of IML such that*

- $\mathcal{V} \subseteq \text{IML}_\downarrow$ and $0 \in \text{IML}_\downarrow$,
- if $E \in \text{IML}$, then $a.E \in \text{IML}_\downarrow$ and $(\lambda).E \in \text{IML}_\downarrow$,
- if $E \in \text{IML}$, and $F \in \text{IML}_\downarrow$, then $E + F \in \text{IML}_\downarrow$,
- if $E\{x ::= E / X\} \in \text{IML}_\downarrow$, then $\underline{x} ::= E \in \text{IML}_\downarrow$.

The complementary set containing all ill-defined expressions will be denoted IML_\uparrow .

2.2.3 Interactive Markov Chains

The same as the mapping relation from CCS to LTS, the expressions IML can also map to the semantic model IMC which is an extension of (rooted) LTS.

Definition 2.9 (Interactive Markov chain). An interactive Markov chain is a tuple $\mathcal{I} = (S, Act, \longrightarrow, \Longrightarrow, s_0)$ where

- S is a nonempty set of states with initial state $s_0 \in S$,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is a set of interactive transitions, and
- $\Longrightarrow \subseteq S \times \mathbb{R}^+ \times S$ is a set of Markovian transitions.

We write $s \xrightarrow{\alpha} s'$, if $(s, \alpha, s') \in \longrightarrow$ and $s \xrightarrow{\lambda} s'$, if $(s, \lambda, s') \in \Longrightarrow$. The states in an IMC are divided into three categories. Let $IT(s) = \{s \xrightarrow{\alpha} s'\}$ denotes the set of interactive transitions that leave s , and $MT(s) = \{s \xrightarrow{\lambda} s'\}$ denotes the set of Markovian transitions that leave s . Then, we say a state s is a

Markovian state , iff $IT(s) = \emptyset$ and $MT(s) \neq \emptyset$;

interactive state , iff $MT(s) = \emptyset$ and $IT(s) \neq \emptyset$;

hybrid state , iff $MT(s) \neq \emptyset$ and $IT(s) \neq \emptyset$;

deadlock state , iff $MT(s) = \emptyset$ and $IT(s) = \emptyset$.

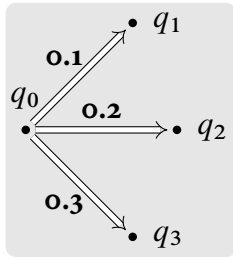
The set of Markovian states is denoted as $MS \subseteq S$, and the set of interactive states is denoted as $IS \subseteq S$ in the IMC \mathcal{I} . Obviously, an LTS is an IMC with all states in it are interactive, i.e., $MT(s) = \emptyset$ for all state s . A CTMC is an IMC with all states in it are Markovian, i.e., $IT(s) = \emptyset$ for all state s . Therefore, IMC is the superset of both CTMC and LTS.

The semantics of a Markovian transition in an IMC is described as follows: a Markovian transition $s \xrightarrow{\lambda} s'$ means the probability to move from state s to s' in d time units is

$1 - e^{-\lambda \cdot d}$, where λ is called the *rate*¹ of the Markovian transition. We denote $\mathbf{R}(s, s') = \sum \{\lambda \mid s \xrightarrow{\lambda} s'\}$ as the rate to move from state s to state s' , if $s \in \text{MS}$. If there exists more than one state s' , such that $\mathbf{R}(s, s') > 0$, then these successor states of s will compete for the successful transition from s . This competition between the successor states of s is known as *race condition*. Let $E(s) = \sum_{s' \in S} \mathbf{R}(s, s')$ be the *exit rate* of state s . Then the probability that for the particular state s' to win the competition within d time units is given by:

$$\frac{\mathbf{R}(s, s')}{E(s)} \cdot (1 - e^{-E(s) \cdot d}).$$

Example 2.3 An example of race condition in an IMC is shown in Figure 2.3.



$$\begin{aligned} \mathbf{R}(q_0, q_1) &= 0.1 \\ \mathbf{R}(q_0, q_2) &= 0.2 \\ \mathbf{R}(q_0, q_3) &= 0.3 \\ E(q_0) &= 0.1 + 0.2 + 0.3 = 0.6 \\ \mathbf{P}(q_0, q_1) &= \frac{\mathbf{R}(q_0, q_1)}{E(q_0)} = \frac{1}{6} \\ \mathbf{P}(q_0, q_2) &= \frac{\mathbf{R}(q_0, q_2)}{E(q_0)} = \frac{1}{3} \\ \mathbf{P}(q_0, q_3) &= \frac{\mathbf{R}(q_0, q_3)}{E(q_0)} = \frac{1}{2} \end{aligned}$$

The probabilities to move from state q_0 to its successor states within $z \in \mathbb{R}_{\geq 0}$ time units:

$$\begin{aligned} \mathbf{P}'(q_0, q_1) &= \frac{1}{6} \cdot (1 - e^{-0.6 \cdot z}) \\ \mathbf{P}'(q_0, q_2) &= \frac{1}{3} \cdot (1 - e^{-0.6 \cdot z}) \\ \mathbf{P}'(q_0, q_3) &= \frac{1}{2} \cdot (1 - e^{-0.6 \cdot z}) \end{aligned}$$

Figure 2.3: A race condition in an IMC

As aforementioned, if an interactive transition can not be delayed by any synchronization caused by other actions, it can be turned into τ -transitions. Since these τ -transitions can be made instantaneously, whereas the probability for any Markovian transition to be made in zero time unit is zero, the maximal progress assumption is defined in IMC as follows.

Definition 2.10 (Maximal progress). In any IMC, internal interactive transitions (τ -transitions) take precedence over Markovian transitions [20].

Definition 2.11 (Parallel composition). Let $\mathcal{I}_1 = (S_1, Act_1, \longrightarrow_1, \Longrightarrow_1, s_{0,1})$ and $\mathcal{I}_2 = (S_2, Act_2, \longrightarrow_2, \Longrightarrow_2, s_{0,2})$ be two IMCs. The parallel composition of \mathcal{I}_1 and \mathcal{I}_2 w.r.t. an action set A is defined by:

$$\mathcal{I}_1 \parallel_A \mathcal{I}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \longrightarrow, \Longrightarrow, (s_{0,1}, s_{0,2}))$$

where \longrightarrow and \Longrightarrow are defined as the smallest relations satisfying

¹The same word rate is used in SADF specification later, but with different meanings.

1. $s_1 \xrightarrow{\alpha}_1 s'_1$ and $s_2 \xrightarrow{\alpha}_2 s'_2$ and $\alpha \in A, \alpha \neq \tau$ implies $(s_1, s_2) \xrightarrow{\alpha} (s'_1, s'_2)$
2. $s_1 \xrightarrow{\alpha}_1 s'_1$ and $\alpha \notin A$ implies $(s_1, s_2) \xrightarrow{\alpha} (s'_1, s_2)$ for any $s_2 \in S_2$
3. $s_2 \xrightarrow{\alpha}_2 s'_2$ and $\alpha \notin A$ implies $(s_1, s_2) \xrightarrow{\alpha} (s_1, s'_2)$ for any $s_1 \in S_1$
4. $s_1 \xRightarrow{\lambda}_1 s'_1$ implies $(s_1, s_2) \xRightarrow{\lambda} (s'_1, s_2)$ for any $s_2 \in S_2$
5. $s_2 \xRightarrow{\lambda}_2 s'_2$ implies $(s_1, s_2) \xRightarrow{\lambda} (s_1, s'_2)$ for any $s_2 \in S_2$

The rule of parallel composition of IMCs follows the (forced-to-synchronize) manner used in TCSP [21]. This forced manner only has effect on the interactive action synchronization (since the total separation of interaction and delay actions) between the two IMCs (\mathcal{I}_1 and \mathcal{I}_2), i.e., the action in the synchronization set A (excluding the τ -action) must be performed by both IMCs simultaneously. This also means, if somewhere one IMC can only perform actions in the set A , it must wait for another IMC to reach the state where these actions in set A can be synchronized. If the actions are not in the set A , then IMC \mathcal{I}_1 and \mathcal{I}_2 can perform the actions either \mathcal{I}_1 or \mathcal{I}_2 can perform. To treat the delay actions in the parallel composition, the last two rules indicate that the IMCs can delay independently due to the memoryless property of exponential distributions (cf. Section 2.2.1). The residual sojourn time of one process is (again) exponentially distributed with its original rate after another process finishes before it.

Definition 2.12 (Hiding). *The hiding of IMC $\mathcal{I} = (S, Act, \xrightarrow{\cdot}, \xRightarrow{\cdot}, s_0)$ w.r.t. the set H of actions is the IMC $\mathcal{I} \setminus H = (S, Act \setminus H, \xrightarrow{\cdot}', \xRightarrow{\cdot}, s_0)$ where $\xrightarrow{\cdot}'$ is the smallest relation defined by:*

1. $s \xrightarrow{\alpha} s'$ and $\alpha \notin H$ implies $s \xrightarrow{\alpha}' s'$, and
2. $s \xrightarrow{\alpha} s'$ and $\alpha \in H$ implies $s \xrightarrow{\tau}' s'$.

The effect of applying Hiding on an IMC is to turn the transitions labelled by the actions in the set H into τ -transitions. Together with maximal progress assumption, the semantics of original IMC will be changed. Since the internal interactive transitions (τ -transitions) happen before the Markovian transitions, the Markovian transitions which compete with internal interactive transitions from the states are simply blocked.

2.3 The Equivalence Notions on IMC

In this section, we will introduce three major equivalence notions on IMC, i.e., strong bisimulation, weak bisimulation, and stochastic branching bisimulation. These equivalences are adapted from the equivalence notions in classical process algebra, and because of the presence of Markovian transitions in IMC, the lumpability defined on the Markov chains is also transformed into the case of IMC.

Before we define these notions on IMC, we start with some definitions. For set $C \subseteq S$ and state s , let $\mathbf{R}(s, C) = \sum_{s' \in C} \mathbf{R}(s, s')$, let $s \xrightarrow{\tau}$ be a boolean, which is evaluated true iff s has no outgoing τ -transitions. For state s , action α and $C \subseteq S$, let $\mathbf{T}(s, \alpha, C) = 1$ iff $\{s' \in C \mid s \xrightarrow{\alpha} s'\}$ is non-empty, and let $\mathbf{W}(s, \alpha, C) = 1$ iff $\{s' \in C \mid s \xrightarrow{\tau^*} \xrightarrow{\alpha} \xrightarrow{\tau^*} s'\}$ is non-empty.

2.3.1 Strong Bisimulation

Definition 2.13 (Strong bisimulation). *Let an IMC $\mathcal{I} = (S, Act, \longrightarrow, \Longrightarrow, s_0)$. An equivalence relation $R \subseteq S \times S$ is a strong bisimulation on \mathcal{I} if for any $(s, t) \in R$ and equivalence class $C \in S/R$ the following holds:*

1. for any $\alpha \in Act$, $\mathbf{T}(s, \alpha, C) = \mathbf{T}(t, \alpha, C)$, and
2. $s \xrightarrow{\tau}$ implies $\mathbf{R}(s, C) = \mathbf{R}(t, C)$.

States s and s' are called strongly bisimilar, written $s \sim s'$, if there exists a strong bisimulation R such that $(s, s') \in R$.

Strong bisimulation makes sure that the strongly bisimilar states, say s and t , must have the same potential to perform every possible interactive actions and additionally the cumulative rate $\mathbf{R}(s, C) = \mathbf{R}(t, C)$. The cumulative rate equality can be matched with the definition of lumpability from Markov chain [11].

2.3.2 Weak bisimulation

Usually the strong bisimulation equivalence is too fine to distinguish the systems, especially in some abstraction when we need to treat some components in the system

as “black boxes” and from the observer’s point of view, these internal transitions in the “black boxes” should be hidden, i.e., the observer can not distinguish several τ -transitions (denoted as $\xrightarrow{\tau^*}$) or a single one $\xrightarrow{\tau}$ (they are in some sense “equivalent” to the observer). Conventionally these hidden internal transitions are turned into the transitions which are specially with τ labelled. Therefore, these τ -transitions play a special role in IMCs [20]. But for Markovian transitions, the probabilistic distribution of a sequence of exponential distributions is not again an exponential distribution but constitutes a phase-type distribution [20], this observation can not be applied to the Markovian transitions. The $\xrightarrow{\tau^*}$ and $\xrightarrow{\tau}$ are treat as equal in interactive processes is not applicable here. The solution here is to force that the Markovian transitions in IMCs have to be mimicked in the strong sense (defined in the strong bisimulation equivalence).

Definition 2.14 (Weak bisimulation). *Let an IMC $\mathcal{I} = (S, Act, \longrightarrow, \Longrightarrow, s_0)$. An equivalence relation $R \subseteq S \times S$ is a weak bisimulation on \mathcal{I} if for any $(s, t) \in R$ and equivalence class $C \in S/R$ the following holds:*

1. for any $\alpha \in Act$, $\mathbf{W}(s, \alpha, C) = \mathbf{W}(t, \alpha, C)$, and
2. $s \xrightarrow{\tau^*} s'$ and $s' \not\xrightarrow{\tau}$ implies $t \xrightarrow{\tau^*} t'$ and $t' \not\xrightarrow{\tau}$ and $\mathbf{R}(s', C) = \mathbf{R}(t', C)$, for some $t' \in S$.

States s and s' are called weakly bisimilar, written $s \approx s'$, if there exists a weak bisimulation R such that $(s, s') \in R$.

Two IMCs \mathcal{I}_1 with state space S_1 , \mathcal{I}_2 with state space S_2 , and $S_1 \cap S_2 = \emptyset$ are said to be strongly (weakly) bisimilar, denoted $\mathcal{I}_1 \sim (\approx) \mathcal{I}_2$, if there exists a strong (weak) bisimulation R on $S_1 \cup S_2$ such that $(s_{0,1}, s_{0,2}) \in R$.

2.3.3 Stochastic Branching Bisimulation

From the definitions of strong bisimulation and weak bisimulation, one problem reveals when we want to test equivalences on IMCs. The equivalence of strong bisimulation is too fine, but the weak bisimulation is too coarse. Consider the following two IMCs (LTSs) showing in Figure 2.4. The two IMCs are not strongly bisimilar but weakly bisimilar. But if we observe them more carefully, they are slightly different. The right IMC \mathcal{I}_2 can

move by the transition b into the class $\textcircled{\text{||||}}$ without passing through the class $\textcircled{\text{|||}}$. In contrast to \mathcal{I}_2 , in the left IMC \mathcal{I}_1 , every state in class $\textcircled{\text{||||}}$ to the class $\textcircled{\text{||||}}$ is forced to pass the class $\textcircled{\text{|||}}$. As a result, for the right IMC \mathcal{I}_1 , the decision of which b to take will affect the later possibility of execution (i.e., can perform an a -action or not), but for the left one, both b -transitions will lead to the state, which can make either a -transition or τ -transition. As the example shows, we need a finer equivalence relation to take the distinction into account.

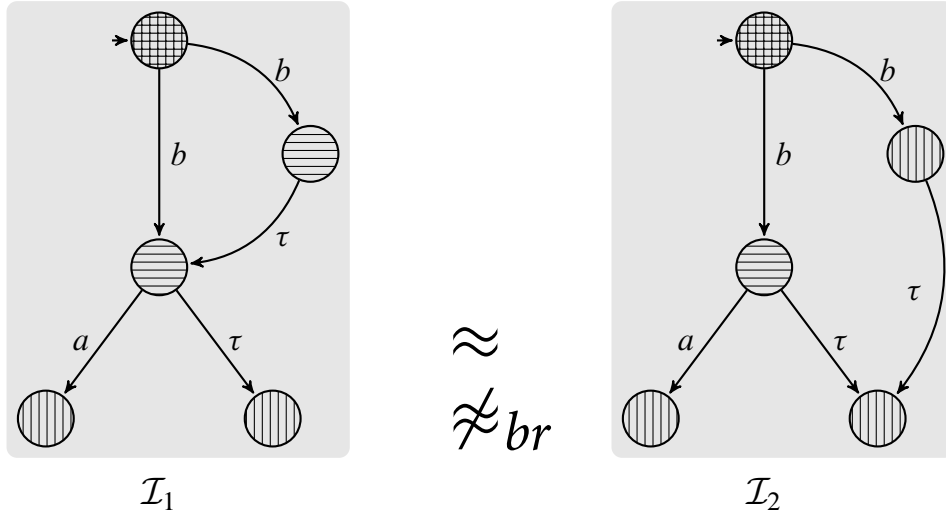


Figure 2.4: The difference between weak and branching bisimulation [19]

Definition 2.15 (Stochastic branching bisimulation). Let an IMC $\mathcal{I} = (S, Act, \longrightarrow, \Longrightarrow, s_0)$, and equivalence relation $R \subseteq S \times S$ is a stochastic branching bisimulation on \mathcal{I} if for any $(s, t) \in R$ and equivalence class $C \in S/R$, the following holds:

1. $s \xrightarrow{\alpha} s'$ implies $\left\{ \begin{array}{l} \text{either } \alpha = \tau \text{ and } (s', t) \in R \text{ or} \\ \exists \textcircled{t}, t' \in S : t \xrightarrow{\tau^*} \textcircled{t} \xrightarrow{\alpha} t' \wedge (s, \textcircled{t}) \in R \wedge (s', t') \in R, \end{array} \right.$
- and
2. $s \xrightarrow{\tau} \textcircled{s}$ implies $t \xrightarrow{\tau^*} t' \xrightarrow{\tau} \textcircled{t} : \mathbf{R}(s, C) = \mathbf{R}(t', C)$ for some $t' \in S$.

States s and s' are called stochastic branching bisimilar, written $s \approx_{br} s'$, if there exists a stochastic branching bisimulation R such that $(s, s') \in R$.

The basic idea of branching bisimulation is: if a step $P \xrightarrow{a} P'$ is simulated by $Q \xrightarrow{\tau^*} \xrightarrow{a} \xrightarrow{\tau^*} Q'$, all states reached by the τ -transitions after the a -transition should be the same class of Q' . And the Markovian transitions have to be mimicked in the strong sense as usual.

Two IMCs \mathcal{I}_1 with state space S_1 , \mathcal{I}_2 with state space S_2 , and $S_1 \cap S_2 = \emptyset$ are said to be stochastic branching bisimilar, denoted $\mathcal{I}_1 \approx_{br} \mathcal{I}_2$, if there exists a stochastic branching bisimulation R on $S_1 \cup S_2$ such that $(s_{0,1}, s_{0,2}) \in R$.

The following lemma expresses the interrelation between these three equivalence relations:

Lemma 2.1 $\sim \subset \approx_{br} \subset \approx$

With the help of these equivalence relations defined on IMC, we can reduce the state space of the given IMC by partitioning the states into certain classes according to these equivalence relations, in other words, we can always obtain an “equivalent” IMC when the original IMC modulo the bisimulation equivalences. For equivalence relation R , on state space S and $s \in S$, let $[s]_R$ denote the equivalence class of s under R , and let $S/R = \{ [s]_R \mid s \in S \}$ denote the quotient space of S under R .

Theorem 2.1 $\sim, \approx, \approx_{br}$ are congruences w.r.t. parallel composition and hiding [19], i.e.,

$$\begin{array}{c}
 \begin{array}{ccc}
 \sim & & \sim \\
 \mathcal{I}_1 \approx_{br} \mathcal{I}_2 \text{ implies} & \mathcal{I}_1 \parallel_A \mathcal{I}_3 \approx_{br} & \mathcal{I}_2 \parallel_A \mathcal{I}_3
 \end{array} \\
 \hline
 \begin{array}{ccc}
 \sim & & \sim \\
 \mathcal{I}_1 \approx_{br} \mathcal{I}_2 \text{ implies} & \mathcal{I}_3 \parallel_A \mathcal{I}_1 \approx_{br} & \mathcal{I}_3 \parallel_A \mathcal{I}_2
 \end{array} \\
 \hline
 \begin{array}{ccc}
 \sim & & \sim \\
 \mathcal{I}_1 \approx_{br} \mathcal{I}_2 \text{ implies} & \mathcal{I}_1 / H \approx_{br} & \mathcal{I}_2 / H
 \end{array}
 \end{array}$$

Definition 2.16 (Quotient IMC). Let $\mathcal{I} = (S, Act, \longrightarrow, \Longrightarrow, s_0)$ be an IMC and R a strong bisimulation on S . The quotient IMC $\mathcal{I}/R = (S/R, Act, \longrightarrow', \Longrightarrow', [s_0]_R)$ where \longrightarrow' and \Longrightarrow' are the smallest relations satisfying:

1. $s \xrightarrow{\alpha} s'$ implies $[s]_R \xrightarrow{\alpha} [s']_R$, and
2. $s \xrightarrow{\lambda} s'$ implies $[s]_R \xrightarrow{\mathbf{R}(s, [s']_R)} [s']_R$.

The definition tells us for any IMC \mathcal{I} and strong bisimulation it holds that $\mathcal{I} \sim \mathcal{I}/R$. This is also true when we replace \sim with \approx or \approx_{br} . The algorithms to compute the quotient IMC according to strong, weak, or stochastic branching bisimulation in the CADP toolset is describe in [22].

In [28], it is proven that bisimulation minimization facilitates the speed of performance evaluation. Now, if we compute the quotient IMC of a given IMC and use this quotient IMC instead in our analysis, we can also benefit from this approach. The problem here is only the algorithm to compute the quotient IMC is not on-the-fly [20]. We need first to generate the whole state space of the IMC and then compute its quotient IMC. If the direct generation of state space of the IMC fails, i.e., the state space of the IMC is too large to generate, then we will not benefit from this approach. However we know that an IMC may consist of parallel compositions of several IMCs and the bisimulation equivalence is a congruence w.r.t parallel composition, these two properties of IMC implies the divide and conquer algorithm in the IMC state space generation phase may be efficient to solve the state explosion problem and avoid unmanageable state space. Let $\widehat{\mathcal{I}}$ denote the quotient IMC \mathcal{I} w.r.t certain bisimulation equivalence (i.e strong bisimulation, weak bisimulation, or stochastic branching bisimulation), then we have $\widehat{\mathcal{I}} \sim (\approx, \approx_{br}) \mathcal{I}$. Assume the IMC we want to analyze has the following form:

$$\mathcal{I} = \mathcal{I}_1 \parallel_{A_1} \mathcal{I}_2 \parallel_{A_2} \cdots \parallel_{A_{N-1}} \mathcal{I}_N.$$

Here we assume the size of \mathcal{I} is too large to generate and hence get the $\widehat{\mathcal{I}}$ direct from \mathcal{I} is not possible. However, the each component $(\mathcal{I}_i, i = 1 \cdots N)$ in \mathcal{I} is of a tractable size. Then we can first get every $\widehat{\mathcal{I}}_i$ from \mathcal{I}_i , and further we have $\mathcal{I}_i \sim (\approx, \approx_{br}) \widehat{\mathcal{I}}_i$ for $0 < i \leq N$. As a result, if we construct the $\widehat{\mathcal{I}}$ by parallel composition of these quotient IMCs, we have:

$$\begin{aligned} \mathcal{I}_1 \parallel_{A_1} \mathcal{I}_2 \parallel_{A_2} \cdots \parallel_{A_{N-1}} \mathcal{I}_N & \sim \\ & \approx \widehat{\mathcal{I}}_1 \parallel_{A_1} \widehat{\mathcal{I}}_2 \parallel_{A_2} \cdots \parallel_{A_{N-1}} \widehat{\mathcal{I}}_N \\ & \approx_{br} \end{aligned}$$

Compositional aggregation approach. If we extend this property on IMC a bit more, we will find the essential idea behind the well known *compositional aggregation approach*.

Now, we do not restrict to every IMC which is parallel composed with other IMC but on every IMC generated by parallel composition. For a more compact generation size, we now do bisimulation minimization on the IMCs each time we get the intermediate IMCs from parallel composition and then adopt this minimized IMC for later parallel composition with the remaining IMCs until we get the end minimize IMC. Using this approach, it directly follows that:

$$\begin{aligned}
 \mathcal{I}_1 \parallel_{A_1} \mathcal{I}_2 \parallel_{A_2} \mathcal{I}_3 \parallel_{A_3} \cdots \parallel_{A_{N-1}} \mathcal{I}_N & \sim \overbrace{\widehat{\mathcal{I}}_1 \parallel_{A_1} \widehat{\mathcal{I}}_2 \parallel_{A_2} \widehat{\mathcal{I}}_3 \parallel_{A_3} \cdots \parallel_{A_{N-1}} \widehat{\mathcal{I}}_N}^{\dots} \\
 & \approx \widehat{\mathcal{I}}_1 \parallel_{A_1} \widehat{\mathcal{I}}_2 \parallel_{A_2} \widehat{\mathcal{I}}_3 \parallel_{A_3} \cdots \parallel_{A_{N-1}} \widehat{\mathcal{I}}_N \\
 & \approx_{br}
 \end{aligned}$$

Note that during applying the compositional aggregation approach, the parallel composition ordering has now a significant impact on the maximal size of intermediate IMCs. This problems will be discussed in [15], and some restrictions of this approach will be discussed later in the following chapters.

2.3.4 Analysis of IMC

For analysis, we always consider the *closed* IMCs, which are the resulting IMCs after composition and not subject to any further synchronization [46]. In these closed IMCs, we can hide all remaining external actions safely. If we are lucky, there is no non-determinism in the resulting closed IMC (or non-determinism can be resolved by bisimulation equivalence), and after applying weak or branching bisimulation on the IMC, we should get a CTMC. The traditional analysis approaches like transient and steady analysis or the CSL model checking can be used on this CTMC.

If the non-determinism is present in the generated IMC after reduction, the analysis depends on a specific resolution of the non-determinism by using *policy* (also called *scheduler*) in the IMC. The barrier of analysis on IMCs with presence of non-determinism is overcome by the approach described in [46], and the analysis tool is already implemented by Software Modeling and Verification (MOVES) Group of RWTH Aachen University [4].

Some case studies by using IMC as semantic model are summarized in [20].

Chapter 3

Scenario-Aware Dataflow (SADF)

3.1 Introduction

In [31] A.Lee and D.Messerschmitt introduced a special data flow paradigm called Synchronous Dataflow (SDF), which are used to describing DSP applications. SDFs are represented as directed graphs which consist of nodes and arcs, where the directed graph describes the algorithm, nodes represent functions, and arcs represent data paths. In SDF, different sample rates in the same system can be easier handled by specify the number of data samples produced or consumed by each node on each invocation[31].

But one handicap exists in SDF: the expressiveness of traditional SDF is limited, since the SDF model can only describe the streaming applications that are static [42], and SDF lacks of means to express the dynamism in the applications. In [42], Bart.D. Theelen et al introduced a new dataflow model which allows dynamism to be modeled by using (sub)scenarios. The benefit of introducing dynamism into the modeling is that we can obtain more realistic (and accurate) performance analysis than traditional design-time analyzable dataflow models. We first introduce some terminologies in SADF specification and then elaborate the syntax and semantics of SADF specification.

3.1.1 Scenario-Aware Data Flow(SADF)

Terminology

Roughly speaking, an SADF graph consists of four kinds of components, i.e., kernels, detectors, and data channels and control channels. We first introduce some definitions used in the SADF specification.

- Processes** The basic functional units of SADF are processes. Two types processes are distinguished: *Kernels* and *Detectors*. Both of them are presented in the SADF graph as circles. In [42], the detectors are represented as dashed line circles and the kernels are represented as solid line circles. In our paper, we use the the green circles to represent the detectors and the blue circles to represent the kernels.
- Kernels** Kernel is similar as the node in SDF model and represents the data processing part in a streaming application. We can think kernel as a worker. This worker will finish certain tasks based on his working scenarios (contexts). To finish his work, he must first receive an order from each of his supervisor(s) (managers, later as detector), based on these orders, he can decide his working scenario and wait some other workers for resource from his supply lines (later as data channels). If all these conditions are fulfilled, he starts to work. After he has finished his work, the required resources are consumed, and the produced resources are available for other workers.
- Detectors** Detectors represents the control part of the streaming application. Detectors detect the scenarios for the kernels it connects. We can think detector as a supervisor (manager) who manages a certain number of workers (may even be managers for other workers). The manager can decide the kind of tasks for the workers by sending the orders to the workers he manages. But the manager may not make the decision arbitrarily. The kinds of order are decided by certain function (later, a embedded Markov chain in the detector), and the time to send order may depends also on the working progress of certain workers (kernels or detectors).
- Token** A Token represents a unit of information can be processed or produced by a process. The tokens are transferred and stored in channels. We can image tokens as resources for management or produc-

tion on a assemble line. There are two kinds of tokens: *data* tokens and *scenario* tokens.

- Channels** The processes are connected by channels, which are represented in SADF graph as edges (arcs). The channels represent the (unbounded) FIFO buffers to store the exchanging information for the processes at both sides. We can image the channels as assemble (supply) lines which connect the workers and managers. Two kinds of channels are distinguished, *data* channels and *control* channels. In the SADF graph, control channels are represented as dashed line edges, whereas the data channels are represented as solid line edges.
- Ports** Processes are connected by data/control channels through corresponding ports. We can image that the processes use their ports to connect the entrances or exits of the assemble lines. Three types of ports defined: (*data*) *input ports*, *control (input) ports*, *output ports*. The output ports of a detector are connected with data channels or control channels, while the output ports of a kernel are always connected with data channels. The input port of processes are always connected with data channels, and the control port of a process are always connected with control channels.
- Data Channels** A kernel's output port is always connected with a data channels. In other words, kernels always produce non-valued data tokens into the data channels, and these data tokens may be consumed by kernels or detectors. We can image the data channels as assemble lines, which only carries one kind of resources for production. Only the amount of resource is important.
- Control Channels** The control channels carries the order information for the workers from their managers, hence the tokens in the control channels are valued, which indicate different orders.
- (Sub)scenarios** Scenarios play a center role in SADF and express the dynamism in SADF specification. Scenarios are the abstraction of the operation modes of components in SADF. Scenarios can be seen as orders from managers. Kernels bear only one scenario set, while detectors always have scenario set and subscenario set. The detector always operates in subscenarios, which are decided based on the value of current scenarios from his managers.
- Rate** The rates indicate the token consumption/production quantities of a process from/to the channels it connects after its firing. The rates

are parameterized and depends on the current (sub)scenario, in which the process is operating. In the SADF, the rates of determined by predefined function.

Like SDF models, the SADF models can also be represented as direct graphs (cf. Figure 3.1). But only direct graph is insufficient for the sake of describing dynamism. Three additional techniques are needed. In SDF, the number of consumption or production samples (rates) of each node is specified *a priori*. The SADF follows that way but extends to (sub)scenarios. A function R is defined to determine the production/consumption rates of a processes associated with (sub)scenarios. Different firing times of a process in one (sub)scenario are determined by a generic discrete execution time distribution. A Markov chain embedded in the detector is used to specify probability of scenario occurrences. We give the abstract syntax of an SADF graph as follows.

3.2 Formal Definition of SADF

For a given SADF graph \mathfrak{G} , following sets and functions are defined [41]:

$\mathcal{K}, \mathcal{D}, \mathcal{P}$ denote the finite set of kernels and detectors and processes in \mathfrak{G} respectively. From the definition we have: $\mathcal{P} = \mathcal{K} \cup \mathcal{D}$ and $\mathcal{K} \cap \mathcal{D} = \emptyset$,

$\mathcal{I}_p, \mathcal{O}_p, \mathcal{C}_p$ denote the input, output and control ports of a process p respectively,

$\mathcal{I}, \mathcal{O}, \mathcal{C}$ denote the union of all input, output and control port set in \mathfrak{G} respectively:
 $\mathcal{I} = \bigcup_{p \in \mathcal{P}} \mathcal{I}_p$, $\mathcal{O} = \bigcup_{p \in \mathcal{P}} \mathcal{O}_p$ and $\mathcal{C} = \bigcup_{p \in \mathcal{P}} \mathcal{C}_p$.

$\mathcal{B}, \mathcal{B}_c, \mathcal{B}_d$ denotes the set of all channels, the set of all control channels, the set of all data channels in \mathfrak{G} respectively. Also, we have $\mathcal{B} = \mathcal{B}_c \cup \mathcal{B}_d$ and $\mathcal{B}_c \cap \mathcal{B}_d = \emptyset$. In the IMC semantics of an SADF graph, we adopt the ordered pair of input and output ports to identify the channels: $(x, y) \in \mathcal{O} \times (\mathcal{I} \cup \mathcal{C})$.

Σ_b As aforementioned, the control channel is in fact modeled as an unbounded FIFO buffer. Then we use Σ_b to denotes the finite set of possible values of scenario token can be transferred in a control channel b and Σ_b^* is the set of all finite sequences of symbols in Σ_b that can be produced on the control channel b . Then a sequence $\sigma \in \Sigma_b^*$ is denoted in the form: $\sigma = \sigma_1 \dots \sigma_n$.

S_p	denotes the non-empty finite set of scenarios in which a process p can operate.
Ω_d	denotes the non-empty finite set of subscenarios of detector d .
R_k^s	denotes the rate function of a kernel $k \in \mathcal{K}$. The function $R_k^s : \mathcal{I}_k \cup \mathcal{O}_k \cup \mathcal{C}_k \rightarrow \mathbb{N}$ assigns a natural number as rate to the ports of kernel k according to the scenario s in k .
R_d^ω	is the rate function of a detector $d \in \mathcal{D}$. The function $R_d^\omega : \mathcal{I}_d \cup \mathcal{O}_d \cup \mathcal{C}_d \rightarrow \mathbb{N}$ specifies the rate for a given $\omega \in \Omega$.
t_d^ω	is the sequence of tokens d produce after its execution in scenario ω onto an output port $o \in \mathcal{O}_d$. we easily infer that $t_d^\omega \in \Sigma_b^*$, if the output port is connected to a control channel b .
E_s^k	is the discrete random variable that captures the generic execution time distribution of a kernel k for scenario $s \in S_k$.
E_ω^d	is the discrete random variable that captures the generic execution time distribution of a detector d for subscenario $\omega \in \Omega_d$. Recall that, in IMC semantics, we adapt the generic execution time distribution for a (sub)scenario in SADF to a single Markovian transition which interpretes a mean delay (value) of the weight average of values in the sample space of the generic execution time distribution.
$(\mathbb{S}, \iota, \mathbb{P})$	is a finite-state discrete time Markov chain (DTMC) associated with a scenario embedded in a detector. This Markov chains can be used to interpret the function to determinate the occurrence of subscenarios. If a detector d has no control ports, it always operates in a default scenario ϵ_d .
Φ_d^s	is the function to determine the subscenario for the detector d according to the current state in the Markov chain associated with scenario s . In other words, the function Φ_d^s takes a state in the Markov chain belonging to scenario s as argument and returns a subscenario, in which the detector d is going to operate.
ϕ	is the <i>data channel status</i> function of the SADF graph. This function takes each (data) channel as argument and returns the current number of tokens stored in each data channel. Thus ϕ has this form: $\mathcal{B}_d \rightarrow \mathbb{N}$.
ψ	is the <i>control channel status</i> . Similar as ϕ , ψ is a function which takes each control channel as argument and returns the sequence of scenario tokens in each control channel. Thus ψ has this form: $\mathcal{B}_c \rightarrow \bigcup_{c \in \mathcal{B}_c} \Sigma_c^*$.

Now we define the syntax of a SADF graph as follows.

Definition 3.1 (SADF Graph) An SADF graph \mathfrak{G} is a tuple $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$ where

- each kernel $k \in \mathcal{K}$ is a tuple $(\mathcal{I}_k, \mathcal{O}_k, \mathcal{C}_k, \mathcal{S}_k, \{(R_k^s, E_k^s) \mid s \in \mathcal{S}\})$ where
 - $\mathcal{S}_k = \prod_{c \in \mathcal{C}_k} \sum_b$, where b is the control channel connected to control port c ,
 - $\forall s \in \mathcal{S}_k$ and $\forall c \in \mathcal{C}_k$, we have $R_k^s(c) = 1$;
- each detector $d \in \mathcal{D}$ is a tuple $(\mathcal{I}_d, \mathcal{O}_d, \mathcal{C}_d, \mathcal{S}_d, \{(\mathbb{S}_d^s, t_d^s, \mathbb{P}_d^s, \Phi_d^s) \mid s \in \mathcal{S}_d\}, \Omega_d, \{(R_d^\omega, t_d^\omega, E_d^\omega) \mid \omega \in \Omega_d\})$ where
 - $\mathcal{S}_d = \prod_{c \in \mathcal{C}_d} \sum_b$, where b is the control channel connected to control port c ,
 - for every $s \in \mathcal{S}_d$, $(\mathbb{S}_d^s, t_d^s, \mathbb{P}_d^s, \Phi_d^s)$ is a non-empty finite state space Markov chain $(\mathbb{S}_d^s, t_d^s, \mathbb{P}_d^s)$ associated with a function: $\Phi_d^s \rightarrow \Omega_d$, which determines the subscenario for each state in the Markov chain,
 - $\forall \omega \in \Omega_d$ and $\forall c \in \mathcal{C}_d$: $R_d^\omega(c) = 1$,
 - $\forall \omega \in \Omega_d$ and $\forall o \in \mathcal{O}_d$, we have $R_d^\omega(o) > 0$ and $t_d^\omega(o) \in \{v \in \Sigma_b^* \mid |v| = R_d^\omega(o)\}$;
- $\phi^* : \mathcal{B}_d \rightarrow \mathbb{N}$ is the initial data channel status;
- $\psi^* : \mathcal{B}_c \rightarrow \cup_{c \in \mathcal{B}_c} \Sigma_c^*$ is the initial control channel status.

Some explanations about Definition 3.1. Here, we have a MPEG-4 Decoder shown in Figure 3.1 as an example. From the definition we have:

Port	(Sub)Scenario								
	I	P_0	P_{30}	P_{40}	P_{50}	P_{60}	P_{70}	P_{80}	P_{99}
a	0	0	1	1	1	1	1	1	1
b	0	0	30	40	50	60	70	80	99
c	99	1	30	40	50	60	70	80	99
d	1	0	1	1	1	1	1	1	1
e	99	0	30	40	50	60	70	80	99

Table 3.1: The rate functions R_k^s and R_d^ω of MPEG-4 decoder

Process	(Sub)Scenario	$E_{(p,s)}$ (kCycles)
VLD	P_0	0
	All except P_0	40
IDCT	P_0	0
	All except P_0	17
MC	I, P_0	0
	P_{30}	90
	P_{40}	145
	P_{50}	190
	P_{60}	235
	P_{70}	265
	P_{80}	310
	P_{99}	390
RC	I	350
	P_0	0
	P_{30}, P_{40}, P_{50}	250
	P_{60}	300
FD	P_{70}, P_{80}, P_{99}	320
	All	0

Table 3.2: The execution time distribution function of processes in MPEG-4 decoder

- $\mathcal{D} = \{FD\}$,
- $\mathcal{K} = \{VLD, IDCT, MC, RC\}$,
- $\mathcal{S}_{VLD} = \mathcal{S}_{IDCT} = \mathcal{S}_{MC} = \mathcal{S}_{RC} = \{I, P_0, P_{30}, P_{40}, P_{50}, P_{60}, P_{70}, P_{80}, P_{99}\}$,
- $\mathcal{S}_{FD} = \{\epsilon_{FD}\}$, $\Omega_{FD} = \{I, P_0, P_{30}, P_{40}, P_{50}, P_{60}, P_{70}, P_{80}, P_{99}\}$,
- The set of function $\{R_k^s \mid k \in \mathcal{K}, s \in \mathcal{S}_k\}$ and $\{R_d^\omega \mid d \in \mathcal{D}, \omega \in \Omega_d\}$ is shown in Table 3.1.
- The execution time distribution function is shown in Table 3.2.
- $(\mathbb{S}_d^{\epsilon_{FD}}, \iota_{FD}^{\epsilon_{FD}}, \mathbb{P}_{FD}^{\epsilon_{FD}})$ is shown in Figure 3.2.
- The associated the function $\Phi_{FD}^{\epsilon_{FD}}$ with the states in the Markov chain is defined by: $\Phi_{FD}^{\epsilon_{FD}}(S_1) = I$, $\Phi_{FD}^{\epsilon_{FD}}(S_2) = P_0$, $\Phi_{FD}^{\epsilon_{FD}}(S_3) = P_{30}$, $\Phi_{FD}^{\epsilon_{FD}}(S_4) = P_{40}$, $\Phi_{FD}^{\epsilon_{FD}}(S_5) = P_{50}$, $\Phi_{FD}^{\epsilon_{FD}}(S_6) = P_{60}$, $\Phi_{FD}^{\epsilon_{FD}}(S_7) = P_{70}$, $\Phi_{FD}^{\epsilon_{FD}}(S_8) = P_{80}$, $\Phi_{FD}^{\epsilon_{FD}}(S_9) = P_{99}$.

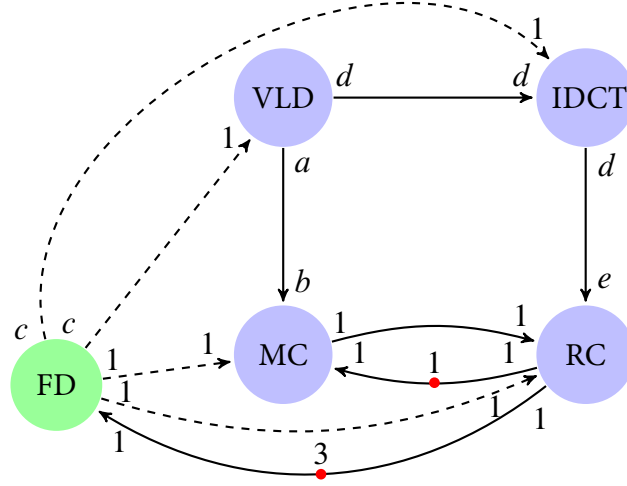


Figure 3.1: An SADF graph to model the MPEG-4 decoder

3.2.1 Operational Semantics of SADF

The operational semantics of an SADF graph is defined by a Timed Probabilistic System (TPS). Timed Probabilistic System is a variant of LTS, which also consists of configurations (states) and transitions. The transitions are distinguished by timeless transitions and timed transitions. First we introduce two additional functions (the data channel status ϕ and the control channel ψ are already introduced in last section).

κ is the *kernel status* function for an SADF $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$. This function assigns to each kernel $k \in \mathcal{K}$ an ordered pair in $(\mathcal{S}_k \cup \{-\}) \times (\mathbb{R}_0^+ \cup \{-\})$, the former element denotes the current scenario in which k is operating and the latter element denotes the remaining time for k to finish its firing. If the kernel is inactive then its status will be denoted as $(-, -)$.

δ is the *detector status* function for an SADF $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$. This function assigns to each detector $d \in \mathcal{D}$ a tuple in $\prod_{s \in \mathcal{S}_d} \mathbb{S}_d^s \times (\Omega_d \cup \{-\}) \times (\mathbb{R}_0^+ \cup \{-\})$. The first part denotes the vector of current states (the vector of the last leaving states in each Markov chains in d) of Markov chains associated with each scenario $s \in \mathcal{S}_d$ as a tuple S_{s_1}, \dots, S_{s_N} , the middle element denotes the current subscenario ω in which d is operating, and the last element denotes the remaining time for d to finish its firing. If the detector d is inactive currently, its status will be denoted as $(S_{s_1}, \dots, S_{s_N}, -, -)$.

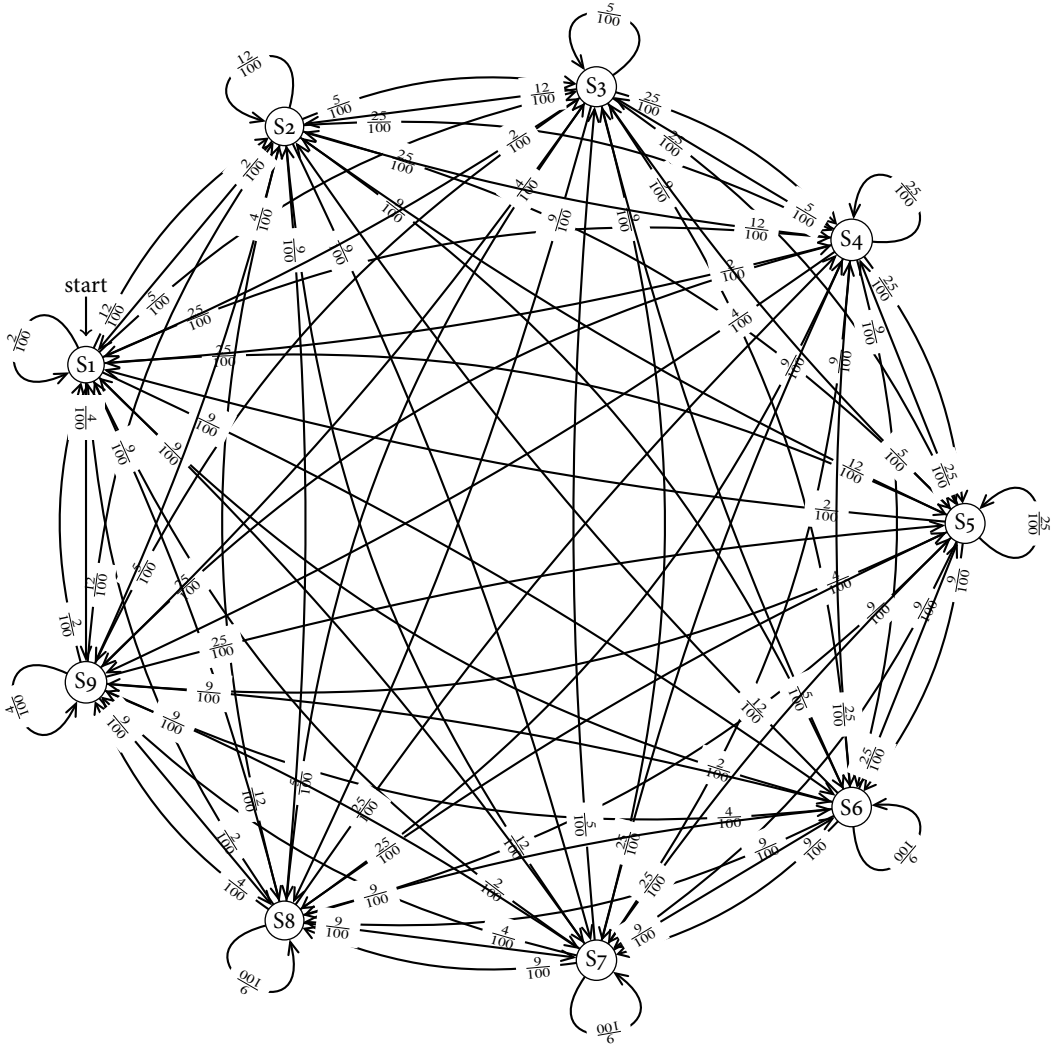


Figure 3.2: The discrete-time Markov chain embedded in detector FD associated with its default scenario ϵ_d

Definition 3.2 (Configuration). A configuration of an SADF graph is a tuple $(\phi, \psi, \kappa, \delta)$, where $\phi, \psi, \kappa, \delta$ denote a channel status, control status, kernel status, and detector status respectively. SADF graph $(\mathcal{K}, \mathcal{D}, \mathcal{B}, \phi^*, \psi^*)$ has initial configuration $C^* = (\phi^*, \psi^*, \kappa^*, \delta^*)$, where the initial kernel status κ^* is defined by $\kappa^*(k) = (-, -)$ for all $k \in \mathcal{K}$ and initial detector status δ^* is defined by $\delta^*(d) = (t_d^{S_1}, \dots, t_d^{S_N}, -, -, -)$ for all $N = |\mathcal{S}_d|$ scenarios of each $d \in \mathcal{D}$.

Definition 3.3 (The operational semantics of an SADF graph). The operational se-

mantics of an SADF graph is an TPS defined as a tuple $\mathfrak{T} = (\Theta, C^*, \mathcal{A}, \mathbb{A}, \mathcal{T}, \mathbb{T})$, where

- Θ is defined as the set of all reachable configurations of the SADF graph, and let $\mathcal{D}(\Theta)$ to be the set of probabilistic distributions over Θ , then we have $\mathcal{D}(\Theta) = \{\boldsymbol{\pi} : \Theta \rightarrow [0, 1] \mid \sum_{c \in \Theta} \boldsymbol{\pi}(c) = 1\}$,
- $C^* \in \Theta$ is an initial configuration,
- \mathcal{A} is a set of actions,
- $\mathcal{T} \subseteq \mathbb{R}_0^+$ is the domain of time and let $\mathcal{T}^+ = \mathcal{T} \setminus \{0\}$,
- $\mathbb{A} \subseteq \Theta \times \mathcal{A} \times \mathcal{D}(\Theta)$ is a set a of action transitions, and
- $\mathbb{T} \subseteq \Theta \times \mathcal{T}^+ \times \mathcal{D}(\Theta)$ is a set of time transitions.

Semantics of Transitions. The core of the operational semantics of an SADF are the transition relations. Two kinds of transitions are distinguished: *action transition* and *time transition*. Again in action transitions, there are 6 types: *control action transition*, *detect action transition*, *kernel start action transition*, *kernel end action transition*, *detector start action transition*, and *detector end action transition*. The concrete semantics of these transitions are described as follows.

Definition 3.4 (Control Action Transition) A control action transition indicates that a kernel k “confirms” a scenario, in which it is going to operate. The prerequisite that kernel k can confirm the scenario is every control channel with which its control ports connect has at least one scenario token. Then we define the control action transition by:

- $\xrightarrow{\text{control}(k)} \subseteq \Theta \times \mathcal{D}(\Theta)$, and if
- $\kappa(k) = (-, -)$ and $|\psi(b_c)| \geq 1$ for each channel b_c connected to a control port $c \in C_k$, then $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{control}(k)} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi, \psi, \kappa', \delta) = 1$, where the resulting kernel status κ' is defined by:
 - $\kappa' = \begin{cases} \kappa[k \rightarrow (\epsilon_k, -)] & \text{if } C_k = \emptyset \\ \kappa[k \rightarrow (\prod_{c \in C_k} \psi(b_c)_1, -)] & \text{otherwise} \end{cases}$

Some explanations on the definition of control action transition: When each control ports of kernel k has at least scenario token available, which means the scenario, in which k is going to operate is uniquely determined. The value of scenario depends on the sequence of the first scenario token values of each control ports of k , i.e. $\prod_{c \in \mathcal{C}_k} \psi(b_c)_1$. Accordingly, after the transition the first element of kernel status of k is modified to this value. Since when the firing of k starts depends on the sufficiency data tokens at its data ports now, the second element of kernel status (the remaining execution time of k) remains undefined (-). A special case is when the k has no control ports, i.e., k operates always in its default scenario (ϵ_k). Moreover, the transition to the next configuration $(\phi, \psi, \kappa', \delta)$ is unique determined, so the probability to be in that configuration is $\pi(\phi, \psi, \kappa', \delta) = 1$.

Definition 3.5 (Detect Action Transition) A detect action transition is a transition to indicate the determination of the scenario and subscenario in a detector. The subscenario is written into the configuration of detector, and used for the later firing. The detect action transition of a detector d is defined as follows:

- $\xrightarrow{\text{detect}(d)} \subseteq \Theta \times D(\Theta)$, and if
- $\delta(d) = (S_{s_1}, \dots, S_{s_{i-1}}, S_{s_i}, S_{s_{i+1}}, \dots, S_{s_N}, -, -)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with d , and $|\psi(b_c)| \geq 1$ for each channel b_c connected to a control port $c \in \mathcal{C}_d$, then $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{detect}(d)} \pi$, where the distribution function π is given by:

$$\pi(\phi, \psi, \kappa, \delta') = \begin{cases} \mathbb{P}_d^{\epsilon_d}(S_{\epsilon_d}, T) \text{ for each } T \in \mathbb{S}_d^{\epsilon_d} & \text{if } \mathcal{C}_d = \emptyset \\ \mathbb{P}_d^{s_i}(S_{s_i}, T) \text{ for each } T \in \mathbb{S}_d^{s_i} & \text{otherwise} \\ \text{with } s_i = \prod_{c \in \mathcal{C}_d} \psi(b_c)_1 & \end{cases}$$

and the detector status is modified from δ to δ' as follows:

$$\delta' = \begin{cases} \delta[d \longrightarrow (T, \Phi_d^{\epsilon_d}(T), -)] \text{ for each } T \in \mathbb{S}_d^{\epsilon_d} & \text{if } \mathcal{C}_d = \emptyset \\ \delta[d \longrightarrow (S_{s_1}, \dots, S_{s_{i-1}}, T, S_{s_{i+1}}, \dots, S_{s_N}, \Phi_d^{s_i}(T), -)] & \text{otherwise} \\ \text{for each } T \in \mathbb{S}_d^{s_i} & \end{cases}$$

If a detector d is not executing in some scenario and also the subscenario is not determined (according to the content of last two elements “–” in $\delta(d)$ ’s status), then two situations are distinguished. If d has no control port, then it can directly enter the Markov Chain associated with its default scenario ϵ_d for requesting the possible subscenarios for the later execution, or if it has control ports, then it must first determine the scenario according to the values of control tokens at its control ports, this is determined by $s_i = \prod_{c \in \mathcal{C}_d} \psi(b_c)_1$, then d enters the Markov chain associated with scenario s_i for requesting the subscenario. Since the vector (tuple) $(S_{s_1}, \dots, S_{s_{i-1}}, S_{s_i}, S_{s_{i+1}}, \dots, S_{s_N})$ remembers the last (leaving) state of each Markov chain with the corresponding scenario, the enter state now is the last leaving state (here S_{s_i}), the probabilities to get to the successor state(s) of S_{s_i} are determined by the function \mathbb{P} of DTMC associated with scenario s_i . Then the probabilistic transitions are made and the last leaving state in the DTMC associated with scenario s_i is modified to the successor state(s) (T) instead of S_{s_i} , and the subscenario determined by $\Phi_d^{s_i}(T)$ is written to the penultimate element of the configuration of d . The last element of the configuration (status) of d (remaining execution time of d) is still undefined (–), this will be modified by the *detector start action transition* after the data token availability is checked. For the detector d without any control channels, this detector has only one Markov chain for the default scenario ϵ_d and the condition of emanating the detect action transition is satisfied immediately.

Definition 3.6 (Kernel Start Action Transition) A kernel start action transition *indicates the start of processing data (firing) by a kernel. The kernel start action transition of a kernel k is defined as:*

- $\xrightarrow{\text{start}(k)} \subseteq \Theta \times D(\Theta)$, and if
- $\kappa(k) = (s, -)$ for some scenario $s \in \mathcal{S}_k$ and $\phi(b_i) \geq R_k^s(i)$ for each channel b_i connected to input port $i \in \mathcal{I}_k$ then $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{start}(k)} \boldsymbol{\pi}$, where
- the function $\boldsymbol{\pi}$ is defined by $\boldsymbol{\pi}(\phi, \psi, \kappa'_e, \delta) = \mathbb{P}(E_k^s = e)$ with $\kappa'_e = \kappa[k \rightarrow (s, e)]$ for all execution times e in the sample space of E_k^s .

If the scenarios s for the kernel k ’s firing is determined by the control action transition (cf. Definition 3.5 with the modified configuration $\kappa(k) = (s, -)$), and now the number of data tokens in each data channel b_i connected with k ’s input port $i \in \mathcal{I}_k$ is check by the function $R_k^s(i)$. If the data tokens are enough for all these channels, then k can start its execution by making a kernel start action transition labelled with k . The probabilities of successor states ($\boldsymbol{\pi}$) are decided by the generic discrete distribution function $\mathbb{P}(E_k^s = e)$, where e is the execution time, and $\mathbb{P}(E_k^s = e)$ returns the probability of

k finishes its firing in e time units in scenario s . After the kernel start action transition, the configuration of k is modified to (s, e) accordingly.

Definition 3.7 (Detector Start Action Transition) A detector start action transition indicates the start of execution of a detector. We define the detector start action transition of a detector d as:

- $\xrightarrow{start(d)} \subseteq \Theta \times D(\Theta)$, and if
- $\delta(d) = (S_{s_1}, \dots, S_{s_N}, \omega, -)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with d , a subscenario $\omega \in \Omega_d$, and if $\phi(b_i) \geq R_d^\omega(i)$ for each channel b_i connected to an input port $i \in \mathcal{I}_d$, then $(\phi, \psi, \kappa, \delta) \xrightarrow{start(d)} \boldsymbol{\pi}$, where
- the function $\boldsymbol{\pi}$ is defined by $\boldsymbol{\pi}(\phi, \psi, \kappa, \delta'_e) = \mathbb{P}(E_d^\omega = e)$ with $\delta'_e = \delta[d \rightarrow (S_{s_1}, \dots, S_{s_N}, \omega, e)]$ for all execution times e in the sample space of E_d^ω .

Similarly as kernel, detector start action transitions describes the execution start of a detector d . The subscenario for later execution is determined some time before by the detector action transition (cf. Definition 3.5). The next step is to wait for all the data tokens available of each channel connected with d 's input ports according to the determined subscenario ω . If all data tokens are available, the kernel starts to execute. The detector start action transition is also probabilistic distributed, the distribution is decided by the function $\mathbb{P}(E_d^\omega = e)$. Then the execution time e is written into the last element (remaining execution time) of the configuration of d .

Definition 3.8 (Kernel End Action Transition) A kernel end action transition indicates the end of firing of a kernel. The kernel end action transition of a kernel k is defined as:

- $\xrightarrow{end(k)} \subseteq \Theta \times D(\Theta)$, and if
- $\kappa(k) = (s, 0)$ for some scenario $s \in \mathcal{S}_k$, then $(\phi, \psi, \kappa, \delta) \xrightarrow{end(k)} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi', \psi', \kappa', \delta') = 1$, where
- the resulting configuration $(\phi', \psi', \kappa', \delta')$ is defined by

$$\phi' = \begin{cases} \phi[b_i \rightarrow \phi(b_i) - R_k^s(i)] & \text{for each non self-loop channel } b_i \\ & \text{connected to an input port } i \in \mathcal{I}_k \\ \phi[b_o \rightarrow \phi(b_o) + R_k^s(o)] & \text{for each non self-loop channel } b_o \\ & \text{connected to an output port } o \in \mathcal{O}_k \\ \phi[b_l \rightarrow \phi(b_{oi}) + R_k^s(o) - R_k^s(i)] & \text{for each self-loop channel } b_{oi} \\ & \text{connecting to an output port } o \in \mathcal{O}_k \\ & \text{to an input port } i \in \mathcal{I}_k \end{cases}$$

and $\psi' = \psi[b_c \rightarrow \psi(b_c) - \psi(b_c)_1]$ for each channel b_c connected to a $c \in \mathcal{C}_k$ and $\kappa' = \kappa[k \rightarrow (-, -)]$.

If kernel k finishes its firing, i.e., the remaining execution time (the last element of k 's configuration) is equal to zero, the kernel end action transition is made. There is no probabilistic distribution of this transitions. After this transition, the data tokens in the control channels and data tokens in the data channels connected with k are consumed or produced at the same time according to the R_k^s function, and the configuration of k becomes undefined to $(-, -)$.

Definition 3.9 (Detector End Action Transition) A detector end action transition indicates the end of execution of a detector. The detector end action transition of detector d is defined as:

- $\xrightarrow{\text{end}(d)} \subseteq \Theta \times D(\Theta)$, and if
- $\delta(d) = (S_{s_1}, \dots, S_{s_N}, \omega, 0)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with d , a subscenario $\omega \in \Omega_d$, then $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{end}(d)} \pi$ with $\pi(\phi', \psi', \kappa', \delta') = 1$, where
- the resulting configuration $(\phi', \psi', \kappa', \delta')$ is defined by

$$\phi' = \begin{cases} \phi[b_i \rightarrow \phi(b_i) - R_d^\omega(i)] & \text{for each non self-loop channel } b_i \\ & \text{connected to an input port } i \in \mathcal{I}_d \\ \phi[b_o \rightarrow \phi(b_o) + R_d^\omega(o)] & \text{for each non self-loop channel } b_o \\ & \text{connected to an output port } o \in \mathcal{O}_d \\ \phi[b_l \rightarrow \phi(b_{oi}) + R_d^\omega(o) - R_d^\omega(i)] & \text{for each self-loop channel } b_{oi} \\ & \text{connected to an } o \in \mathcal{O}_d \text{ to an } i \in \mathcal{I}_d \end{cases}$$

and

$$\psi' = \begin{cases} \psi[b_c \rightarrow \psi(b_c) - \psi(b_c)_1] & \text{for each non self-loop channel } b_c \\ & \text{connected to an } c \in \mathcal{C}_d \\ \psi[b_o \rightarrow \psi(b_o) + t_d^\omega(o)] & \text{for each non self-loop channel } b_o \\ \text{with } t_d^\omega \in \sum_{b_c}^* \text{ and } |t_d^\omega(o)| = R_d^\omega(o) & \text{connected to an } o \in \mathcal{O}_d \\ \psi[b_l \rightarrow \psi(b_{oc}) + t_d^\omega(o) - \psi(b_{oc})_1] & \text{for each self-loop channel } b_{oi} \\ \text{with } t_d^\omega \in \sum_{b_{oc}}^* \text{ and } |t_d^\omega(o)| = R_d^\omega(o) & \text{connected to an output port } o \in \mathcal{O}_d \\ & \text{to an control port } c \in \mathcal{C}_d \end{cases}$$

$$\text{and } \delta' = \delta[d \rightarrow (S_{s_1}, \dots, S_{s_N}, -, -)].$$

If a detector d finishes its execution, i.e., the remaining firing time (the last element of the configuration of d) is equal to zero, the detector end action transition is made. The tokens are produced and consumed according to the function R_d^ω at the same time, and the last two elements (the current subscenario and remaining time) become undefined.

Definition 3.10 (Time Transition) A time transition denotes the time advancing of the execution of the SADF graph. The time transition of t time units is defined as:

- $\xrightarrow{\text{time}(t)} \subseteq \Theta \times D(\Theta)$, and if
- there exists a process with a positive remaining execution time and no end transitions are enabled, and t is the smallest remaining execution time of all processes with a positive remaining execution time, then $(\phi, \psi, \kappa, \delta) \xrightarrow{\text{time}(t)} \boldsymbol{\pi}$ with $\boldsymbol{\pi}(\phi, \psi, \kappa', \delta') = 1$, where the resulting configuration $(\phi, \psi, \kappa', \delta')$ is defined by
 - $\kappa'(k) = (s, n - t)$ if $\kappa(k) = (s, n)$ for some $s \in \mathcal{S}_k$ and $n > 0$ for each kernel k , and
 - $\delta'(d) = (S_{s_1}, \dots, S_{s_N}, \omega, n - t)$ if $\delta(d) = (S_{s_1}, \dots, S_{s_N}, \omega, n)$ for some combination of states for the $N = |\mathcal{S}_d|$ Markov chains associated with detector d , some $\omega \in \Omega_d$ and $n > 0$ for each detector d .

One thing to remark is that before any time transition can be made, there should be no end transitions can be enabled. This means finalizing the firing of processes has a higher

priority over advancing time [42]. On the other hand, this restriction does not state about how to resolve the non-determinism between other actions (e.g., control action transitions, detect action transitions, etc.) and advancing time (i.e. the time transition), which allows to capture all possible scheduling policies in the system. Further, among these possible scheduling policies, a class of scheduling policies which prescribes that performing actions cost no time and therefore should take precedence over advancing time. This assumption between time advancing and performing actions is called action urgency [36]. However, if we compare action urgency with the maximal progress assumption in IMC (cf. Definition 2.10), we will find the maximal progress is only allowed on the special internal interactive transitions (τ -transition). To allow action urgency on IMCs, we must manually set the priority on actions in IMC, and this will be discussed further on in this paper. This assumption (action urgency) matches in fact the so called *self-timed execution* for SDF graphs[42, 18], which is a scheduler that the firing of every process starts immediately when all required tokens are available. The self-timed execution is of special interest, since it ensures the best (tightest) bound that can be given on the timing behavior of the application base on the SDF model. But the action urgency assumption may be disadvantages for other metrics like latency [42].

3.3 Important Properties

In [41], some useful properties of SADF specification are stated. In this section, we introduce three important properties: boundedness, determinacy and long-run equivalence in SADF specification, which are very closely related to our IMC semantics. Note that, if we want to analyze the SADF model, many performance metrics (like throughput and time-average buffer occupancy) have no meaning, when the SADF is untimed [41].

Definition 3.11 [41] (**Untimed SADF**) *An SADF is called untimed, if the firing time of all processes in all their (sub)scenarios is zero.*

3.3.1 Boundedness

The most important property for our IMC semantic of SADF specification is the boundedness of an SADF graph. The definition of boundedness of SADF graph is defined as:

Definition 3.12 [41] (**Boundedness**) *An SADF is called bounded iff a $\mathcal{B} \in \mathbb{N}$ such that*

for all reachable configurations $(\phi, \psi, \kappa, \delta)$, $\phi(dc) \leq \mathfrak{B}$ for all $dc \in \mathcal{B}_d$ and $|\phi(cc)| \leq \mathfrak{B}$ for all $cc \in \mathcal{B}_c$. Otherwise, the SADF is unbounded.

Theorem 1 in [41] gives the criterion to determine whether a strongly consistent SADF graph is bounded or not. When we express the semantics of SADF model in terms of IMC, the only difference is the execution times per (sub)scenario expressed by a generic discrete distribution change to a single execution time expressed by a random variable which is governed by an exponential distribution. Since the definitions like strong consistency, strong dependency, active process in SADF specification which are not subject to the execution time of processes, the properties stated above are unchanged and can also simple be adapted into our case. Therefore, in our IMC model, it directly follows that if the SADF graph is bounded, then the state space of its corresponding IMC is finite.

3.3.2 Determinacy

Theorem 3 in [41] states that the functionality represented by an SADF does not affect by the non-deterministic choices but only by the probabilistic choice that determine the sequence of scenarios successively detected by each detector.

This theorem is validated by the diamond property in SADF.

Definition 3.13 [41] Let $\pi_{C,a} \in D(\Theta)$ denote the probability distribution over Θ after performing an action or time transition α from a configuration C if it exists. Relation $\pi_1 \xrightarrow{\alpha} \pi_2$ holds if for all configuration C with positive $\pi_1(C)$ transition α can be performed and $\pi_2(C') = \sum_{C \in \Theta} \pi_1(C) \cdot \pi_{C,\alpha}(C')$ for all possible resulting $C' \in \Theta$.

The the following lemma states the policy for resolving the non-deterministic choice between two actions that are performed immediately after each other is irreverent [41].

Lemma 3.1 [41] (**Diamond Property**) Let α and β be two enabled actions. If $\pi \xrightarrow{\alpha} \pi'$ and $\pi \xrightarrow{\beta} \pi''$ with $\pi, \pi', \pi'' \in D(\Theta)$, then there exists some $\pi^* \in D(\Theta)$ such that $\pi' \xrightarrow{\beta} \pi^*$ and $\pi'' \xrightarrow{\alpha} \pi^*$.

3.3.3 Long-run equivalence

For the performance analysis of untimed SADF, two approaches are used. One approach is to derive a (discrete-time) Markov chain from the involved TPS and then analyze the obtained Markov chain using a similar approach in [45]. With this approach, any long-run average or worst-case performance metric can be analyzed. The first step of this approach is to resolve the non-determinism in the involved TPS. The scheduler used to resolve the non-determinism may affect certain metrics like the maximum occupancy of a buffer, but will not affect many long-run average performance metrics. This property is stated by the Theorem 4 in [41].

Theorem 3.1 [41] (**Long-Run Equivalence**) *Any long-run average for a timed SADF model that merely depends on the status of the system in configuration of the TPS just before and after time transitions converges to the same result regardless how non-determinism is resolved.*

After this step, a new TPS is obtained and if the SADF is deadlock-free and further, by shifting the information on the occurrence of actions and process of time into the configuration, a discrete-time Markov chain is obtained for performance analysis. Here, the state-space of the obtained discrete-time Markov chain is finite if the SADF is bounded.

The second approach to evaluate certain specific worst-case metrics is the possibility to convert an SADF model into a set of SDF models. One SDF model in the set is for each possible scenario combinations using the largest possible execution time for each process in that scenario combination [41].

For details of the properties of SADF graph we refer to [41]. Furthermore, we also have the similar properties on the IMC semantics of SADF, and the differences and relations between the TPS semantics and IMC semantics of SADF models is discussed in Section 4.3.1.

Chapter 4

The IMC Semantics of SADF

Many successful case studies have proven that IMCs are a natural semantic model for stochastic process algebras [20]. In this chapter, we attempt to present a framework to build and analyze the SADF models by using IMCs. The details of the formal definition of an SADF graph and its operational semantics represented by Timed Probabilistic System (TPS) are introduced in Chapter 3. In our SADF specification, two modifications are needed before we adopt IMCs as the semantic model of SADF as aforementioned: First, in the TPS semantics of SADF, a generic discrete execution time distribution per (sub)scenario $\mathbb{P}(E_p^s = e)$ or $\mathbb{P}(E_d^\omega = e)$ is used. Now, we assume if the random variables in the sample space of a process p 's (a detector d 's) generic discrete execution time distribution of scenario s (subscenario ω) are e_1, e_2, \dots, e_n , then we emerge the generic discrete execution time distribution of s (ω) into a single constant execution time by $e_p^s = (\mathbb{P}(E_p^s = e_1) \cdot e_1 + \dots + \mathbb{P}(E_p^s = e_n) \cdot e_n)$ ($e_d^\omega = (\mathbb{P}(E_d^\omega = e_1) \cdot e_1 + \dots + \mathbb{P}(E_d^\omega = e_n) \cdot e_n)$). We do the emerge of the generic discrete execution time distribution for each process of all (sub)scenarios it has. Since the time transition represents a fixed-time delay before moving to a successor state from the current state, and this assumption does not coincide with the assumption that a Markovian transition in IMCs describe an exponentially distributed delay. We translate the average execution time e_p^s (e_d^ω) obtained by the first step into an exponentially distributed delay with mean time e_p^s (e_d^ω), which is represented by a Markovian transition with rate $\lambda_p^s = 1/e_p^s$ ($\lambda_d^\omega = 1/e_d^\omega$).

Furthermore, recall the unbounded channels in SADF, the status of a data channel can be represented as a natural number, and the status of a control channel is a sequence of values of tokens. It is a natural choice to represent the status of channels as state variables. Following this idea, we integrate the *state variables* into LTSs (it is the same

way to extend with IMCs). These variables have a certain predefined data type, and also some predefined operations are used. For instance, we may have an LTS representing a counter and integrate a natural number as variable in this LTS, the conventional operations on natural numbers like “ $+$, $-$, \leq , \geq , \dots ” can be used as their usual meanings. In this way, we obtain an LTS whose states and transitions represent the control part of the LTS and the variables represent the data part of the LTS. Now, we extend the state variables in our case and allow variables to be integrated in our IML expressions. Then, our extended IML expressions can interact with variables in two ways [7]:

Assignments: the values of variables can be changed by the transition.

Guards: a transition is called a *guarded* transition when the transition is only possible when the condition on the variables holds.

Example 4.1 Figure 4.1 shows a small example of extended LTS with a variable $x \in \mathbb{N}$. This LTS describes a bounded buffer (capacity=3) with certain functions. The variable x represents the current number of tokens in the buffer.

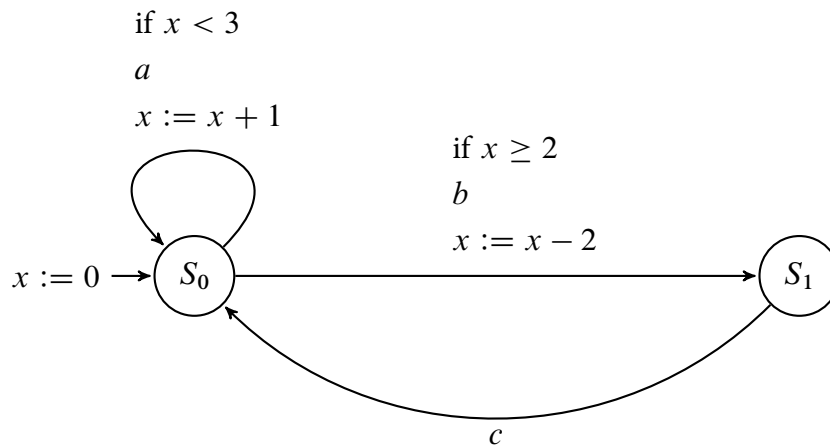


Figure 4.1: An LTS with a variable x described in Example 4.1

Unfolding Usually, the extended version of LTS with assignments and guards is convenient to read but sometimes the formal way of describing transitions becomes unclear. On the other hand, process algebras like IML are basically *behavior-oriented* and the states in the modeled system are only auxiliary elements [24]. The equivalence notations on process algebra, i.e. the bisimilarities concentrate only on the behavior of states. If there are variables in the states, to find the equivalent class of states by applying bisimulation on these LTSs are not feasible. The solution here is to remove the

assignments and guards transitions in our system, in other words, in the case of LTS, we can always find an equivalent LTS without assignment and guard transitions to the original extended version of LTS. The unfolded LTS of Example 4.1 is shown in Figure 4.2.

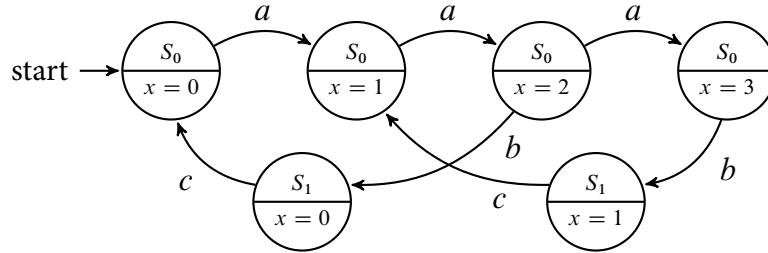


Figure 4.2: An unfolded LTS of Example 4.1

In Figure 4.2, we can easily find the transitions in the unfolded LTS are neither guarded nor have assignments on transitions. We can remove the guards in the unfolded LTS, for a guard transition is possible or not depends on the value of variable x , which is now explicitly valued in each state. Since the value of variable is explicitly integrated into the states, we can easily determine the successor states, which the transitions lead to, from the current state. In Chapter 5, we will let the toolset CADP (Construction and Analysis of Distributed Processes) to unfold the expressions in IML and generate the equivalent IMC for us. Note that we only allow variables to appear in our IML expressions, for the generated IMCs, the transitions are no longer guarded or have assignments on them.

Even when the buffers in SADF specification are first defined as unbounded, the unfold technique is still applicable to obtain an equivalent infinite IMC to represent these channels, due to the fact the values to represent the status of channels are countable. If the values are specified, the possible transitions are uniquely determined. On the other hand, we concentrate only on the SADF models that are *bounded*. An unbounded SADF graph implies some of its channels are unbounded, and to check the non-trivial behavior properties on the unbounded channel systems is undecidable [40]. For bounded SADF, the IMCs to represent the channels must be finite, even though we cannot determine the exact size before we generate the whole IMC for this SADF.

4.1 The elementary IMC models for SADF

The rules of parallel composition and hiding in IMC provide a means to obtain a clean and well-understood semantics of modeling real life systems [20]. We follow this ele-

gant way and first provide an overview of our target model SADF. In an SADF model, four kinds of components are defined with individual functionalities, i.e. kernels, detectors, data channels, and control channels. Briefly we recall the functionalities of these elements in an SADF graph.

A *Kernel* is a data processing component with a life cycle of four steps: 1) it determines the scenario in which it later operates; 2) checks the availability of data tokens; 3) starts to fire (execution); 4) consumes and produces data/control tokens after firing.

A *Detector* is the scenario decision component for its dominating processes with a life circle of four steps: 1) it determines the scenario; 2) determines the subscenario based on the scenario; 3) checks the availability of data tokens; 4) starts to fire; 5) consumes and produces data/control tokens after firing.

A *Data channel* is defined as an unbounded FIFO buffer which can store the data tokens. The process connects the data channel through its output port acts as a “producer” which produces the data tokens and puts them into the channel. The process connects the data channel through its input port acts as a “consumer” which gets the data tokens from the channel and consumes them.

A *Control channel* is defined as an unbounded FIFO buffer which can store scenario tokens. Different from data tokens, scenario tokens are “valued”, i.e., the control tokens consists of the different values that represent different (sub)scenario types. The behavior of the processes may depend on such values of (sub)scenarios.

Before we define the semantics of elements in SADF graph by using IMC, there are some observations from the SADF graph. The data channels can be treated as counters, as the value of tokens is undistinguishable and only the number of tokens stored in the channel influences the behavior of processes. The data channels and control channels are unbounded, i.e., they are modeled as infinite-size buffers. The possibility to bound the buffer size of each components and apply the compositional aggregation approach for state space generation is discussed in Chapter 5.

To take the advantages of IMC as semantic model, we should construct the whole system component-wisely and build the IMC of SADF specification from smaller partial SADF graph until the whole graph through parallel composition and hiding. The decomposition of SADF model can simply follow the idea of the components of SADF, i.e., divide an SADF into kernels, detectors, data channels and control channels. But for efficiency (cf. the first case study in Section 5.4.1), we integrate the channels into the process, which consume the data/control tokens from these channels. In our definition of IMC semantics of SADF, i.e., these channels are modeled as variables in this pro-

cess. We also allow the actions expressed in IML to be guarded and have assignments on them.

To distinguish the participants (i.e., the recipient or sender) during the synchronization in IMC, e.g., $\mathcal{I}_1 \parallel_A \mathcal{I}_2$), in following text, we divide the actions in A into two sets: the *input* actions and *output* actions. We add an overline above an action (e.g., \overline{output}) to indicate it is an output action, otherwise (i.e., no signature is added to the action) it is an input action. If an output action, say \bar{a} is emanated by the sender, and at the other side, a recipient can perform a action a , then an a synchronization happens, and both participants can make a a -action. Note that the input actions are delayable, since the recipient is waiting for another process in the system to perform an output action.

4.1.1 The IMC Semantics of Data Channels

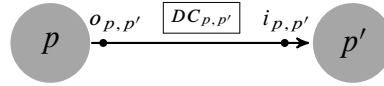


Figure 4.3: A general form of data channel in SADF graph

Figure 4.3 shows a general data channel $DC_{p,p'}$, which connects processes p and p' . The process p has an output port $o_{p,p'} \in \mathcal{O}_p$ and the process p' has an input port $i_{p,p'} \in \mathcal{I}_{p'}$. As aforementioned, data channel can be seen as a counter with aware of its token-number status. So we define the data channel in IML with a variable $cb_{p,p'} \in \mathbb{N}$. There are three main issues of defining the IMC semantics for data channel:

1. The data channels are defined as an unbounded buffers. Although every data channel has bounded buffer size if the SADF is *bounded* [41], the exact size of buffer is still unknown before we get the whole state space by using state exploration approach.
2. The number of tokens to be produced by p or consumed by p' depends on the current scenario in which p or p' is operating. There are two ways to implement this: either the two processes send several “sequential” elementary increase or decrease signals to synchronize with the data channel or they just send their current scenario to the data channel and let the data channel do this for them. We adopt the latter way.
3. The data channel is not accessible from other processes except the two that are connected through this data channel. For instance, in Figure 4.3, p can only produce tokens and put them into the channel and p' can only fetch them out, no

other processes in the of SADF may have the authority. So to avoid awkward state space (caused by synchronization transitions between processes) during generation of the system, we decide to integrate the data channel into the process, which fetches the data tokens from this data channel, as a variable (natural number). We do not separately give a formal semantics of data channel.

Only to give a simple understanding of how a data channel is defined, we define a state variable $db_{p,p'} \in \mathbb{N}$ and give an IMC of a simplified data channel, which can only store and transfer data tokens, and is not aware of the number of tokens in it, by using IML:

$$DC(db_{p,p'}) := \begin{cases} \text{produce. } DC(db_{p,p'}+1) & : \text{ if } db_{p,p'} = 0 \\ \text{produce. } DC(db_{p,p'}+1) + \text{consume. } DC(db_{p,p'}-1) & : \text{ if } db_{p,p'} \geq 1 \end{cases}$$

Figure 4.4 shows the IMC of such an infinite data channel with initial token number $db_{p,p'} = 1$. Note that all actions in the data channel is input actions, which means the data channel will only change its status when the environment requests.

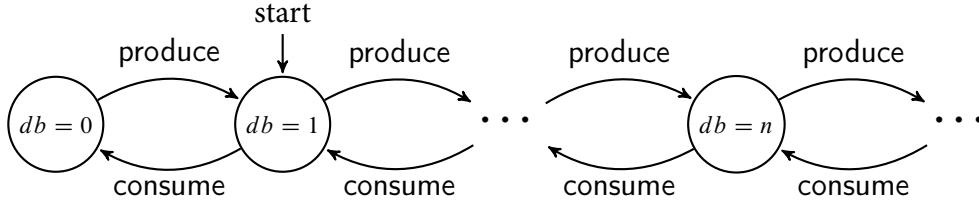


Figure 4.4: The infinite IMC represents an unbounded data channel

4.1.2 The Semantics of Control Channels

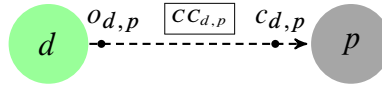


Figure 4.5: A general form of a control channel

Similar as the data channel, we integrate the control channel as a variable into the process, whose control port is connected with this control channel. For the control channel is a scenario-valued unbounded FIFO buffer rather than a counter like data channels, we define a data type called control buffer. To give an intuitive understanding, we define

the control channel separately, and the integrated case of control channel is shown in the kernel's or detector's IML expressions.

First we give an informal description of the data type control buffer, the formal definition is described in Chapter 5 by using the algebraic specification language ACT ONE. Here, we briefly describe this data type informally. We assume the set of token values for (sub)scenarios in a control channel is finite and denote it as Σ . A data type *control buffer* is defined as a list of scenario values (scenario string) and can be denoted as the set Σ^* . A special bottom symbol $[\]$ represents an empty control buffer without any values in it; the infix-function “:” means to add a value to the front of the control buffer, and the value can be fetched only from the top of control buffer; the function *empty* takes a control buffer as argument and returns a boolean: “true”, if the control buffer is empty or “false”, if it has scenario value(s) in it; the function *first* takes a non-empty control buffer as argument and returns its topmost value; a sequence of values stored in the control buffer can be denoted as string xs instead of denoting as $x_n : \dots : x_1$; the function $|xs|$ takes the value string xs as argument and returns the natural number of how many elements in the string. The tail function simply removes the topmost element of the control buffer and returns to the remaining part of control buffer. Instead of adding the values one by one into the control buffer, we the function “++” can be used to concatenate two value strings. The “++” function can be defined as:

$$[x_n : \dots : x_1] ++ [y_m : \dots : y_1] = [x_n : (x_{n-1} \dots : (x_1 : (y_m : \dots : y_1))) \dots]$$

These operations on control buffer will be used in the IML expressions of kernels and detectors. Here we give an example of simplified control channel without integration into the semantics of process semantics:

$$CC(cb_{d,p}) := \begin{cases} \sum_{x \in \Sigma} \text{produce}_x . CC([x]) & : \text{ if } cb_{d,p} = [\], x \in \Sigma \\ \sum_{x \in \Sigma} \text{produce}_x . CC([x:cb_{d,p}]) & : \text{ if } |cb_{d,p}| \geq 1, \text{ and} \\ + \text{consume}_y . CC(\text{tail}(cb_{d,p})) & : y = \text{first}(cb_{d,p}) \end{cases}$$

An LTS of a bounded control channel with capacity=2 and $\Sigma = \{a, b\}$ is shown in Figure 4.6.

4.1.3 The IMC Semantics of Detectors

Figure 4.7 shows a general form of a detector in an SADF graph. Our goal here is to define the semantics of this general detector d by giving

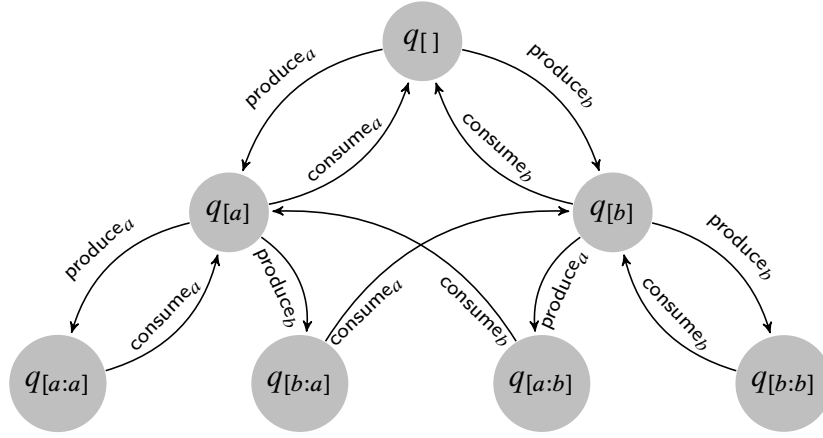


Figure 4.6: An example of a bounded control channel

an equivalent IMC. Recall that from Chapter 3, a detector d is a tuple $(\mathcal{I}_d, \mathcal{O}_d, \mathcal{C}_d, \mathcal{S}_d, \{(\mathbb{S}_d^s, t_d^s, \mathbb{P}_d^s, \Phi_d^s) \mid s \in \mathcal{S}_d\}, \Omega_d, \{(R_d^\omega, t_d^\omega, E_d^\omega) \mid \omega \in \Omega_d\})$. Generally, we assume a detector d is connected with detectors $d_1, \dots, d_i, d_{i+1}, \dots, d_j$ and kernels $k_1, \dots, k_m, k_{m+1}, \dots, k_n$, moreover we call $d_1, \dots, d_i, k_1, \dots, k_m$ the successor processes of d and call $d_{i+1}, \dots, d_j, k_{m+1}, \dots, k_n$ the predecessor processes of d . We also assume the functions $R_d^\omega, t_d^\omega, E_d^\omega$ for each subscenario $\omega \in \Omega_d$ and the finite-state discrete-time Markov chains $(\mathbb{S}_d^s, t_d^s, \mathbb{P}_d^s, \Phi_d^s)$ for capturing the occurrences associated with each scenario $s \in \mathcal{S}_d$ are given. Note that the generic discrete execution time distributions of detector d are now adapted to single execution delays which are governed by exponential distributions. Every such distribution is characterized by a single parameter $\lambda \in \mathbb{R}^+$.

We identify the channels by the processes that are connected through these directed channels, e.g., if a set of control channels from a detector d' pointing to a detector d , then these control channels are identified as $CC_{d',d}^1, \dots, CC_{d',d}^{z^{d',d}}$. $z^{d',d}$ is the number of these control channels from d' to d . Similar as control channels, the set of data channels from d' to d is also identified as $DC_{d',d}^1, \dots, DC_{d',d}^{y^{d',d}}$, with $y^{d',d}$ is the number of these data channels. Note that from a kernel k' to a detector d , there are only data channels possible to connect them, so these data channels are identified as $DC_{k',d}^1, \dots, DC_{k',d}^{x^{k',d}}$. The input, output and control ports are also identified in the same way, e.g., d' and d are connected by data and control channels $DC_{d',d}^1, \dots, DC_{d',d}^{y^{d',d}}, CC_{d',d}^1, \dots, CC_{d',d}^{z^{d',d}}$ through the output ports $\{oi_{d',d}^1, \dots, oi_{d',d}^{y^{d',d}}, oc_{d',d}^1, \dots, oc_{d',d}^{z^{d',d}}\} \in \mathcal{O}_{d'}$, the input ports $\{i_{d',d}^1, \dots, i_{d',d}^{y^{d',d}}\} \in \mathcal{I}_d$ and the control ports $\{c_{d',d}^1, \dots, c_{d',d}^{z^{d',d}}\} \in \mathcal{C}_d$. Especially, in the original SADF

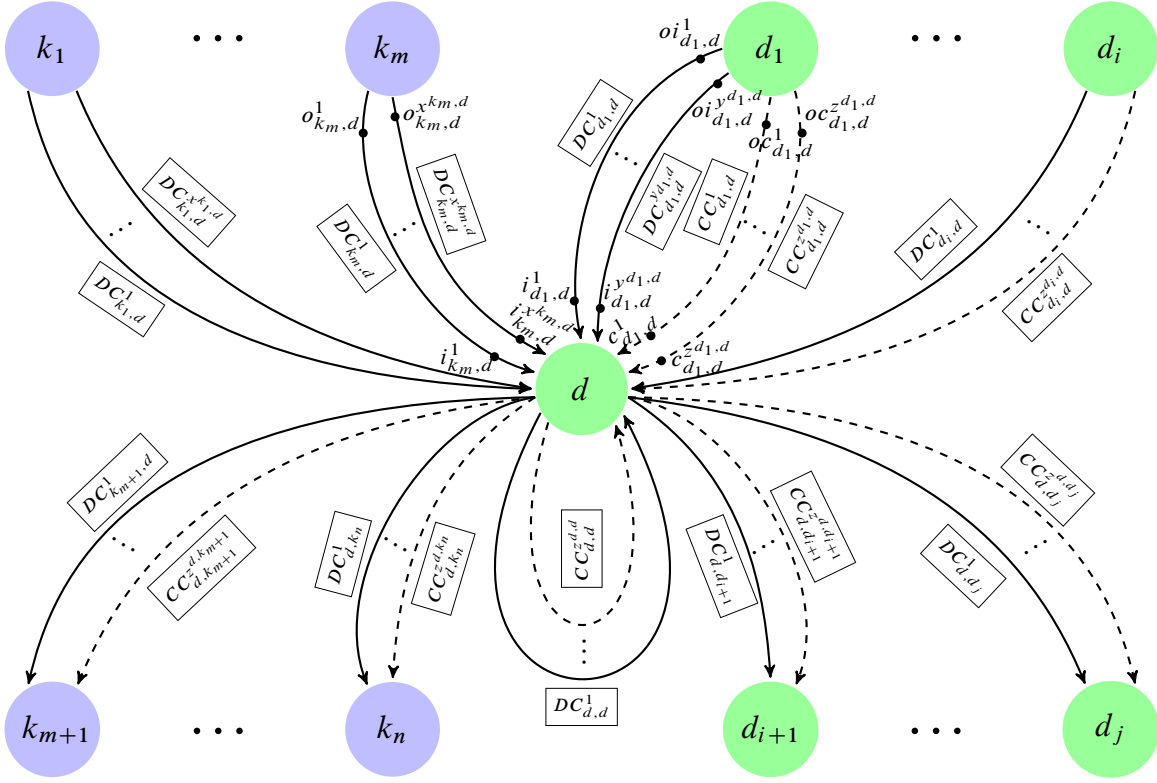


Figure 4.7: A general form of a detector

semantics allows the detector d to have data and control channels connected with itself. These channels will be identified as $DC^1_{d,d}, \dots, DC^{y^{d,d}}_{d,d}$ with input and output ports $i^1_{d,d}, \dots, i^{y^{d,d}}_{d,d}$, $oi^1_{d,d}, \dots, oi^{y^{d,d}}_{d,d}$ and $CC^1_{d,d}, \dots, CC^{z^{d,d}}_{d,d}$ with control and output ports $c^1_{d,d}, \dots, c^{z^{d,d}}_{d,d}$, $oc^1_{d,d}, \dots, oc^{z^{d,d}}_{d,d}$. Then we have

$$\begin{aligned} \mathcal{I}_d &= \bigcup_{k' \in \{k_1, \dots, k_m\}} \{i^1_{k',d}, \dots, i^{x^{k',d}}_{k',d}\} \cup \bigcup_{d' \in \{d_1, \dots, d_i\}} \{i^1_{d',d}, \dots, i^{y^{d',d}}_{d',d}\} \\ \mathcal{C}_d &= \bigcup_{d' \in \{d, d_1, \dots, d_i\}} \{c^1_{d',d}, \dots, c^{z^{d',d}}_{d',d}\} \\ \mathcal{O}_d &= \bigcup_{k' \in \{k_{m+1}, \dots, k_n\}} \{oi^1_{k',d}, \dots, oi^{y^{k',d}}_{k',d}\} \cup \bigcup_{k' \in \{k_{m+1}, \dots, k_n\}} \{oc^1_{k',d}, \dots, oc^{z^{k',d}}_{k',d}\} \\ &\quad \cup \bigcup_{d' \in \{d, d_{i+1}, \dots, d_j\}} \{oi^1_{d',d}, \dots, oi^{y^{d',d}}_{d',d}\} \cup \bigcup_{d' \in \{d, d_{i+1}, \dots, d_j\}} \{oc^1_{d',d}, \dots, oc^{z^{d',d}}_{d',d}\} \end{aligned}$$

For $k' \in \{k_1, \dots, k_m\}$ and $d' \in \{d_1, \dots, d_i\}$, we have :

$$\mathcal{O}_{k'} \supseteq \{o_{k',d}^1 \dots o_{k',d}^{x_{k',d}}\}, \mathcal{O}_{d'} \supseteq \{oi_{d',d}^1 \dots oi_{d',d}^{y_{d',d}}\} \cup \{oc_{d',d}^1 \dots oc_{d',d}^{z_{d',d}}\}.$$

For the purpose of clarity, we divide the IMC semantics of detector into two modules. The first module is called the *function module* and the second module is called the *subscenario-specification module*. The function module of the detector is the backbone of a detector with following tasks: It decides the current scenario for subscenario-specification module based on the values of its control ports, then it passes the scenario to the scenario-specification module, soon it can receive the subscenario values from the scenario-specification module, then it determines the availability of each input ports depending on the subscenario. After it is aware that enough tokens are available at each of its input ports, it starts to fire. After firing, it consumes data tokens and scenario tokens from each channel differently based on the value of the subscenario. The subscenario-specification module is based on the set of discrete-time Markov chains $\{(\mathbb{S}_d^s, \iota_d^s, \mathbb{P}_d^s, \Phi_d^s) \mid s \in \mathcal{S}_d\}$ with the task of determining the subscenario for the function module. Note that the scenario-specification module can send more than one subscenarios back the function module, because it determine the subscenarios probabilistically.

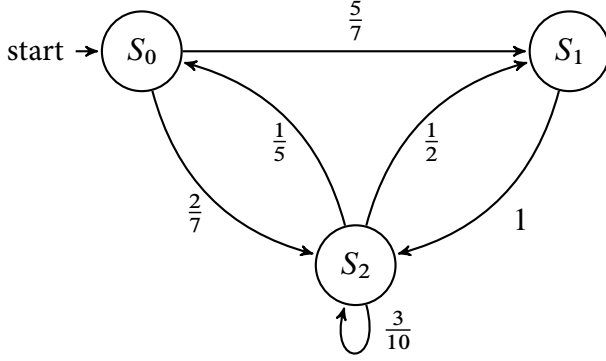
The Semantics of Subscenario-specification Module

Our first step is to define an IMC to express the finite-state discrete-time Markov chain $(\mathbb{S}_d^s, \iota_d^s, \mathbb{P}_d^s, \Phi_d^s)$ for each scenario $s \in \mathcal{S}_d$. We assume $\mathbb{S}_d^s = \{S_0, S_1, \dots, S_x\}$, $\iota_d^s = S_0$ and \mathbb{P}_d^s is the matrix of one-step transition probabilistic with $\sum_{y \in \mathbb{S}} \mathbb{P}(x, y) = 1$ for all $x \in \mathbb{S}_d^s$. We also define the set $T_{S_i}^s = \{T \mid \mathbb{P}_d^s(S_i, T) > 0\}$, $S_i \in \mathbb{S}_d^s$. An example of such a finite-state discrete-time Markov chain is shown in Figure 4.8.

If we have such a set of Markov chains associated with scenarios, we first define an IMC in IML for each scenario. We choose a scenario $s \in \mathcal{S}_d$ and its corresponding Markov chain $(\mathbb{S}_d^s, \iota_d^s, \mathbb{P}_d^s, \Phi_d^s)$ as a representative.

Definition 4.1 (The IMC semantics of scenario-specification module for scenario s) The IMC semantics of the scenario-specification module for scenario s is described by following IML expressions:

$$SP_d^s := SP_{S_0}^s \quad (4.1)$$



The Φ -function

$$\Phi(S_0) = \omega_1$$

$$\Phi(S_1) = \omega_2$$

$$\Phi(S_2) = \omega_3$$

Figure 4.8: An example of $(\mathbb{S}, \iota, \mathbb{P}, \Phi)$ in a detector

$$SP_{S_i}^s := \text{req}_s . SP_{S_i'}^s \quad (4.2)$$

$$SP_{S_i'}^s := \sum_{T \in T_{S_i}^s} \text{Prob}_d^{(s, S_i, T)} . SP_{T''}^s \quad (4.3)$$

$$SP_{S_i''}^s := \overline{\Phi_d^s(S_i)} . SP_{S_i}^s \quad (4.4)$$

The first expression (4.1) says that we begin with the initial state S_0 in the Markov chain. The second expression (4.2) means that for every state $S_i \in \mathbb{S}_d^s$, if the scenario-specification module receives a signal labelled with scenario s from the function module of detector, they will synchronize with this signal and the scenario-specification module will behave like the expression $SP_{S_i'}^s$. This req_s signal tells the scenario-specification module that the prerequisite to decide a subscenario in scenario s is fulfilled and asks the subscenario-specification module to finish this task. After this transition, in expression (4.3), the subscenario-specification module must decide which are the successor states of S_i and with what probability. This is done by checking the given matrix of one-step transition probabilities in SADF. Assume the set of successor states of the current state S_i is $T_{S_i}^s$, i.e. the set of states $T \in \mathbb{S}_d^s$ such that $\mathbb{P}_d^s(S_i, T) > 0$. For the state T , we do not make a transition direct to T , but add an additional intermediate state T'' which belongs to the state T , this state is reached by a probabilistic transition according to \mathbb{P}_d^s from state S . Since IMC does not directly support any probabilistic transitions, these probabilistic transitions are first represented by normal transitions labelled with $\sum_{T \in T_{S_i}^s} \text{Prob}_d^{(s, S_i, T)}$, i.e., we have a non-deterministic “action” for each probabilistic option origination from the discrete-time Markov chains associate to the detector d and the scenario s . In this way, the probabilistic transitions in the Markov chains associate to the detectors are unique labelled (associated) with the transitions in the resulting IMC. Latter, we will use $\mathbb{P}_d^s(S_i, T)$ to replace the transitions labelled with $\text{Prob}_d^{(s, S_i, T)}$ as a scheduler to solve the non-determinism in IMC. The unique identifica-

tion of probabilistic transitions in IMC and the forbid of these action names to appear in the synchronization set avoid these “probabilistic choice actions” to be synchronized with other actions. From the intermediate state T'' , the only possible transition is to send a subscenario $\Phi_d^s(T)$ back to the function module (expression 4.4) and then the state T is reached. To replace these transitions back to real probabilistic transitions will be discussed in Chapter 5.

Example 4.2 Apply our definition to the Markov chain and Φ -function in Figure 4.8, we obtain the following expressions. In the square brackets $[\dots]$ is the actual probabilistic information of this transition. The corresponding IMC is illustrated in Figure 4.9.

$$\begin{aligned}
SP_d^{\epsilon D} &:= SP_{S_0}^{\epsilon D} \\
SP_{S_0}^{\epsilon D} &:= \text{req}_{\epsilon D}.SP_{S_0'}^{\epsilon D} \\
SP_{S_0'}^{\epsilon D} &:= \text{Prob}_d^{(\epsilon D, S_0, S_1)}[\frac{5}{7}].SP_{S_1''}^{\epsilon D} + \text{Prob}_d^{(\epsilon D, S_0, S_2)}[\frac{2}{7}].SP_{S_2''}^{\epsilon D} \\
SP_{S_0''}^s &:= \overline{\omega_1}.SP_{S_0}^s \\
\\
SP_{S_1}^{\epsilon D} &:= \text{req}_{\epsilon D}.SP_{S_1'}^{\epsilon D} \\
SP_{S_1'}^{\epsilon D} &:= \text{Prob}_d^{(\epsilon D, S_1, S_2)}[1].SP_{S_2''}^{\epsilon D} \\
SP_{S_1''}^{\epsilon D} &:= \overline{\omega_2}.SP_{S_1}^{\epsilon D} \\
\\
SP_{S_2}^{\epsilon D} &:= \text{req}_{\epsilon D}.SP_{S_2'}^{\epsilon D} \\
SP_{S_2'}^{\epsilon D} &:= \text{Prob}_d^{(\epsilon D, S_2, S_0)}[\frac{1}{5}].SP_{S_0''}^{\epsilon D} + \text{Prob}_d^{(\epsilon D, S_2, S_1)}[\frac{1}{2}].SP_{S_1''}^{\epsilon D} + \text{Prob}_d^{(\epsilon D, S_2, S_2)}[\frac{3}{10}].SP_{S_2''}^{\epsilon D} \\
SP_{S_2''}^{\epsilon D} &:= \overline{\omega_3}.SP_{S_2}^{\epsilon D}
\end{aligned}$$

Definition 4.2 (The IMC semantics of scenario-specification module of SADF). Let SP_d^s to be the IMCs according to discrete-time Markov chains in SADF for each $s \in \mathcal{S}_d$, then the whole IMC SP_d for the scenario specification part is defined as:

$$SP_d = SP_d^{s_1} \parallel SP_d^{s_2} \parallel \dots \parallel SP_d^{s_n} \quad (\mathcal{S}_d = \{s_1, s_2, \dots, s_n\})$$

The Semantics of Function Module

The general form of a detector d is shown in Figure 4.7. Since we integrate the data channels and control channels as variables into the process, which consumes the data

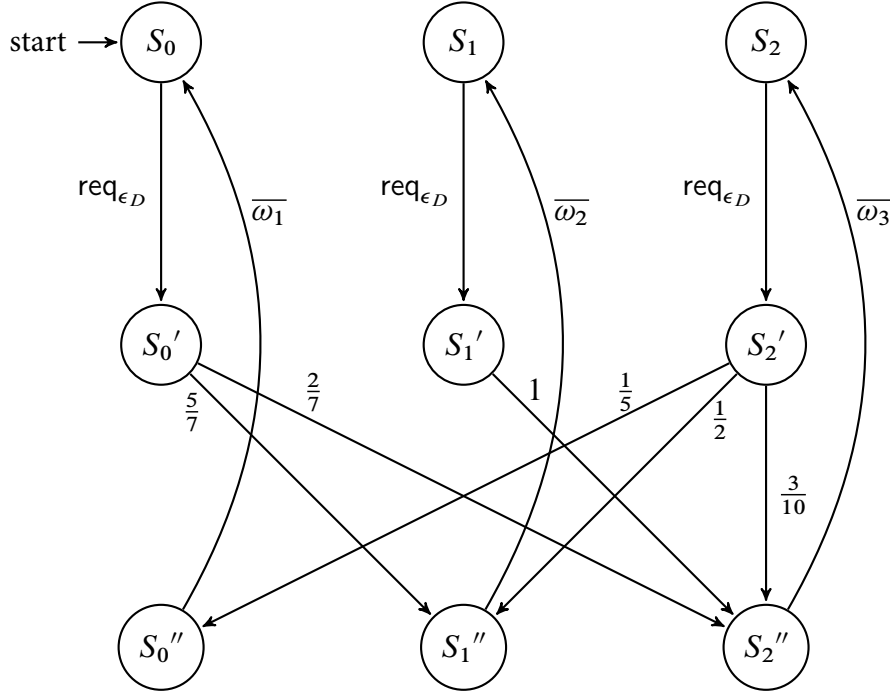


Figure 4.9: The IMC semantics of the Markov chain $(\mathbb{S}_d^{\epsilon_D}, \iota_d^{\epsilon_D}, \mathbb{P}_d^{\epsilon_D}, \Phi_d^{\epsilon_D})$

tokens or scenario tokens from these channels, we need to synchronize with all the processes connected with d . Note that only the channels between the detector d and its predecessor processes are integrated into d . The channels connected with d 's output ports belong to its successor process respectively. Then d should integrate the channels from processes $k_1, \dots, k_m, d_1, \dots, d_i$ as variables in its expressions. For each kernel $k' \in \{k_1, \dots, k_m\}$, the data channels to integrate in the the detector d are $DC_{k',d}^1, \dots, DC_{k',d}^{x^{k',d}}$. We use $\mathcal{DB}_{k',d} = (db_{k',d}^1, \dots, db_{k',d}^{x^{k',d}})$ as the variables which are of type natural number to represent the status of these channels. For each detector $d' \in \{d, d_1, \dots, d_i\}$, the data channels from d' to d are $DC_{d',d}^1, \dots, DC_{d',d}^{y^{d',d}}$, and the control channels from d' to d are $CC_{d',d}^1, \dots, CC_{d',d}^{z^{d',d}}$. We integrate these data channels as variables of type natural number: $\mathcal{DB}_{d',d} = (db_{d',d}^1, \dots, db_{d',d}^{y^{d',d}})$, and integrate these control channels as variables of predefined type *control buffer*: $\mathcal{CB}_{d',d} = (cb_{d',d}^1, \dots, cb_{d',d}^{z^{d',d}})$. We denote the subscenario set Φ_d^s belongs to each scenario s as Ω_d^s .

Definition 4.3 (The IMC semantics of function module) *The IMC semantics of the function module in detector d is described by the following IML expressions:*

$$FP_d(\mathcal{DB}_{d,d}, \mathcal{CB}_{d,d}, \mathcal{DB}_{k_1,d}, \dots, \mathcal{DB}_{k_m,d}, \mathcal{DB}_{d_1,d}, \mathcal{CB}_{d_1,d}, \dots, \mathcal{DB}_{d_i,d}, \mathcal{CB}_{d_i,d}) :=$$

$$\sum_{s \in \mathcal{S}_d} \overline{\text{req}}_s \cdot FP_d^s \left(\mathcal{DB}_{d,d}, \mathcal{CB}_{d,d}, \mathcal{DB}_{k_1,d}, \dots, \mathcal{DB}_{k_m,d}, \mathcal{DB}_{d_1,d}, \mathcal{CB}_{d_1,d}, \dots, \mathcal{DB}_{d_i,d}, \mathcal{CB}_{d_i,d} \right) :$$

$$\text{if } |cb_{d,d}^1| > 0 \wedge |cb_{d,d}^{z^{d,d}}| > 0 \wedge \dots \wedge |cb_{d_i,d}^1| > 0 \wedge \dots \wedge |cb_{d_i,d}^{z^{d_i,d}}| > 0 \wedge$$

$$\text{first}(cb_{d,d}^1) \times \dots \times \text{first}(cb_{d,d}^{z^{d,d}}) \times \dots \times \text{first}(cb_{d_i,d}^1) \times \dots \times \text{first}(cb_{d_i,d}^{z^{d_i,d}}) = s, s \in \mathcal{S}_k \quad (4.5)$$

$$+ \sum_{k' \in \{k_1, \dots, k_m\}} \sum_{s' \in \mathcal{S}_{k'}} \text{produce}_{k'}^{s'} \cdot FD_d \left(\dots, (db_{k',d}^1 + R_{k'}^{s'}(o_{k',d}^1)), \dots, (db_{k',d}^{x^{k',d}} + R_{k'}^{s'}(o_{k',d}^{x^{k',d}})), \dots \right) \quad (4.6)$$

$$+ \sum_{d' \in \{d_1, \dots, d_i\}} \sum_{\omega' \in \Omega_{d'}} \text{produce}_{d'}^{\omega'} \cdot FD_d \left(\dots, (db_{d',d}^1 + R_{d'}^{\omega'}(oi_{d',d}^1)), \dots, (db_{d',d}^{y^{d',d}} + R_{d'}^{\omega'}(oi_{d',d}^{y^{d',d}})), \dots \right)$$

$$\left([t_{d'}^{\omega'}(oc_{d',d}^1)] + cb_{d',d}^1, \dots, ([t_{d'}^{\omega'}(oc_{d',d}^{z^{d',d}})] + cb_{d',d}^{z^{d',d}}), \dots \right) \quad (4.7)$$

$$FP_d^s \left(\mathcal{DB}_{d,d}, \mathcal{CB}_{d,d}, \mathcal{DB}_{k_1,d}, \dots, \mathcal{DB}_{k_m,d}, \mathcal{DB}_{d_1,d}, \mathcal{CB}_{d_1,d}, \dots, \mathcal{DB}_{d_i,d}, \mathcal{CB}_{d_i,d} \right) :=$$

$$\sum_{\omega \in \Omega_d^s} \omega \cdot FP_d^\omega \left(\mathcal{DB}_{d,d}, \mathcal{CB}_{d,d}, \mathcal{DB}_{k_1,d}, \dots, \mathcal{DB}_{k_m,d}, \mathcal{DB}_{d_1,d}, \mathcal{CB}_{d_1,d}, \dots, \mathcal{DB}_{d_i,d}, \mathcal{CB}_{d_i,d} \right) \quad (4.8)$$

$$FP_d^\omega \left(\mathcal{DB}_{d,d}, \mathcal{CB}_{d,d}, \mathcal{DB}_{k_1,d}, \dots, \mathcal{DB}_{k_m,d}, \mathcal{DB}_{d_1,d}, \mathcal{CB}_{d_1,d}, \dots, \mathcal{DB}_{d_i,d}, \mathcal{CB}_{d_i,d} \right) :=$$

$$\lambda_d^\omega \cdot \overline{\text{produce}}_d^\omega \cdot FP_d \left((db_{d,d}^1 - R_d^\omega(i_{d,d}^1) + R_d^\omega(oi_{d,d}^1)), \dots, (db_{d,d}^{y^{d,d}} - R_d^\omega(i_{d,d}^{y^{d,d}}) + R_d^\omega(oi_{d,d}^{y^{d,d}})), \dots, \right.$$

$$\text{tail}(cb_{d,d}^1), \dots, \text{tail}(cb_{d,d}^{z^{d,d}}), (db_{k_1,d}^1 - R_d^\omega(i_{k_1,d}^1)), \dots, (db_{k_1,d}^{x^{k_1,d}} - R_d^\omega(i_{k_1,d}^{x^{k_1,d}})), \dots, (db_{k_m,d}^1 - R_d^\omega(i_{k_m,d}^1)), \dots,$$

$$(db_{k_m,d}^{x^{k_m,d}} - R_d^\omega(i_{k_m,d}^{x^{k_m,d}})), (db_{d_1,d}^1 - R_d^\omega(i_{d_1,d}^1)), \dots, (db_{d_1,d}^{y^{d_1,d}} - R_d^\omega(i_{d_1,d}^{y^{d_1,d}})), \text{tail}(cb_{d_1,d}^1), \dots,$$

$$\left. \text{tail}(cb_{d_1,d}^{z^{d_1,d}}), \dots, (db_{d_m,d}^1 - R_d^\omega(i_{d_m,d}^1)), \dots, (db_{d_m,d}^{y^{d_m,d}} - R_d^\omega(i_{d_m,d}^{y^{d_m,d}})), \text{tail}(cb_{d_m,d}^1), \dots, \text{tail}(cb_{d_m,d}^{z^{d_m,d}}) \right) :$$

$$\text{if } db_{d,d}^1 \geq R_d^\omega(i_{d,d}^1) \wedge \dots \wedge db_{d,d}^{x^{d,d}} \geq R_d^\omega(i_{d,d}^{x^{d,d}}) \wedge db_{k_1,d}^1 \geq R_d^\omega(i_{k_1,d}^1) \wedge \dots \wedge db_{k_1,d}^{x^{k_1,d}} \geq R_d^\omega(i_{k_1,d}^{x^{k_1,d}}) \wedge$$

$$\wedge \dots \wedge db_{k_m,d}^1 \geq R_d^\omega(i_{k_m,d}^1) \wedge \dots \wedge db_{k_m,d}^{x^{k_m,d}} \geq R_d^\omega(i_{k_m,d}^{x^{k_m,d}}) \wedge db_{d_1,d}^1 \geq R_d^\omega(i_{d_1,d}^1) \wedge \dots$$

$$\wedge db_{d_1,d}^{y^{d_1,d}} \geq R_d^\omega(i_{d_1,d}^{y^{d_1,d}}) \wedge \dots \wedge db_{d_i,d}^1 \geq R_d^\omega(i_{d_i,d}^1) \wedge \dots \wedge db_{d_i,d}^{y^{d_i,d}} \geq R_d^\omega(i_{d_i,d}^{y^{d_i,d}}) \quad (4.9)$$

$$+ \sum_{k' \in \{k_1, \dots, k_m\}} \sum_{s' \in \mathcal{S}_{k'}} \text{produce}_{k'}^{s'} \cdot FD_d^\omega \left(\dots, (db_{k',d}^1 + R_{k'}^{s'}(o_{k',d}^1)), \dots, (db_{k',d}^{x^{k',d}} + R_{k'}^{s'}(o_{k',d}^{x^{k',d}})), \dots \right) \quad (4.10)$$

$$+ \sum_{d' \in \{d_1, \dots, d_i\}} \sum_{\omega' \in \Omega_{d'}} \text{produce}_{d'}^{\omega'} \cdot FD_d^\omega \left(\dots, (db_{d',d}^1 + R_{d'}^{\omega'}(oi_{d',d}^1)), \dots, (db_{d',d}^{y^{d',d}} + R_{d'}^{\omega'}(oi_{d',d}^{y^{d',d}})), \dots \right)$$

$$\left([t_{d'}^{\omega'}(oc_{d',d}^1)] + cb_{d',d}^1, \dots, ([t_{d'}^{\omega'}(oc_{d',d}^{z^{d',d}})] + cb_{d',d}^{z^{d',d}}), \dots \right) \quad (4.11)$$

In these IML expressions, the variables $db_{p',d}^1 \dots db_{p',d}^{x(y)^{p',d}}$ represent the current status (number of data tokens) of the corresponding data channels from a process p' to d , and the variables $cb_{d',d}^1, \dots, cb_{d',d}^{z^{d',d}}$ represent the current status (scenario string) of

the corresponding control channels from a detector d' to d . Initially the detector d can have three possible kinds of actions.

1. As soon as each control channel, which is connected with d 's control port, is not empty, d can start to determine the needed subscenario for its coming firing. It first determines the scenario which is to be sent to the scenario-specification module, the scenario is just the value sequence of the first scenario token of each control channel: $\text{first}(cb_{d,d}^1) \times \dots \times \text{first}(cb_{d_i,d}^{z^{d_i,d}}) = s \in \mathcal{S}_d$. Then d sends a signal with the scenario information s to the scenario-specification module with the request of the subscenario (expression 4.5). The scenario-specification module returns with the subscenario(s) to the function module. The function module stores the subscenario and moves on to the expression $FP_d^\omega(\dots)$ (expression 4.8). This can be seen as the equivalent semantic part to the "detect(d)" transition in TPS semantics of SADF.
2. If d has received a synchronization signal from its successor processes, it must modify the variables correspondingly. In our example, if d has received a produce $_{k'}^{s'}$ signal from kernel k' , then d knows k' has finished its firing and then d checks the functions $R_k^s(o_{k',d}^1), \dots, R_k^s(o_{k',d}^{x^{k',d}})$. These functions return the information of how many tokens d need to be added to the corresponding data channels, and d modifies the variable $db_{k',d}$ accordingly (expression 4.6).
3. The synchronization signals from successor detectors of d follow the same way but extended with the variables $cb_{d',d}^1, \dots, cb_{d',d}^{z^{d',d}}$ are of type a control buffer and the result is a string of scenario values. This string of scenario values is concatenated with the end of the control buffer by using the operator ++.

The expression $FP_d^\omega(\dots)$, which bears the ω subscenario needed for later firing, starts to determine the data-token availability of each data channel, i.e. the data channels $DC_{k_1,d}^1, \dots, DC_{k_m,d}^{x^{k_m,d}}, DC_{d,d}^1, \dots, DC_{d_i,d}^{y^{d_i,d}}$. The data-token availability is to be checked with the aid of the functions $R_d^\omega(i_{k_1,d}^1), \dots, R_d^\omega(i_{d_i,d}^{y^{d_i,d}})$. If the data tokens are enough for the subscenario ω in all these data channels, d starts to fire. We assume the execution time of d in subscenario ω is a random variable governed by an exponential distribution and characterized by the rate λ_d^ω . Recall that the generic discrete execution time distributions in original SADF specification is now replaced by the rate $\lambda_d^\omega = 1/e_d^\omega$, where $e_d^\omega = (\mathbb{P}(E_d^\omega = e_1) \cdot e_1 + \dots + \mathbb{P}(E_d^\omega = e_n) \cdot e_n)$, where e_1, \dots, e_n , which represent a exponentially distributed delay with a mean duration of e_d^ω time units. The Markovian transitions in IMC can be seen as the corresponding part of *time transitions* TPS semantics. After d finished its firing, d starts to consume

and produce data tokens and scenario tokens in each data/control channels, which connect with it. Since d itself has the variables representing the status of each channel which connect d with its predecessor processes, this is done by modifying the variables using the predefined functions on the corresponding data types. The status of channels, which are connected through the output port of d , needs to be modified by sending a synchronization signal “produce $_d^\omega$ ”, which means to the successor processes of d , that d has finished its firing in subscenario ω , and modify the status of corresponding control buffers according to the $R_d^\omega(o)$ function. This is equivalent to the “end(d)” transitions in TPS semantics. After d has determined the subscenario ω for later execution, FP_d^ω need also to allow the synchronization signals from its predecessor processes to modify the status of the channels (expression 4.10, 4.11).

Note that the sequential transitions req $_s$ and ω , λ_d^ω and produce $_d^\omega$ are seen as atomic, i.e., between these transitions, there are no produce synchronization requests from other processes allowed. The reason that no other transitions can interrupt the transitions between req $_s$ and ω is the request for subscenario costs no time, it is assumed these transition are internal transitions and happens instantaneously. The Markovian transition λ_d^ω and interactive transition produce $_d^\omega$ are also non-divisible, since the channel status modifications on each channel should happen directly after d 's firing.

Now we can get our whole IMC semantics I_d of detector d as follows:

$$I_d = D (\mathcal{DB}_{d,d}, \mathcal{CB}_{d,d}, \mathcal{DB}_{k_1,d}, \dots, \mathcal{DB}_{k_m,d}, \mathcal{DB}_{d_1,d}, \mathcal{CB}_{d_1,d}, \dots, \mathcal{DB}_{d_i,d}, \mathcal{CB}_{d_i,d}) := \\ (FP_d (\mathcal{DB}_{d,d}, \mathcal{CB}_{d,d}, \mathcal{DB}_{k_1,d}, \dots, \mathcal{DB}_{k_m,d}, \mathcal{DB}_{d_1,d}, \mathcal{CB}_{d_1,d}, \dots, \mathcal{DB}_{d_i,d}, \mathcal{CB}_{d_i,d}) \parallel_{A_d} SP_d) \setminus A_d \\ \text{where } A_d = \{ \text{req}_s \mid s \in \mathcal{S}_d \} \cup \Omega_d.$$

4.1.4 The Semantics of Kernels

Kernels are the functional components in SADF, they represent the data processing part of a streaming application. Different from detectors, kernels do not have a set of sub-scenarios and can determine directly the scenario needed for the coming firing from the value of the first scenario-token in each control ports. Note that for every process p , the function $R_p^s(c) = 1$, for every control port $c \in \mathcal{C}_p$ and every s belongs to the (sub)scenario set of p . Here, the way to identify the channels and ports is the same as the case in the detector. Therefore, a general form of kernel is shown in Figure 4.10.

Definition 4.4 (The IMC Semantics of Kernels) *The IMC semantics of a kernel in a*

$$\begin{aligned}
& \lambda_k^s \cdot \overline{\text{produce}}_k^s . K \left((db_{k,k}^1 - R_k^s(i_{k,k}^1) + R_k^s(o_{k,k}^1)), \dots, (db_{k,k}^{x_{k,k}} - R_k^s(i_{k,k}^{x_{k,k}}) + R_k^s(o_{k,k}^{x_{k,k}})), \right. \\
& (db_{k_1,k}^1 - R_k^s(i_{k_1,k}^1)), \dots, (db_{k_1,k}^{x_{k_1,k}} - R_k^s(i_{k_1,k}^{x_{k_1,k}})), \dots, (db_{k_m,k}^1 - R_k^s(i_{k_m,k}^1)), \dots, (db_{k_m,k}^{x_{k_m,k}} - R_k^s(i_{k_m,k}^{x_{k_m,k}})), \\
& (db_{d_1,k}^1 - R_k^s(i_{d_1,k}^1)), \dots, (db_{d_1,k}^{y_{d_1,k}} - R_k^s(i_{d_1,k}^{y_{d_1,k}})), \text{tail}(cb_{d_1,k}^1), \dots, \text{tail}(cb_{d_1,k}^{z_{d_1,k}}), \dots, \\
& \left. (db_{d_m,k}^1 - R_k^s(i_{d_m,k}^1)), \dots, (db_{d_m,k}^{y_{d_m,k}} - R_k^s(i_{d_m,k}^{y_{d_m,k}})), \text{tail}(cb_{d_m,k}^1), \dots, \text{tail}(cb_{d_m,k}^{z_{d_m,k}}) \right) : \\
& \text{if } db_{k,k}^1 \geq R_k^s(i_{k,k}^1) \wedge \dots \wedge db_{k,k}^{x_{k,k}} \geq R_k^s(i_{k,k}^{x_{k,k}}) \wedge db_{k_1,k}^1 \geq R_k^s(i_{k_1,k}^1) \wedge \dots \wedge db_{k_1,k}^{x_{k_1,k}} \geq R_k^s(i_{k_1,k}^{x_{k_1,k}}) \wedge \\
& \wedge \dots \wedge db_{k_m,k}^1 \geq R_k^s(i_{k_m,k}^1) \wedge \dots \wedge db_{k_m,k}^{x_{k_m,k}} \geq R_k^s(i_{k_m,k}^{x_{k_m,k}}) \wedge db_{d_1,k}^1 \geq R_k^s(i_{d_1,k}^1) \wedge \dots \\
& \wedge db_{d_1,k}^{y_{d_1,k}} \geq R_k^s(i_{d_1,k}^{y_{d_1,k}}) \wedge \dots \wedge db_{d_i,k}^1 \geq R_k^s(i_{d_i,k}^1) \wedge \dots \wedge db_{d_i,k}^{y_{d_i,k}} \geq R_k^s(i_{d_i,k}^{y_{d_i,k}}) \quad (4.15)
\end{aligned}$$

$$\begin{aligned}
& + \sum_{k' \in \{k_1, \dots, k_m\}} \sum_{s' \in \mathcal{S}_{k'}} \text{produce}_{k'}^{s'} . K^s \left(\dots, (db_{k',k}^{s'} + R_{k'}^{s'}(o_{k',k}^{s'})), \dots, (db_{k',k}^{x_{k',k}} + R_{k'}^{s'}(o_{k',k}^{x_{k',k}})), \dots \right) \\
& \quad (4.16)
\end{aligned}$$

$$\begin{aligned}
& + \sum_{d' \in \{d_1, \dots, d_i\}} \sum_{\omega \in \Omega_{d'}} \text{produce}_{d'}^\omega . K^s \left(\dots, (db_{d',k}^1 + R_{d'}^\omega(o_{d',k}^1)), \dots, (db_{d',k}^{y_{d',k}} + R_{d'}^\omega(o_{d',k}^{y_{d',k}})), \right. \\
& \left. ([t_{d'}^\omega(oc_{d',k}^1)] + cb_{d',k}^1), \dots, ([t_{d'}^\omega(oc_{d',k}^{z_{d',k}})] + cb_{d',k}^{z_{d',k}}), \dots \right) \quad (4.17)
\end{aligned}$$

In IML we define a kernel as an expression with variables $db_{k_1,k}^1, \dots, db_{k_m,k}^{x_{k_m,k}}, db_{d_1,k}^1, \dots, db_{d_i,k}^{y_{d_i,k}}$ as natural numbers and variables $cb_{d_1,k}^1, \dots, cb_{d_i,k}^{z_{d_i,k}}$ as control buffers. These variables represent the status of the corresponding channels. The operations on data type control buffer are described in section 4.1.2. The expression K has three possible kinds of actions:

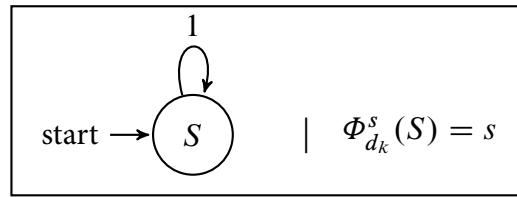
1. If the status of the channels satisfies the prerequisite to fire, i.e., the kernel has determined the scenario s for subsequent firing from control channels $CC_{d_1,k}^1, \dots, CC_{d_i,k}^{z_{d_i,k}}$ and the data tokens are adequate in each data channel $DC_{k,k}^1, \dots, DC_{k_m,k}^{x_{k_m,k}}, DC_{d_1,k}^1, \dots, DC_{d_i,k}^{y_{d_i,k}}$ for the scenario s , the kernel k starts to fire (expression 4.12). The execution time of d is described by a delay governed by an exponential distribution which is characterized by the rate λ_k^s . We obtain the rates for kernels from the generic execution time distribution in original SADF specification in the same way as the detectors. Directly after the execution, the kernel k sends a synchronization signal to the related processes $k_{m+1}, \dots, k_n, d_{i+1}, \dots, d_j$ to modify their channel status and also the variables present the corresponding channel status in k are changed (expression 4.15).
2. The kernel k may also receive the synchronization signals from k_1, \dots, k_m , that request to modify the corresponding channel status before and after it has determined the scenario, in which it is going to operate. (expression 4.13, 4.16).

3. The kernel k receives the synchronization signals from d_1, \dots, d_i , and changes the status of corresponding control channel before and after d has determined the scenario, in which it is going to operate (expression 4.14, 4.17). These transition-meanings are identical with the definition in detectors.

Lemma 4.1 *The kernels belong to a subset of detectors. In other words, a kernel k is a detector d_k has following form:*

- d_k does not connect with any control channels through its output ports, and the input ports and control ports are identical with k ,
- the scenario set and subscenario set are identical to the scenario set of k : $\Omega_{d_k} = \mathcal{S}_{d_k} = \mathcal{S}_k$,
- the a set of $\{ (\mathbb{S}_{d_k}^s, \iota_{d_k}^s, \mathbb{P}_{d_k}^s, \Phi_{d_k}^s) \mid s \in \mathcal{S}_k \}$, where $\mathbb{S}_{d_k}^s = S, \iota_{d_k}^s = S, \mathbb{P}_{d_k}^s(S, S) = 1, \Phi_{d_k}^s(S) = s$,
- the rate function of d_k : $R_{d_k}^\omega$ is identical with the R_k^s function in k for all input ports and output ports belong to d_k ,
- the function $E_{d_k}^\omega$ and the probabilistic distribution $\mathbb{P}(E_{d_k}^\omega = e)$ on the samples in $E_{d_k}^\omega$ are also identical with the functions E_k^s and $\mathbb{P}(E_k^s = e)$ in k .

An illustration of the $(\mathbb{S}_{d_k}^s, \iota_{d_k}^s, \mathbb{P}_{d_k}^s, \Phi_{d_k}^s)$ of scenario $s \in \mathcal{S}_k$ is shown in following figure:



4.2 The IMC Semantics of an SADF Graph

To define the IMC semantics of a given SADF graph, our idea is to gradually add the processes into the current generated IMC which represents the partial SADF graph, until the whole SADF graph is obtained.

Different from compositional aggregation approach, the ordering to involve the components is irrelevant here if we generate the system directly. But the synchronization set of parallel composition should be different if the ordering changes.

Now we show how to get the IMC of partial SADF graph by using parallel composition. Assume we have two IMCs \mathcal{I}_{p_x} and \mathcal{I}_{p_y} to represent the processes p_x and p_y respectively, then the IMC for the partial SADF graph with only p_x and p_y can be defined as $\mathcal{I}_{p_x} \parallel_A \mathcal{I}_{p_y}$, where A is the action set that the both IMCs are forced to perform simultaneously (recall that this rule of parallel composition in IMC is similar to TCSP [21]). Recall that the action sets of p_x and p_y are denoted as Act_{p_x} and Act_{p_y} respectively. Then the synchronization sets A between process p_x and p_y is defined as $A = Act_{p_x} \cap Act_{p_y}$. Now we add the processes into the current generated IMC one by one until no process in the SADF graph is left, then we hide all actions used for synchronization. Assume we have now an IMC represent a partial SADF graph with a set of processes $\{p_1, \dots, p_N\}$, then we pick one process p_{N+1} which are not in the partial SADF and use parallel composition to involve this process into our partial SADF graph to form a new one.

Assume we have already construct an IMC represents a partial graph of SADF with the following form:

$$I_{p_1} \parallel_{A_1} I_{p_2} \parallel \dots \parallel_{A_{N-1}} I_{p_N}$$

Now we add the IMC of process p_{i+1} into our obtained partial SADF graph, then IMC need to parallel composed with the IMC of process p_{i+1} and with a new synchronization set A_i . The resulting IMC has the form:

$$(I_{p_1} \parallel_{A_1} I_{p_2} \parallel \dots \parallel_{A_{N-1}} I_{p_N}) \parallel_{A_N} I_{p_{N+1}}$$

where $A_N = Act_{p_{N+1}} \cap (Act_{p_1} \cup \dots \cup Act_{p_N})$.

Return to the initial channel status in SADF, we recall that the initial data channel status is determined by the function $\phi^* : \mathcal{B}_d \rightarrow \mathbb{N}$ and the initial control channel status by the function $\psi^* : \mathcal{B}_c \rightarrow \cup_{c \in \mathcal{B}_c} \Sigma_c^*$. We can simply translate the initial status into our definition by assigning values to the according channel variables in IML expressions. For instance, if we have a data channel DC_{p_x, p_y} and with the initial function that $\phi^*(DC_{p_x, p_y}) = n \in \mathbb{N}$, then in the definition of process p_y , the initial value of variable of DC_{p_x, p_y} is assigned to n . Similar is the case of initial control channel status. Then the IML expressions of a process p should have this form: $I_p = P(\phi^*(\dots), \dots, \phi^*(\dots), \psi^*(\dots), \dots, \psi^*(\dots))$.

Now we demonstrate the IMC semantics of SADF with an example shown in Figure 4.11.

Example 4.3 The SADF specification is shown in Figure 4.11. The SADF graph with two kernels A , B and one detector D is illustrated in the upper right. In the bottom right is the Markov chain associate with detector D for determination of subscenarios. On the right side is the function table of execution times and rates. And the initial channel status (initial data tokens and scenario strings) is represented as red points.

The semantics of the SADF specification of Example 4.3 is defined as follows:

$$\begin{aligned}
 S & := (\mathcal{I}_a \parallel_{G_1} \mathcal{I}_b \parallel_{G_2} \mathcal{I}_d) \setminus H \\
 \text{where } \mathcal{I}_a & = A (db_{b,a}^1=0, cb_{d,a}^1=[1]) \\
 \mathcal{I}_b & = B (db_{a,b}^1=2, cb_{d,b}^1=[1]) \\
 \mathcal{I}_d & = D (db_{b,d}^1=1) \\
 G_1 = G_2 = H & = \{\text{produce}_x^y \mid x \in \{a, b, d\}, y \in \{s, t\}\}.
 \end{aligned}$$

Process	λ_d^s	λ_d^t	ports	R_d^s	R_d^t	t_d^s	t_d^t
D	0.0490	0.1	$i_{b,d}$ $o_{d,b}$ $o_{d,a}$	1 1 1	1 1 1	S S S	T T T
Process	λ_a^s	λ_a^t	ports	R_a^s	R_a^t		
A	0.0347	-	$o_{a,b}$ $i_{b,a}$ $c_{d,a}$	2 1 1	0 0 1		
Process	λ_b^s	λ_b^t	ports	R_b^s	R_b^t		
B	0.0588	0.0313	$i_{a,b}$ $o_{b,a}$ $o_{b,d}$ $c_{d,b}$	2 1 1 1	0 0 1 1		

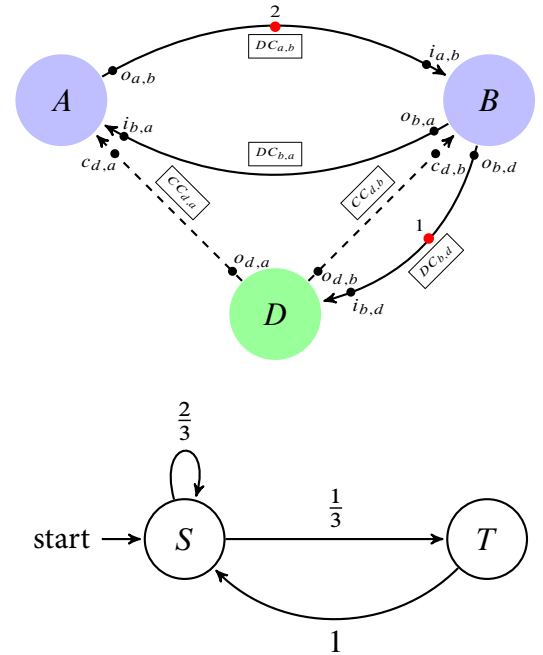


Figure 4.11: An SADF graph with its execution time distribution function and the rates of processes

The IMC of kernel A (for B analogously) is defined as an expression in IML as follows:

$$\begin{aligned}
 A (db_{b,a}^1, cb_{d,a}^1) & := \mathbf{0.0347} \cdot \overline{\text{produce}_a^s} \cdot A ((db_{b,a}^1-1), \text{tail}(cb_{d,a}^1)) : \\
 & \text{if } \text{first}(cb_{d,a}^1) = s \wedge db_{b,a}^1 \geq 1
 \end{aligned}$$

$$\begin{aligned}
& \overline{+\text{produce}_a^t . A (db_{b,a}^1, (\text{tail}(cb_{d,a}^1)))} : \\
& \quad \text{if } \text{first}(cb_{d,a}^1) = T \\
& +\text{produce}_d^s . A ((db_{b,a}^1, ([s:cb_{d,a}^1])) \\
& +\text{produce}_d^t . A ((db_{b,a}^1, ([t:cb_{d,a}^1])) \\
& +\text{produce}_b^s . A (((db_{b,a}^1+1), cb_{d,a}^1)) \\
& +\text{produce}_b^t . A ((db_{b,a}^1, cb_{d,a}^1))
\end{aligned}$$

The IMC of detector D is defined as an expression in IML

$$I_d = FD_d (db_{b,a}^1=1) \parallel_{A_d} SP_d^{\epsilon_D},$$

where $FD_d (db_{b,a}^1)$ is defined by the way that definition 4.3 describes, $A_d = \{\text{req}_{\epsilon_D}, s, t\}$, and the $SP_d^{\epsilon_D}$ is illustrated in Figure 4.12.

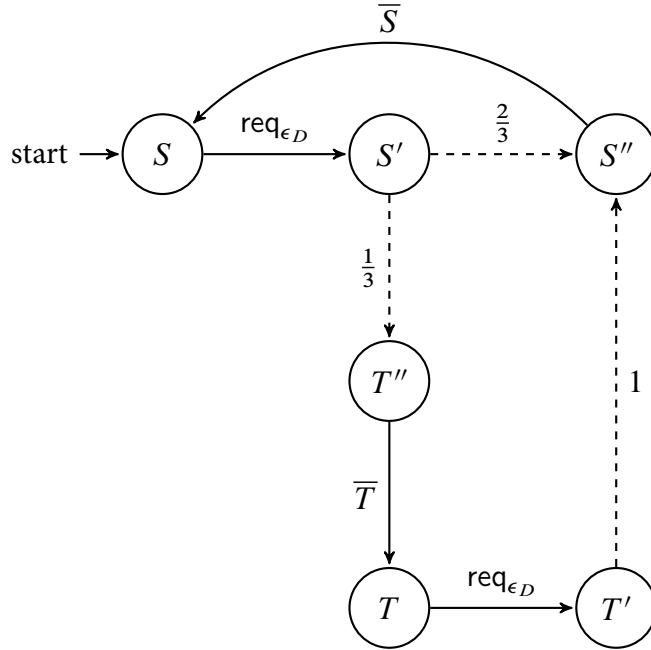


Figure 4.12: The IMC of $SP_d^{\epsilon_D}$

The generated IMC of this example is shown in Figure 5.11.

4.3 The Non-determinism and Properties in the resulting IMC

If we have constructed an IMC of an SADF graph according to the aforementioned approach, the IMC we obtain is obviously a *closed* IMC, i.e., all the actions in the resulting IMC are internal actions and these actions cannot be delayed, because the IMC has no actions to synchronize with other actions from external environment outside the SADF graph. On the other hand, in the resulting IMC, the transitions can be only probabilistic transitions, τ -transitions or Markovian transitions, since the action set A for hiding contains all actions in the resulting IMC. Furthermore, after generation of the whole IMC, applying stochastic branching bisimulation on the IMC should give us a minimized IMC.

As pointed out by [19], stochastic branching bisimulation facilitates to resolve the non-determinism in IMC, if any outcome of the non-deterministic choice leads to the same class of behaviors. In many cases of our SADF models, the minimized IMC by applying stochastic branching bisimulation has no τ -transition anymore, i.e. the non-determinism in the IMC after reducing is resolved. We discuss some situations about non-determinism in our IMC model in comparison with the non-determinism in the original TPS semantics of SADF. Note that after we have generated our IMC of an SADF graph, only three kinds of transitions are left: probabilistic transitions, internal τ -transitions, and Markovian transitions. Probabilistic transitions are labeled together with its process name, i.e. with the form of (d, p_{di}) (d is the process this probabilistic transition belongs to, i is the index of this probabilistic transition in the process d). Here we adopt this form of transitions to represent the *reactive* model of probabilistic processes [44]. According to the IML definition, the probabilistic transitions have this form: $Prob_d^{(\omega, x, y)}$, where d is the detector name, $Prob$ indicates this is a probabilistic transition (note that IML does not support any probabilistic transitions, we first treat the transitions as normal transitions labelled with “ $Prob$ ”), and (ω, x, y) is the index (identifier) of the probabilistic transition means this probabilistic transition belongs to detector d , and is the transition from state x to y in Markov chain associated with sub-scenario ω . Latter the real probabilistic information will be added by replacing this label with the corresponding probability.

Some observations. In following discussion, we first distinguish two assumptions defined in process algebra [36].

Maximal progress The internal interactive transitions take precedence over Markovian transitions.

Action urgency Markovian transitions can only happen when there is no other transitions than Markovian transitions can happen.

If we have only maximal progress assumption on the resulting IMC, there exists a non-deterministic choice in the IMC between probabilistic transitions and Markovian transitions. The maximal progress only resolve the conflict between τ -transitions (transitions labelled with produce) and Markovian transitions, i.e., the actions to modify the status of channels (also indicate the finalizing the firing of processes) is prioritized over advancing time. This assumption matches the original TPS semantics in [42] that requests the time transition can happen when no end action transitions are enabled. This non-determinism can be resolved by a scheduler, which has an impact on the metrics like throughput and buffer occupancy. Hence, in following discussion, we will concentrate on the IMC in which the action urgency assumption is applied, i.e., we assume the probabilistic transitions should happen instantaneously, because that making a decision of probabilistic choices costs no time. Since only three kinds of transitions are present in the resulting IMC, the action urgency makes sure that the Markovian transitions are postponed to some state where probabilistic transitions and internal interactive transitions can not happen any more.

Here, we call a process in SADF model is *ready-to-execute* when the process has determined the execution (sub)scenario and the data tokens in each corresponding channels are enough for firing. We assume the execution time (if not zero) of this process is modeled by a exponentially distributed delay characterized by the rate λ (or μ).

1. Ready-to-execute processes conflict (Figure 4.13(a)) Assume we have two or more processes are ready-to-execute at the same time and can perform their (Markovian) transitions. Recall that from 2.2.1, we have a resulting IMC shown in Figure 4.13(a). If the transitions were treated as normal transitions, after the processes are parallel composed (interleaving), the non-determinism would occur. In the case of Markovian transitions, the IMC we obtained has the same form like the normal transitions [27] due to the expansion law is hold for exponential distribution, but with a different meaning. In the example of Figure 4.13(a), after parallel composition of two ready-to-execute processes, first to take the Markovian transition λ or μ is not non-deterministic. From [20], we know the probability that the Markovian transition with rate λ happens earlier than the Markovian transition with rate μ within time d is $\frac{\lambda}{\lambda+\mu}$ and similarly the probability that the Markovian transition with rate μ happens earlier than the Markovian transition with rate λ within time d is $\frac{\mu}{\lambda+\mu}$. Note that, even two processes have the same rate in both Markovian transitions (i.e., they have the same mean time of firing), they will not finish their firings at the same time, since the firing time is modeled as a random variable which is exponentially distributed. In the TPS semantics of SADF, if two or

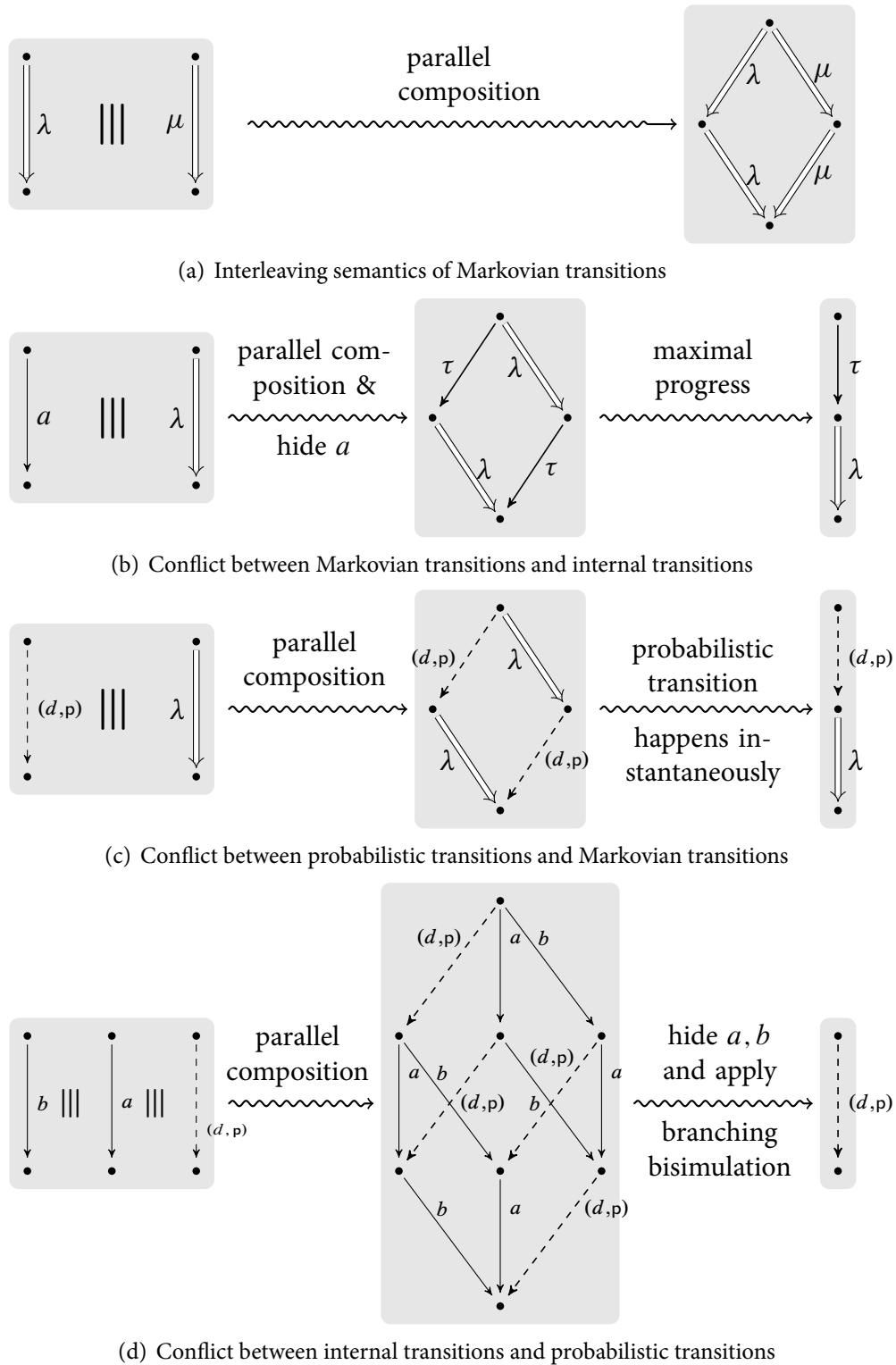


Figure 4.13: Non-determinism in IMC model of SADF specification

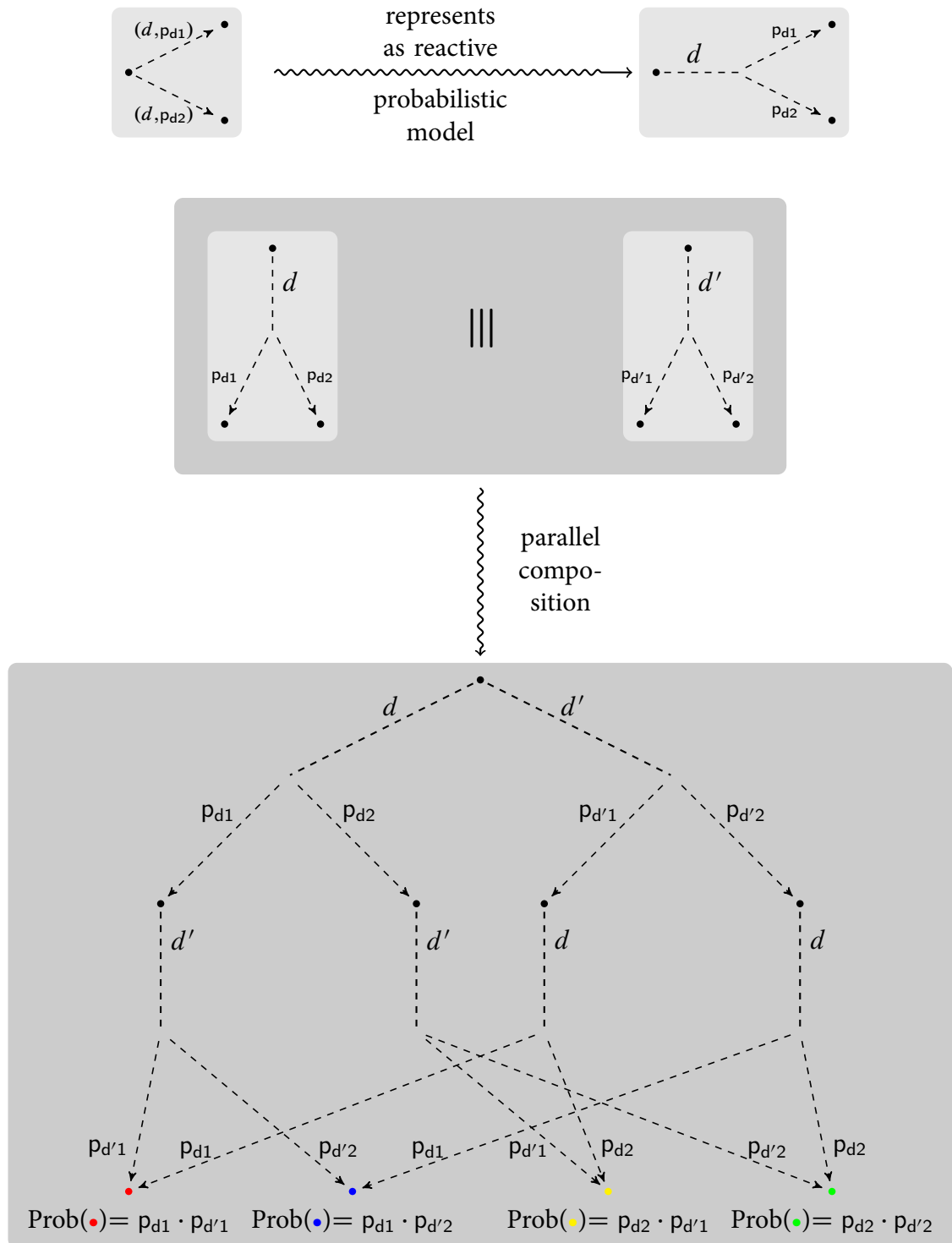


Figure 4.14: The conflict between probabilistic transitions

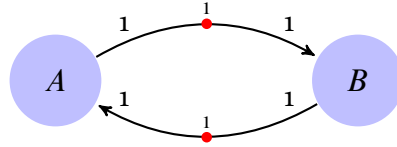
more processes have the same remaining firing time, non-determinism arises between the end action transitions of these processes. This kind of non-determinism does not exist in IMC semantics. On the other hand, in the IMC semantics of SADF, if two or more processes are ready-to-execute and the firings of all these processes cost no time, the way to resolve the non-determinism (i.e. the scheduler) to may affect certain performance metrics like max buffer occupancy. This problem is discussed further on in this section.

2. *The conflict between normal transitions and Markovian transitions* (Figure 4.13(b)) These conflicts are resolved by action urgency assumption. Since all the normal transitions in our IMC model are used for synchronization, these transitions are hidden (turn into τ -transitions) after we generate the whole system. By the assumption of *maximal progress*, the Markovian transitions are postponed if they are conflicting with the τ -transitions. The states and transitions reached by taking the Markovian transitions are removed.

3. *The conflict between probabilistic transitions and Markovian transitions* (Figure 4.13(c)) These conflicts are resolved by action urgency assumption. If one process can emanate a probabilistic transition and the other can first make a Markovian transition, they seem to conflict with making both transitions at the same time point after parallel composition. But the probabilistic transition should happen instantaneously, whereas the probability of Markovian transition to happen at once is zero. Consequently, the probabilistic transitions in our IMC should take precedence over Markovian transitions.

4. *The conflict between normal transitions and probabilistic transitions* (Figure 4.13(d)) In TPS and IMC semantics of SADF, the *end* actions and produce actions are used to modify the status of channels respectively. In IMC semantics, if all processes in all their (sub)scenarios need some time greater than zero to finish their firings, due to action urgency assumption, only one produce transition (after hiding as τ -transition) can happen between two successive Markovian transitions. The multiple produce transitions can only happen between two successive Markovian transitions, if the produce transition following the Markovian transition enables some processes cost no time to finish their firings in their (sub)scenarios. This situation is illustrated by the Figure 4.13(d), and in this figure a non-deterministic choice between two normal transitions a, b (in the IMC semantics of SADF can be seen as two different produce actions) and a probabilistic transition (d, p) exists. It is also analogous if we have multiple probabilistic transitions. If we hide a and b actions, after (stochastic) branching bisimulation, only the probabilistic transition are left. The non-determinism is resolved by (stochastic) branching bisimulation. One issue reveals here is the IMC is not state-oriented but behavior-oriented [24] and the states in IMC is not revelent. This property

of IMC has an impact on some performance metrics in SADF, e.g., if we want to know the max buffer occupancy in SADF. Recall that the status (the current tokens) of a channel is modeled in IMC as variables integrated in the processes. Let us consider a simple SADF shown in following figure:



In this SADF graph, if both firings of kernel A and B in their scenarios cost no time, there will be a non-deterministic choice between produce_A and produce_B . If produce_A happens first, the maximal buffer occupancy in $DC_{A,B}$ is 2 (i.e., the data value of $DC_{A,B}$ integrated in the state is 2) and the maximal buffer occupancy is 1. If produce_B happens first, then $DC_{B,A}$ has a maximal buffer occupancy of 2 and $DC_{A,B}$ has a maximal buffer occupancy of 1. Therefore, the way to resolve the non-determinism has an impact on the maximal buffer occupancy distribution. In our case, this differences of states are not distinguished. Furthermore, the channel status integrated in the process as variable is only visible, when we show the value of the variable integrated into an additional action. To investigate such properties in SADF graph, we need other techniques.

5. The conflict between probabilistic transitions (Figure 4.14) The complex situation occurs if there are more than one process can make probabilistic choice at the same point in the SADF specification. In our model, if we parallel compose with such processes, we should have a reactive model of probabilistic process [44]. From the discussion above, we know that the conflict between normal transitions and probabilistic transitions is solved by branching bisimulation and only probabilistic transitions are left. So, we only discuss the conflicts between probabilistic transitions. Fortunately, the order of choosing which process happens first does not influence probability to reach the states (cf. Figure 4.14). First, we know the Markovian transitions are postponed to some states where no τ -transitions and probabilistic transitions can happen any more (due to action urgency), and the transitions between two successive Markovian transition cost no time (contains no information of time). In Figure 4.14, we assume the points colored with red, blue, yellow and green are the states where no probabilistic and normal transitions are possible, and no τ -transitions are left due to branching bisimulation. From this figure, we can see that the probabilities to reach these states does not depend on the choice of which detector d or d' goes first, we reach the same state with the same probability. This phenomenon also matches the *diamond property* in TPS semantics of SADF introduced in Section 3.3.2. Based on this property, we can set the priority on processes intentionally to resolve the non-determinism, for instance, in Figure 4.14, we may set either d goes first or d' goes first after we have generate the IMC model.

To sum up, from TPS semantics of SADF, many long-run average metrics of an SADF are not affected by the scheduler used for resolving non-determinism. This is also valid in our IMC semantics of SADF. First the action urgency assumption postpones the Markovian transitions to the states, where no synchronization and probabilistic choice action is possible. This means that between two successive Markovian transition, no temporal information is contained, and the probabilistic transitions and synchronization transitions has only divided the states where only the next Markovian transitions can happen, into certain equivalent classes. Then we have shown such classes are not affected by synchronization actions but by probabilistic choices (the conflict between normal transition and probabilistic transitions). After that, the diamond property assures the scheduler to resolve the non-determinism will not affect on the reachability of each equivalent class and leaves the temporal properties of IMC untouched. This observation matches the long-run equivalence in [41].

4.3.1 Infinite System caused by Non-determinism

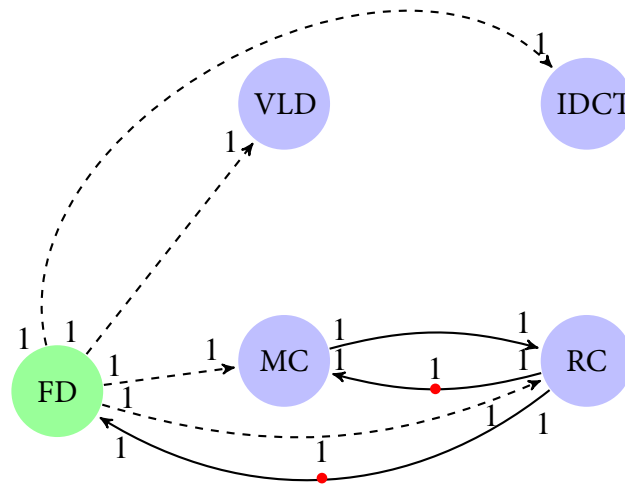


Figure 4.15: Non-determinism may cause unbounded system in our IMC model

In SADF graph, the circular dependency of components makes sure that the resulting semantic model is bounded. But in our IMC model, if the original SADF is *untimed*, an infinite system may be caused by the parallel composition of the components that are not strongly connected, which violates the circular dependency. Figure 4.15 illustrates such an SADF graph with this issue. This SADF graph is an example of the MPEG-4

decoder but with only one scenario P_0 . The channels with rate 0 at their input ports and output ports are simply removed here, and the firings of all processes cost zero (no) time. The zero execution time of processes leads to a non-deterministic choice if at the same time point two or more processes are ready-to-execute. Note that, if a process cost some time to finish its firing, there must be a Markovian transition (with rate λ) between the transitions used for synchronization with this form: $(\dots \xrightarrow{\alpha} \xrightarrow{\lambda} \xrightarrow{\beta} \dots)$. Since the firing of processes cost no time, there will no such transitions (no Markovian transition can express an immediately firing), to add a special interactive transition (e.g., like a transition $\xrightarrow{\text{execution}_p}$) to indicate a process's zero-time execution is also not an option here, since this complicates the system and such transitions will be hidden finally. Now, we simulate to generate the IMC of this SADF model. Since there is no scenario tokens in any control channels in the SADF, the only possibility is the FD starts to execute (one data token is available in $DC_{rc,fd}$) and then it consumes the data token and produces one scenario token P_0 to each control channel which are connected with its output ports (i.e. $CC_{fd,vld}$, $CC_{fd,idct}$, $CC_{fd,mc}$, $CC_{fd,rc}$). Now, the kernels VLD , $IDCT$, MC are all ready-to-execute, since all their execution time is zero. A non-deterministic choice exists now among these three processes.

Assume we let MC to fire first rather than VLD and $IDCT$, and direct after MC 's firing, MC produces one data token into the channel $DC_{mc,rc}$ and consumes the data token in channel $DC_{rc,mc}$ (the production and consumption of data tokens cost also no time), and then VLD , $IDCT$, RC are ready-to-execute. Again, all the processes VLD , $IDCT$, RC cost zero time to execute, and a non-deterministic choice appears here. Now, we unfairly let RC fire before VLD , $IDCT$ again, then after RC 's firing, it produces directly one data token to $DC_{rc,mc}$ and consumes the data token in $DC_{mc,rc}$. Now VLD , $IDCT$, FD are all ready-to-execute, and again we let FD to fire before VLD , $IDCT$ and the competition between the two processes VLD , $IDCT$ and the three processes FD , MC , RC repeats. As a result, if we always suppress the firing of VLD and $IDCT$, then the three processes FD , MC and RC form an execution circle. Now we observe at the status of control channels $CC_{fd,vld}$ and $CC_{fd,idct}$ in this unfair situation: after every execution loop of the three processes FD , MC and RC , the detector FD will produce one scenario token P_0 to all control channels including $CC_{fd,vld}$ and $CC_{fd,idct}$. However, VLD and $IDCT$ have never the chance to consume these control tokens due to the suppression. Therefore, the channels $CC_{fd,vld}$ and $CC_{fd,idct}$ collect the control tokens produced by FD infinitely. To solve this problem, we can manually set that the kernel RC as the last process of each execution "loop" among all these processes, i.e., we manually let the VLD , $IDCT$ to synchronize RC with a additional execution end action indicating RC 's finish of firing in scenario P_0 . This assumption guarantees the processes VLD , $IDCT$, MC and RC consume the scenario tokens from their control channels at the same pace, and the system becomes never infinite.

Chapter 5

State Space Generation, Analysis of IMC and Case Studies

In Chapter 4, we presented a framework to define the IMC semantics using IML expressions for every component in an SADF specification. As we can see, the IML expressions only define the possible behaviors of components, but for the generation of the IMC model, we exploit the CADP (Construction and Analysis of Distributed Process) toolset developed by the VASY team at INRIA Rhone-Alpes. The CADP toolset allows us not only to generate the IMC model by giving the IML expressions, but also provides tools to analyze the generated model qualitatively and quantitatively [14]. The strength of IMC is the integration of stochastic behavior into the framework of process algebra, which allows to check both functional properties and temporal properties. The CADP toolbox provides a mature solution covering the three phases of model checking: state space generation, state space minimization, functional and numerical analysis.

5.1 Overview of CADP toolbox

Figure 5.1 shows us the mainly used tools in CADP in our case.

System behavior description. Our first step is to translate the IML expressions into the system specification language LOTOS in CADP. From Chapter 2, we know that in the IMC, the stochastic and interactive transitions are strictly separated. On the one hand, this separation facilitates a clear semantical model [19], on the other hand, this makes

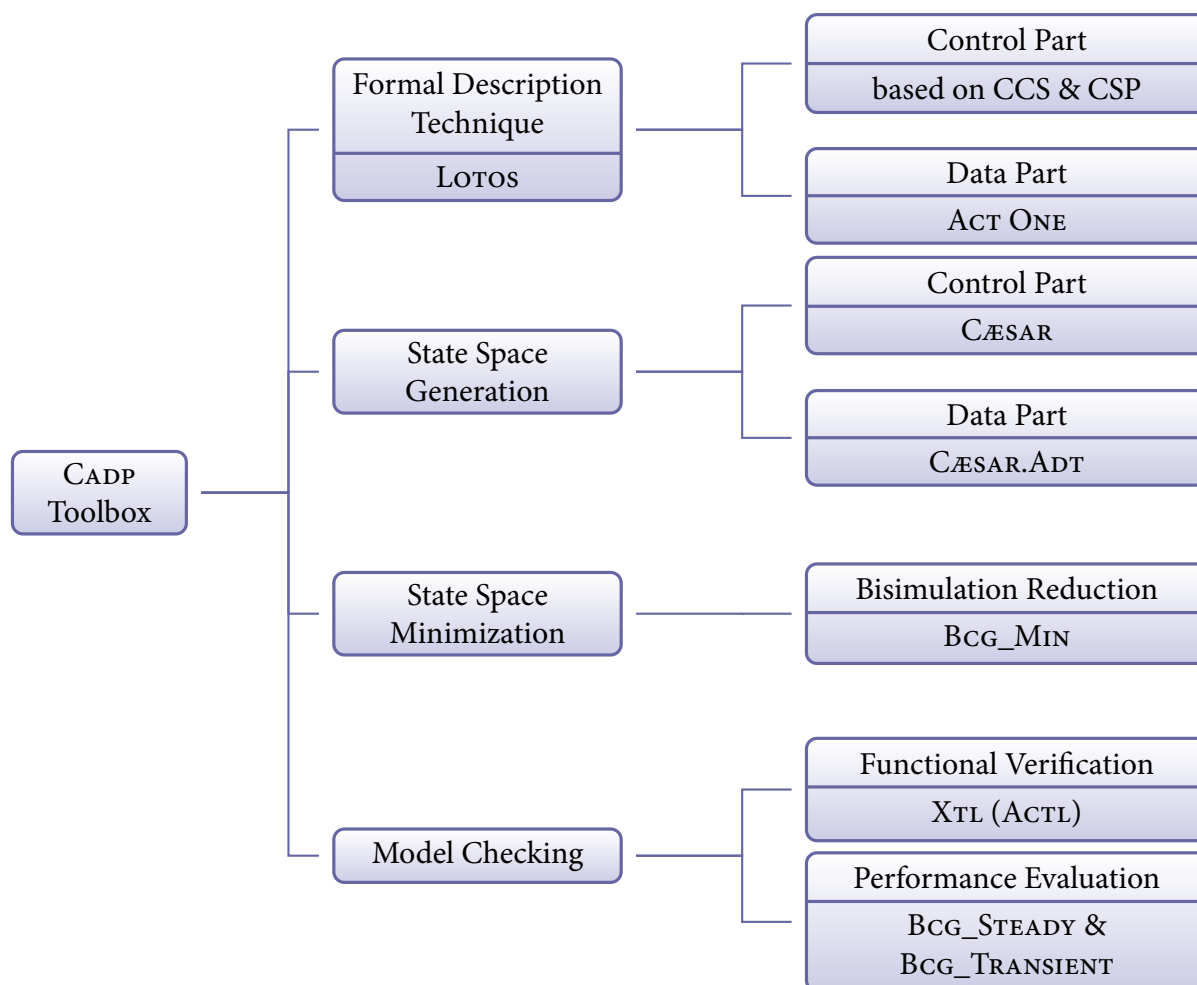


Figure 5.1: The mainly used tools of CADP in our case

IMC also compatible with most existing process algebras without additional syntactic and semantic modifications [14]. Therefore, CADP did not develop new tools for IMC but only extended its functionality with the support of Markovian transitions. CADP adopts the process algebra LOTOS for process¹ specification. LOTOS is an ISO standardized Formal Description Technique and widely used for the specification of large concurrent systems. In LOTOS, two sub-languages are used for system specification: the abstract data type language ACT ONE for the data algebraic part, and a process algebra based on combination of CCS and CSP for the control part. The translation from IML to LOTOS is described in Section 5.2.

¹The definitions of process are used both in SADF specification and in process algebras and have different meanings. To distinguish the process definition, the process defined in process algebra is written as “process”.

State space generation. Our next step is to generate the state space of our IMC model. This is done by the tools `CÆSAR` and `CÆSAR.ADT`. The tools `CÆSAR.ADT` and `CÆASR` compile the data part and the control part of LOTOS to C code respectively. `CÆSAR` is responsible for generating the LTS according to our given LOTOS specification. The obtained LTS is encoded in the BCG (sometimes also AUT) format and then can be used for further analysis or minimization. If our model is an IMC rather than an LTS, we need to turn the LTS into the corresponding IMC. In the generated LTS, the Markovian transitions or the probabilistic transitions are only treated as normal labelled transitions without any synchronization capabilities. To turn the LTS into IMC, we employ the `BCG_LABEL` tool of CADP, which can replace these normal labelled transitions with Markovian transitions or probabilistic transitions and adds the rate/probabilistic information into the transitions.

State space minimization. For state space minimization, we employ the `BCG_MIN` tool. In Chapter 2, we introduced several equivalence notions on IMCs, e.g., strong bisimulation, weak bisimulation and stochastic branching bisimulation. The `BCG_MIN` tool in CADP supports these minimization techniques on the generated state space of IMC. The tool `BCG_MIN` accepts the following three kinds of (extended) LTS in BCG graph format and does the minimization based on the modulo bisimulation equivalence.

- *normal LTS*;
- *probabilistic LTS*, whose transitions may have the following three forms:

- \xrightarrow{a} : a normal transition a ,
- $\xrightarrow[p]{- - - -}$: a *probabilistic* transition with a probability $p \in (0, 1]$,
- $\xrightarrow{(a, p)}{- - - -}$: a transition labelled by the couple (a, p) .

The last form can represent the different models, i.e. reactive, generative and stratified models of probabilistic processes introduced in [44]. We will see later we adopt reactive model of probabilistic transitions in our IMC model;

- *stochastic LTS*, whose transitions are similar as in the probabilistic case, with the probability p is replaced by a rate λ (a positive real number).

Functional verification. For functional verification, we use the ACTL Logic [35] implemented by XTL Language in CADP toolset. ACTL is an action-based temporal logic, and especially appropriate for the behavior-oriented specifications.

Performance evaluation. The two tools BCG_STEADY and BCG_TRANSIENT can do performance analysis on (extended) CTMCs or DTMCs. For each state of a given CTMC or DTMC, the tool BCG_STEADY computes the probability to be in that state on the long run based on a Gauss-Seidel algorithm. The tool BCG_TRANSIENT computes the transition probability distribution of a given (extended) CTMC or DTMC at user-given time instances based on the uniformization algorithm and the Fox-Glynn method [14]. Beyond this, both tools can also compute the *transient* and *steady* transition throughputs. These (labelled) transition throughputs are computed as follows: for each occurring of a transition labelled with "x", all Markov transitions "*x; rate %f*" are collected in a weighted sum, where the respective *%f* value is weighted with the transient (or steady) probability of the tangible source state of this transition.

5.2 Generation of IMC models

To generate the IMC models, we translate the IML expression into two parts, the data part and the control part. In the data part, the data types used in the control part are (pre)defined. In the control part, the behavior of system are formally defined by using LOTOS like other process algebras.

5.2.1 Definition of data types

To generate the IMC of an SADF graph from the definition in Chapter 4, we need to translate our IML expressions into LOTOS. The first step is to define our data types by using the formal algebraic specification language ACT ONE. We define here two data types, i.e., *scenario token* and *control buffer*. Since the data channel in SADF is treated as natural number (a counter), which is predefined in CADP, we just use it by a declaration. Since the basic data type *boolean* is also needed in our definition, we include again the boolean data type in front of our definition.

The data type SCENARIO_TOKEN consists of the definitions of sorts, operations (opns), and equations (eqns). A sort is a set of values. In our case, we have only one sort called the Scenario-Token. Then the values (e.g. S, T, and default) of sort Scenario-Token are defined as constructors (basic elements) in the operation part. Further, a function called eqt is also declared in the operation part to compare the values of (sub)scenario tokens. This function is defined in the eqns section, which takes two scenario tokens as arguments and returns "true" if the two scenario tokens are identical, and otherwise it

returns “false”. A fragment of the ACT ONE definition of SCENARIO_TOKEN is shown in Figure 5.2. Now we define the data type of control buffer. In Chapter 4, we introduced

```

type SCENARIO_TOKEN is Boolean
  sorts Scenario-Token
  opns
    S      (*! constructor *),
    T      (*! constructor *),
    default (*! constructor *) :-> Scenario-Token
  _ eqt _ : Scenario-Token, Scenario-Token -> Bool
  eqns
    forall x:Scenario-Token, y:Scenario-Token
      ofsort Bool
        x eqt x = true;
        x eqt y = false;
endtype

```

Figure 5.2: A definition fragment of data type SCENARIO_TOKEN

some functions in the control buffer informally, and now we define the functions by using ACT ONE. The first line says that the functions in data type CONTROL_BUFFER can return values in data types boolean, SCENARIO_TOKEN, and natural numbers. The only sort in the control buffer is called Control_Buffer (lists of sort Scenario_Tokens) and consists of two constructors: Nil and Produce. Nil is the bottom symbol of a control buffer and represents an empty control buffer. The elementary constructor Produce is used to add values of scenario tokens to the front of the control buffer, and is defined as a function, which takes a value of data type Scenario-Token and a list of Scenario_Tokens as argument and returns the new list of Scenario_Tokens. Also a function Batch_Prod is defined, this function is used, if we need to add several scenario tokens of the same value into the list. This function takes three arguments: the value of Scenario-Token, a natural number (how many tokens to be added), and the list of Scenario_Tokens, then it returns the new resulting list of Scenario_Tokens. The Consume function is used to remove the topmost scenario token of the list.

5.2.2 Definition of Control Part

The process algebra LOTOS is standardized by Iso and based on CCS and CSP. The semantics of parallel composition in LOTOS has the same form in CSP, which forces the

```

type CONTROL_BUFFER is Boolean, SCENARIO_TOKEN, Natural
  sorts
    Control_Buffer
  opns
    Nil      (*! constructor *) : -> Control_Buffer
    Produce  (*! constructor *) :
      Scenario-Token, Control_Buffer -> Control_Buffer
    Batch_Prod : Scenario-Token, Nat, Control_Buffer
      -> Control_Buffer

    Consume   : Control_Buffer -> Control_Buffer
    Empty     : Control_Buffer -> Bool
    First     : Control_Buffer -> Scenario-Token
    Last      : Control_Buffer -> Scenario-Token
  eqns
    forall ST:Scenario-Token, CB:Control_Buffer, N:Nat
      ofsort Control_Buffer
        (* Consume (Nil) is not defined *)
        Consume (Produce ( ST , Nil ) ) = Nil;
        not (Empty (CB)) =>
          Consume(Produce (ST, CB)) = Produce(ST, Consume (CB));
        Batch_Prod(ST,0,CB)=CB;
        Batch_Prod(ST,N,CB)=Batch_Prod(ST,(N-1),Produce(ST,CB));
      ofsort Bool
        Empty (Nil) = true;
        Empty (Produce(ST,CB)) = false;
      ofsort Scenario-Token
        (* First (Nil) is not defined *)
        First (Produce (ST, Nil)) = ST;
        not (Empty (CB)) =>
          First (Produce (ST, CB)) = First (CB);
        (* Last (Nil) is not defined *)
        Last (Produce (ST,CB)) = ST;
    endtype

```

Figure 5.3: A definition fragment of control buffer

actions to be synchronized if they are in the synchronization set. In IMC, the actions in synchronization set between two behavior expressions are also forced to synchronize, so we can translate our parallel composition operation in IMCs directly into LOTOS. We first introduce some basic terminologies in LOTOS comparing with IMC and then the basic operators in LOTOS and their semantics.

Concept and Terminologies in LOTOS

Process

The processes are the elementary components of a system. In our case, the kernels and detectors are defined as processes in LOTOS (cf. Figure 5.4). Also local process definitions in a process are possible, for example, the process detector consists of scenario-specification module and functional module, and both modules are defined as processes. The process describes a certain functionality of a component. If we can hide all the state transitions and interactions happened inside the process according to a certain abstraction level, then the process can be seen as a black box leaving specific interfaces to communicate/synchronize with other processes.

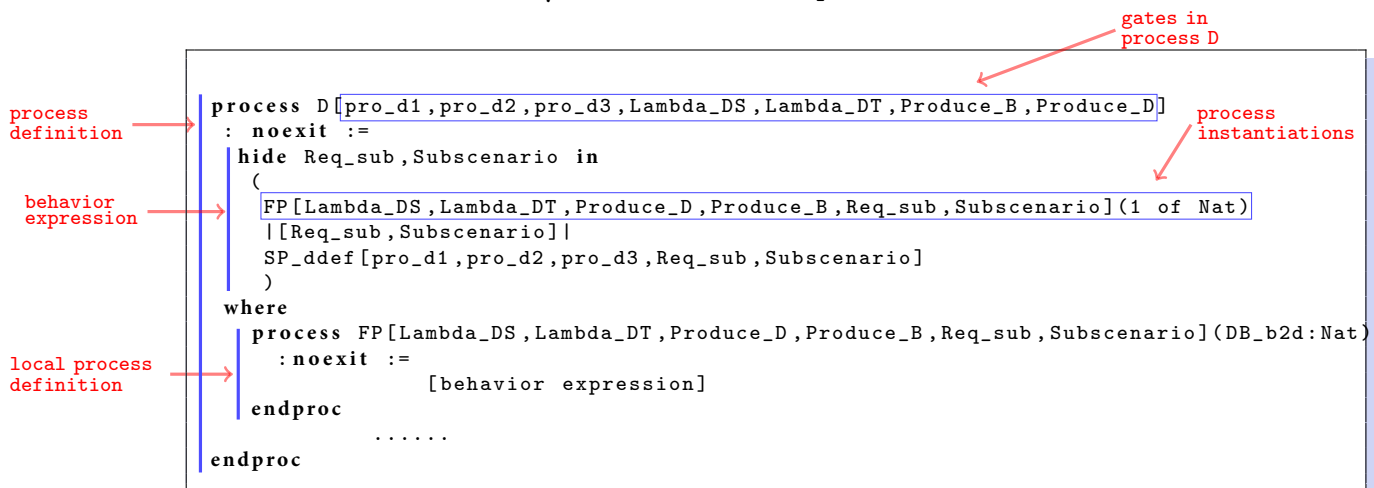


Figure 5.4: An excerpt of the definition of detector D

Events

The synchronization between processes is called an event. Therefore, events provide a process the opportunity to communicate with its outside environment and to participate a certain function task. Events between processes allow three kinds of synchronization:

- *Pure synchronization.* The processes to be synchronized are forced to make a transition together.
- *Synchronization with value passing.* If one process finishes its task with certain parameter, this process can synchronize with another process and pass the parameter to the other process. In our case, the detector may finish its firing and pass the determined scenario to the processes it controls.

- *Synchronization with value agreement.* Two or more processes can agree with a set of values during synchronization.

Gate

An event occurs at gate, and if the synchronization has no data exchange, the event name is the same as the gate name. So, in basic LOTOS, an observable action coincides with a gate name. In full LOTOS, the gate can contain values of different data types. For instance, in our example, the process of detector D may provide a gate named Produce_D together with a (sub)scenario token and synchronize with other processes based on the value of (sub)scenario token, e.g. Produce_D !S of Scenario-Token. The expression in IML produce_D^S is equivalent to the gate Produce_D !S of Scenario-Token in LOTOS. Another form of gate is Produce_D ?ST : Scenario-Token. This kind of gate can be used, for instance, if another process is waiting for detector D to synchronize with a set of values and then determines its later behavior depending on the scenario values. The ST is a variable here and then is assigned to the value of the scenario after synchronization, and the ? means the variable is a value among the sort (set) Scenario-Token. It may also happen that the processes at the both sides have this form of gate, then a gate with a certain set of value is allowed to synchronize (synchronization with value agreement).

In the behavior expressions of processes, several possible operators are used. Here we introduce some operators used in our case.

Sequentiality

The operator “ ; ” in LOTOS is called the *action prefix*. It is equivalent to the “ . ” operator of IML. It is used to prefix an behavior expression with an gate (action), e.g., an expression (action; B) in LOTOS means the process executes the action and then behaves like B . Several actions may associated sequential by using action prefix with this form $\text{action}_1; \dots; \text{action}_n; B$.

Choice

The choice operator “[] ” has the same semantics as the choice operator “ + ” in IML. This operator allows process to behave alternatively. An behavior expression $B_1 [] B_2$ means the system can behave either like a B_1 or like a B_2 , but as soon as it enters the behavior expression B_1 or B_2 , the other one is discarded automatically.

Parallelism

For describing the parallelism of system, LOTOS provide three operators, i.e., interleaving, partial synchronization, full synchronization.

Interleaving. The interleaving parallel compositional operator “ \parallel ” is used to model concurrency and allows processes to be parallel composed without any synchronization. This operator is equivalent to “ \parallel_{\emptyset} ”.

Partial synchronization. The partial synchronization parallel composition operator is denoted as “ $| [< gates >] |$ ”. If the events occurring at the gates are listed in the set “ $< gate >$ ”, a synchronization behavior will happen, otherwise the events will interleave. Note that the events (actions) in the synchronization set of IMC and LOTOS are “forced” to be synchronized.

Full synchronization. The full synchronization parallel composition operator is denoted in LOTOS as “ \parallel ”. The expression $B_1 \parallel B_2$ means the gates occur in either behavior expressions are forced to synchronize.

Termination

LOTOS provides two expressions to represent the behavior of termination. They are “stop” and “exit” with following semantics:

1. stop represents an unsuccessful termination, i.e. inaction or a deadlock.
2. exit represents a successful termination of the system.

From IML to LOTOS

Figure 5.5 shows an excerpt of the LOTOS definition of the example in Figure 4.11. Compared to our SADF semantics of this example in IML expressions, the IML expressions intuitively translates into LOTOS.

5.2.3 State space Generation

To generate the IMC model according to a given system behavior description, two approaches are available. The first approach is called “state space exploration”, which generates the state space straightforwardly. Most of time, this “brute force” technique suffers from the well-known *state explosion* problem. Nowadays, more and more researchers interest a new approach called “compositional minimization” or “compositional aggregation”. The essential idea behind this approach is to construct the state space of the system gradually. At the beginning, the system is decomposed into subsystems/components, then we choose two components doing parallel composition on them to get a partial system. This partial system is not immediately used to parallel compose with

```

specification ABD[pro_1,pro_2,pro_3,Lambda_DS,Lambda_DT,Lambda_AS,Lambda_BS,Lambda_BT]: noexit

[data part]

behavior
  (
    D[pro_D1,pro_D2,pro_D3,Lambda_DS,Lambda_DT,Produce_D,Produce_B]
    | [Produce_D,Produce_B] |
    (
      A[Lambda_AS,Produce_A,Produce_B,Produce_D](0 of Nat,Nil of Control_Buffer)
      | [Produce_A,Produce_B,Produce_D] |
      B[Lambda_BS,Lambda_BT,Produce_B,Produce_A,Produce_D](2 of Nat,Nil of Control_Buffer)
    )
  )
where
[process definitions]
endspec

```

Figure 5.5: An excerpt of the example in Figure 4.11

other components, but aggregated by applying certain *reduction* technique. These reduction techniques let this partial system to modulo an appropriate equivalence or pre-order relation to get a smaller but “equivalent” system for further parallel composing. This obtained system should prevent the properties to be verified on the whole system as the original one. The remaining components are involved in the intermediate partial system one by one until we get the whole system with using the reduction technique each time on the intermediate partial system. The usually used equivalence relation are introduced in Chapter 2, e.g., strong bisimulation, weak bisimulation and branching bisimulation. In many case studies [20, 8], this approach has proven its strength. Also the ordering to involve these components have a significant impact of the size of partial system and this problem is discussed in [15].

The compositional aggregation approach is not always productive. In some cases, we may successfully generate the whole state space by using the state space exploration approach, but the parallel composition with two subsystems (components) leads to state explosion [17]. Some solutions are given to tackle this issue, the mostly used approach is to take the *interface (context) constraints* into account when we are doing parallel composition. Generally speaking, the interface constraints are the environment (context) of the process in the system (usually its neighbor processes). In the classic way of doing parallel composition, the intermediate partial system does not know its context in the whole system, but there must exist some restriction from the other processes outside it. The obvious restriction is the synchronization actions (gates) between the processes in the partial system and their neighbor processes. Since the synchronization has a forced manner, some “free” actions appear in the partial system is actually restricted by other processes outside. In some cases, we can even eliminate states and

transition that are not reachable before we get the whole system. Two methods are proposed in [12] and [30]. In [29] and [30], the solution is called *interface*, which expresses the environment by a sub-expression, i.e., an LTS representing a set of “authorized” execution sequence that can be performed. This method is also implemented by CADP toolset. In CADP, a tool called PROJECTOR is used to construct the intermediate partial system under interface constraints. The interfaces can be generated automatically by the tool or given by the user self. Also a new parallel composition operator called *semi-composition* (we denote this operator as \boxtimes) is defined in the PROJECTOR tool.

Semi-Composition

Definition 5.1 (Semi-Composition) Let $\mathcal{I}_1 = (S_1, Act_1, \longrightarrow_1, s_{0_1})$ and $\mathcal{I}_2 = (S_2, Act_2, \longrightarrow_2, s_{0_2})$ be two LTSs, A is a label set with $A \in \mathcal{A}$ (\mathcal{A} is the set of observable actions) and the LTS resulting from $\mathcal{I}_1 \parallel_A \mathcal{I}_2 = (S', Act', \longrightarrow', s'_0)$. Then we define the LTS denoted as $\mathcal{I}_1 \boxtimes_A \mathcal{I}_2 = (S, Act_1, \longrightarrow, s_{0_1})$ resulting from the semi-composition of \mathcal{I}_1 by \mathcal{I}_2 as follows:

$$S = \{ p \mid (p, q) \in S' \}$$

$$\longrightarrow = \longrightarrow_1 \cap \{ (p_1, a, p_2) \mid (p_1, q_1) \xrightarrow{a}' (p_2, q_2) \}$$

We call the set A the synchronization set and the pair (A, S_2) the interface, and if an action $a \in A_1 \cap A$, we say the action a is controlled by the interface (A, S_2) .

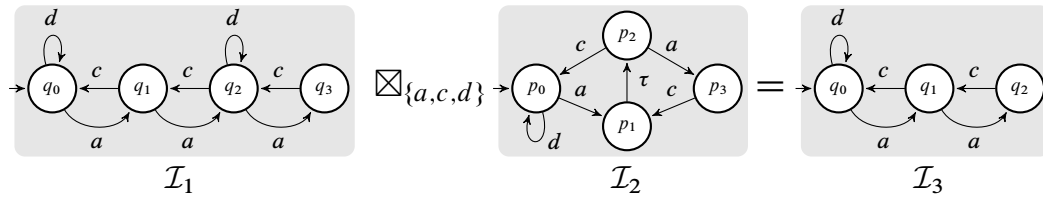


Figure 5.6: A example of semi-composition: $\mathcal{I}_3 = \mathcal{I}_1 \boxtimes_{\{a,c,d\}} \mathcal{I}_2$

Obviously, the obtained LTS $\mathcal{I}_1 \boxtimes_A \mathcal{I}_2$ is a sub-LTS of \mathcal{I}_1 , and means also that applying semi-composition of \mathcal{I}_1 by \mathcal{I}_2 does increase the number of states and transitions of \mathcal{I}_1 . If we use this “context-aware” LTS $\mathcal{I}_1 \boxtimes_A \mathcal{I}_2$ rather than \mathcal{I}_1 as parallel composition component for further compositional aggregation, it can contribute for generating the whole system with a tractable state space size if the redundant states and transition are removed as early as possible. In [30], three significant properties of semi-composition are stated:

1. Semi-composition can reduce the state space, since the resulting LTS $\mathcal{I}_1 \boxtimes_A \mathcal{I}_2$ is a sub-LTS of \mathcal{I}_1 . Even in the worst case we have $\mathcal{I}_1 \boxtimes_A \mathcal{I}_2 = \mathcal{I}_1$.
2. $(\mathcal{I}_1 \boxtimes_A \mathcal{I}_2) \parallel_A \mathcal{I}_2 = \mathcal{I}_1 \parallel_A \mathcal{I}_2$. This property of semi-composition fulfills the requirement of the essential idea behind using context constraints: the semi-composition preserves not only the temporal property of the system $\mathcal{I}_1 \boxtimes_A \mathcal{I}_2$, but also cut-off the unreachable states and transition of \mathcal{I}_1 by given a context LTS \mathcal{I}_2 .
3. $\mathcal{I}_1 \boxtimes_A \mathcal{I}_2 = \mathcal{I}_1 \boxtimes_A \mathcal{I}'_2$, if $\mathcal{L}(\text{hide } (\mathcal{A} \setminus A) \text{ in } \mathcal{I}_2) = \mathcal{L}(\text{hide } (\mathcal{A} \setminus A) \text{ in } \mathcal{I}'_2)$, where $\mathcal{L}(\mathcal{I})$ denote the observable traces of LTS \mathcal{I} . This property makes sure that we can first reduce the state space of the interface of \mathcal{I}_1 , i.e. \mathcal{I}_2 , before we do semi-composition. We can first find a reduced LTS, i.e. \mathcal{I}'_2 , which is observable trace equivalent to the original \mathcal{I}_2 , and semi-compose \mathcal{I}_1 with \mathcal{I}'_2 .

One limitation of this approach so far is the context constraints are limited to the “direct” relation between LTS \mathcal{I}_1 and \mathcal{I}_2 . In other words, sometimes the context of interface \mathcal{I}_2 is too weak to restrict the behavior of \mathcal{I}_1 . For instance, for a parallel composed LTS $(\mathcal{I}_1 \parallel_A \mathcal{I}_2) \parallel_B \mathcal{I}_3$, the LTS \mathcal{I}_2 does not constrain the behavior of \mathcal{I}_1 , but \mathcal{I}_3 does. To handel this problem, in [29] Krimm & Mounier presented the following semi-composition laws:

$$\begin{aligned}
\mathcal{I}_1 \parallel_A \mathcal{I}_2 &= (\mathcal{I}_1 \boxtimes_A \mathcal{I}_2) \parallel_A \mathcal{I}_2 \\
(\mathcal{I}_1 \parallel_{A_1} \mathcal{I}_2) \parallel_{A_2} \mathcal{I}_3 &= ((\mathcal{I}_1 \boxtimes_B \mathcal{I}_3) \parallel_{A_1} \mathcal{I}_2) \parallel_{A_2} \mathcal{I}_3 \\
&\quad \text{where } B = A_2 \cap (A_1 \cup (\text{Act}(\mathcal{I}_1) \setminus \text{Act}(\mathcal{I}_2))) \\
(\text{hide } A_1 \text{ in } \mathcal{I}_1) \parallel_{A_2} \mathcal{I}_2 &= (\text{hide } A_1 \text{ in } (\mathcal{I}_1 \boxtimes_{A_2 \setminus A_1} \mathcal{I}_2)) \parallel_{A_2} \mathcal{I}_2
\end{aligned}$$

Although, the three laws improve the performance of semi-composition in practice, the approach of Krimm & Mounier has still its weaknesses. These issues are stated in [29] and a refined approach to generate interface constraints is represented.

Issue 1. Krimm & Mounier’s method does not allow to take more than one interface into account simultaneously. For instance, if we have an expression of LTS like $\mathcal{I}_1 \parallel_{A_1} (\mathcal{I}_2 \parallel_{A_2} \mathcal{I}_3)$, it is possible that both LTSs generated by $\mathcal{I}_3 \boxtimes_{A_1 \cap \text{Act}_{\mathcal{I}_3}} \mathcal{I}_1$ or $\mathcal{I}_3 \boxtimes_{A_2} \mathcal{I}_2$ are the same as \mathcal{I}_3 , i.e., no reduction of \mathcal{I}_3 by applying semi-composition. Although, if we take both \mathcal{I}_1 and \mathcal{I}_2 simultaneously into account yields a reduction of \mathcal{I}_3 (example cf. [30]).

Issue 2. Interface may not enough to restrict the behavior of system if the synchronization on given transitions can be decided nondeterministically.

To overcome these issues, [30] presented a new refined algorithm by using following strategy.

1. translate the composition of processes into a general model called “synchronization networks”;
2. extract an “interface network” from the network model;
3. generate the interface graph corresponding to the interface network.

For details of this algorithm we refer to [30]. Both versions of semi-composition are implemented in the PROJECTOR tool in CADP.

In our case, the major benefit of this algorithm is the simultaneous consideration of several neighbor processes as context constraints. As aforementioned, we can only determine the channel sizes after we generated the whole state space. As a result, the IMCs are defined as infinite. Based on these improvements on context constraints in parallel composition, we attempt to see whether our main issue, i.e., to bound the channel sizes efficiently and correctly, can be solved.

First attempt – bound the channel size by context

Back to our case, if we generate the IMC model of the SADF specification straightforwardly, we are naturally confronted with the state-explosion problem. With the aid of interface constraints, we may have a possible solution, and we elaborate the issues we encountered as follows. In LOTOS, two kinds of interfaces (for semi-composition) are accepted: either a user-given one or an interface generated automatically. Since a user given one is not possible here (too complicated and varies depend on the ordering of parallel composition), we concentrate on the interface that can be generated automatically. First, we consider the approach of Krimm & Mounier (non-refined). The problem here we encounter is the above mentioned issue 1: the context constraints of the neighbor processes of one process are too weak to reduce the state space of the process. Note that in order to apply the compositional aggregation with semi-composition to our model, we need to change our model a little: we separate the semantics of channels out of the detectors and kernels. If we compare the original IMC semantics described in Chapter 4, we notice that, in the original IMC semantics, all the IMCs to interpret the processes are infinite, since we integrate the unbounded channels into the processes. If we want to apply compositional aggregation, the better way is not to integrate the channels as variables into the processes’ semantics, which gives us a clear semantics. In our SADF specification, if we only consider the functionality of kernel or detector, the IMCs to express the kernel and detector are all finite, because the kernel and detector

finish their tasks in limited steps and then returns to their initial state, only the channels are treated with unbounded size. We do not know the exact size of the buffers before we have obtained the whole state space of the model. On the other hand, we are sure that the IMCs to represent the buffers in a bounded SADF graph are finite from SADF theory [41]. An illustration of the architecture of the modified semantic model of the example in Figure 4.11 is shown in Figure 5.7(a). In this example, we assume now the IMCs to express the processes A , B and D are finite due to the separation, and all the channels are modeled as infinite IMCs. Since the infinite IMCs are present, to use compositional aggregation on this model is obviously impossible. Our next possibility is try to apply the semi-composition approach from Krimm & Mounier. Again take the SADF graph from Figure 4.11 for example, let the expression \mathcal{I}_x express the IMC of process/channel x and the set G_i be the synchronization set between IMCs, then we assume now we obtained an IMC $\mathcal{I}_{DA} = \mathcal{I}_D \parallel_{G_1} \mathcal{I}_A$. If we want to parallel compose the two IMCs \mathcal{I}_{DA} and $\mathcal{I}_{CC_{d,a}}$, we need to find the boundary of channel $CC_{d,a}$, i.e., $\mathcal{I}_{CC_{d,a}}$ must be finite. This forces us to use the \mathcal{I}_{DA} as interface and to restrict the behavior of $CC_{d,a}$. But we fail here, since $\mathcal{I}_{CC_{d,a}} \boxtimes_{G_2} \mathcal{I}_{DA}$ is still infinite. The reason is the data channel $DC_{b,d}$ here, so far $DC_{b,d}$ has not been taken into account in our parallel composition and this means that the forced synchronization actions (should restrict the behavior of D) between detector D and $DC_{b,d}$ are treated as “free” actions. The detector D can get infinitely many data tokens based on these free actions, and fires for ever, and this causes the $CC_{d,a}$ to be infinite. Even with the refined approach, the simultaneous consideration of IMCs \mathcal{I}_A and \mathcal{I}_D as $\mathcal{I}_{CC_{d,a}}$'s context constraints has no effort to restrict the unbounded channel to bounded.

In conclusion of the first attempt, we have encountered some major issues while using compositional aggregation. First, we need to consider different (even maybe the rest of system) interface constraints simultaneously. As a result, using interface constraints to determine the exact size of buffers is not possible. Second, parallel composition of a finite IMC with an infinite IMC is not supported by CADP. Even we can generate a finite IMC of $((((\mathcal{I}_A \parallel_{G_3} \mathcal{I}_B) \parallel_{G_4} \mathcal{I}_{DC_{a,b}}) \parallel_{G_5} \mathcal{I}_{DC_{b,a}}) \parallel_{G_6} \mathcal{I}_D)$ in our case. This IMC is much large than the directly generated IMC, since the control channels are not involved here, kernel A and B can have more possibilities of execution in all “scenario sequence” (detector D responses for the determination of possible “scenario sequence”). On the other hand, if we want to parallel compose further using this finite IMC with an infinite IMC of a channel, this is not possible. (CADP will first generate the IMC of the unbounded channel).

Second attempt – first restriction then check

Another way is to combine the approaches from [30] and [43]. Also in [43], they have confronted the issue caused by unbounded buffer. In their model, if the size boundary

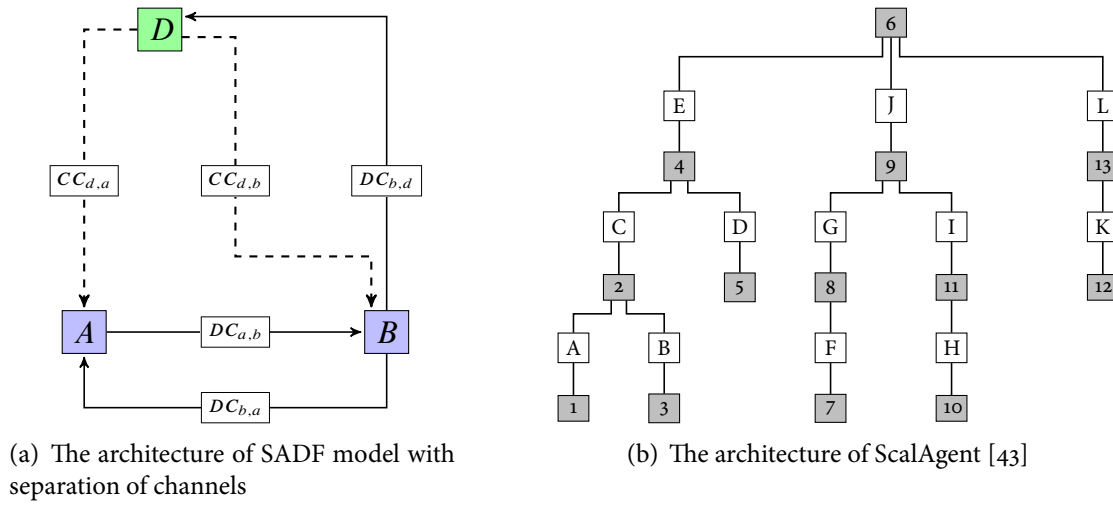


Figure 5.7: The architecture difference between SADF model and ScalAgent

of the buffer are not correctly determined, unexpected deadlocks or lost behavior will reveal due to the overflows of buffer. The approach used in [43] is to “guess” the bound of buffer then check whether the “guess” is correct. They first restrict the size of buffer from N (e.g. $N = 3$) places, and generate the LTS of the buffer. After the LTS of the buffer is parallel composed with its connected activities, the resulting LTS is used as interface to restrict the behavior of the buffer (e.g., general buffer includes all the possible content in the buffer, but in a specific system some content will never occur). Then, they check whether the size of buffer is correctly bounded. This is done by checking whether a sequence of N successive message exists in the LTS of the reduced buffer. If this is true, they increase the size of buffer and repeat the check experiment again. Otherwise, if they are lucky, the result comes out false, this means the size of buffer was bounded correctly.

The significant difference between their model and ours is the architecture of the system (cf. Figure 5.7). The sequential processes in their system are parallel composed in a hierarchical manner, whereas in our model the processes may mutually related. The “guess” of size of the buffer does not work well here: to generate the model, we need to guess all the bounded sizes of each channel at the same time, but obviously, different channels have different sizes. Only to test the combination of different sizes of all channels is a cumbersome thing. Assume we apply the similar strategy in [43]: we begin our test with the minimal size of each channel (here, we let maximal rate of all scenarios at the output port, to be the minimal size of the channel), we first generate the model by using compositional aggregation, and then check whether there is an overflow in any channel. If any channel has an overflow, we increase the size of that channel, and gener-

ate the system using compositional aggregation and check the occurrence of overflow again. One problem in this approach is the initial token plays an important role only at the end of composition aggregation as restrictions, even here the neighbor processes are used as interfaces. For instance, if we have a channel with maximal size of 3 but only 1 token in it initially. Using compositional aggregation, we need to generate IMC of the channel first. But after generation, this IMC can represent even 3 initial tokens in it (since the produce-action is free in the channel, 2 consequential produce-actions will lead the IMC to 3 tokens initially). This modification may cause state space explosion during generation. We know that the circular dependency makes sure the semantic model bounded if its representing SADF is bounded. But this circular dependency is based on many different processes which form a dependency circle in SADF, not only the neighbor processes. The channel with 1 initial token may at most times only 1 token in it and only for short time 3 tokens. But if we first generate the channel with maximal size 3, the generated IMC may contains 3 initial tokens in it and influences the further parallel composition with other processes. Only after the last component is involved in the system, the unreachable states caused by 2 or 3 initial tokens in that channel can be determined and be removed.

This is not the same situation as we can determine the bound of buffers in a bottom-up fashion [43]. The “neighborhood” context is still insufficient to filter the unreachable states earlier which can only be done by involving the last process in the system. Moreover, if the initial check size of buffer is already very large (say, in the MPEG-4 decoder example, every control channel is at least a 99-place buffer, which can store 9 different values. Only one control channel has already 99^9 states), and with such a large state space, even to generate the state space (LTS) of one channel is impossible. Another problem is caused by the separation of channels as individual IMCs, since many states and transitions used for synchronization between processes and channels can not be hidden immediately (the process must put tokens at the same time into all the channels, which are connected with it through its output ports, or the channels must notify the process of token availability etc.), these states which can not be hidden immediately and contribute for an exponential growth of state space.

To sum up, in our model, generally to predicate the exact size of channels (buffers) is even harder than direct generation of the model due to the recursive dependency. It also results the compositional aggregation approach is inapplicable (may work for small SADF). Since the buffers are not shared, to avoid reduce the redundant synchronization states and transitions, we integrate them into processes. By this way, the generation of state space becomes more efficient.

5.2.4 State Space Generation

In following sections, we will introduce the steps to generate the IMC based on the toolset in CADP and how to obtain the corresponding extended continuous-time Markov Chain (CTMC). The steps are shown in Figure 5.8.

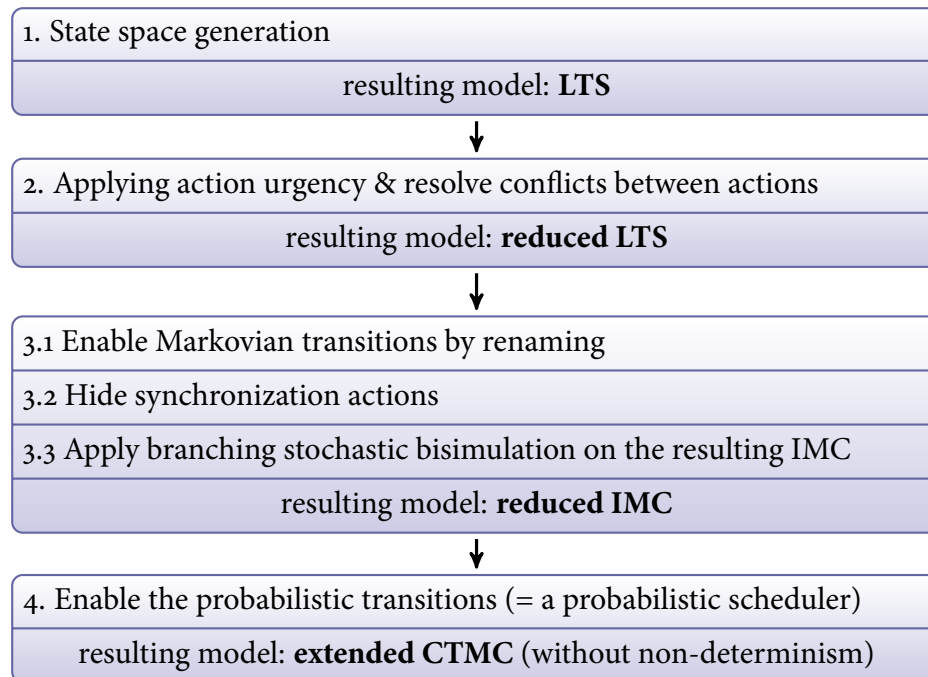


Figure 5.8: The steps of IMC state space generation based on the CADP toolset

In last sections, we introduced some syntax and semantics of LOTOS. To generate the state space of the system, whose behavior is described by using LOTOS, we employ the tool SVL in CADP. SVL is a scripting language, which provides convenient functions in the generation and verification tasks. There are some major functionalities of SVL:

- the operators in LOTOS also are also supported in the SVL language, it provides operators for considering context constraints, i.e. abstraction, refined abstraction of an expression w.r.t some interface;
- by using the so-called *meta-operations*, various compositional reduction approaches can be written more concisely;
- allows direct using Bourne Shell Script language in the SVL file (begin with %);
- includes operators for generating the explicit LTS of an express.

To generate the system of example in Figure 4.11, we only need to write a line in SVL:

```
"abd.bcg" = generation of "ABD.lotos";
```

This SVL will call the tools `CÆASR` and `CÆASR.ADT` to generate the LTS of the example for us. The resulting LTS is coded in a so-called “bcg” format [1]. As aforementioned, if we already hide all the internal synchronization actions in the `LOTOS` file, this LTS contents only two kinds of transitions: the internal τ -transition and transition with normal labels. Now, the Markovian transitions in this LTS are only labeled with normal labels named with “LAMBDA_Ps”, where P stands for the process name and s stands for the (sub)scenario name (this represents Markovian transitions of process p in (sub)scenario s in the IML expression).

5.2.5 Apply Action Urgency on Generated IMC

To apply action urgency assumption on the IMC generated by the first step, we need to specify the priority on actions. Since the synchronization actions are labelled with produce and the probabilistic transitions with “Prob_px” (both begin with “PRO-”, since all labels in generated IMC are upper case), we use the priority setting command in SVL:

```
"abdacturg.bcg" = generation of
    gate prio
        "PRO.*" > all but "PRO.*"
    in "abd.bcg"
    end prio;
```

With this command, all transitions with the labels that begin with “pro” have a higher priority than other transitions (the lambda transitions).

5.2.6 State Space Reduction

To turn this LTS into an IMC, we depend on the “rename” command in SVL. This command will call the tool `BCG_LABELS` to rename this transition labels and add *rate* infor-

mation to the labels. With the change of labels on the transitions, we can choose two kinds stochastic models.

1. Interactive Markov Chain, which contains the transitions of the form “rate % f ”.
2. Timed Processes for Performance (TIPP) Model [23], which contains the transitions of the form “ $label$; rate % f ”.

Since we need to know the execution status of each kernel and detector while we are doing performance evaluation, we adopt the form of TIPP. Note that we did not adopt the TIPP model before our state generation. Because the labels (action names) attached on the Markovian transitions are added after the generation of whole state space, these labels (actions) do not have any synchronization capability.

```
"abdimc.bcg" = total rename
    "LAMBDA_AS" -> "Execution_A; rate 0.0347",
    . . . . .
    "LAMBDA_DT" -> "Execution_D; rate 0.001"
in "abdacturg.bcg";
```

After the normal transitions named “LAMBDA_Ps” are replaced with the Markovian transitions in the first generated LTS, now we obtain a real IMC.

Then, we turn the transitions used for synchronization by hiding the produce actions.

```
"abdhid.bcg" = hide Produce_A, Produce_B, Produce_D
in "abdimc.bcg";
```

In the resulting IMC, three kinds of transitions are present, i.e. the internal transition τ , the normal transition labeled with “PRO_Px” (x is the identification number of the probabilistic transition belongs to it) and the Markovian transition “Execution_P; rate % f ”.

Note that until now, this IMC does not contain any real probabilistic transitions. These transitions are treated as normal transitions labeled with “Prob_px”. These normal transitions can be seen as non-deterministic choice if they are emitted from one state,

and if we rename the transitions with probabilistic labels, they become deterministic transitions solved by probabilistic choice. But before we do that, we first do stochastic branching bisimulation on the resulting IMC with following command.

```
"abdred.bcg" = branching stochastic reduction of
                "abdimc.bcg";
```

5.2.7 Adding Probabilistic Information

In the end of Chapter 4, we have discussed the non-determinism in the IMC model of SADF specification. After applying the branching stochastic bisimulation on the resulting IMC, only the non-determinism caused by probabilistic transitions of different process is left. Furthermore, there should exist no τ -transition any more due to the (stochastic) branching bisimulation and we check whether it is true with this SVL command:

```
% bcg_info "abdimc.bcg"
```

If there is only one detector in the SADF specification, we can skip the step of removing the non-determinism between probabilistic transitions. If there are more than one detector in the SADF graph, refer to the explanation in Section 4.3.1, we should determine the priority of the detectors. Since the probabilistic transitions are not renamed yet, the probabilistic transitions should have the form: "PRO_Dx", where D stands for the detector D and x stands for the identification number of this probabilistic transition belongs to the detector D . Again we use the priority setting in SVL to generate the IMC for us. Note that from the diamond property discussed in Section 4.3.1, we know that the priority settings on probabilistic transitions and internal interactive transitions (τ -transitions) will not affect the outcome of long-run averages, but has an impact on other metrics like buffer occupancy.

```
"prio2.bcg" = gate prio
                "PRO_D.*" > "PRO_G.*"
                .....
                in "prio1.bcg"
```

```
end prio;
```

Now we should turn the “pseudo” probabilistic transitions into real probabilistic transitions by the following command:

```
"abdredpro.bcg" = total rename
    "PRO_D1" -> "detD; prob 0.67",
    "PRO_D2" -> "detD; prob 0.33",
    "PRO_D3" -> "detD; prob 1.00"
in "abdprio.bcg";
```

The pure probabilistic transition is also fine here, but in contrast with the original TPS semantics of SADF specification, we employ the generative model.

Now our basic semantic model of SADF graph is totally generated. This model is a IMC with deterministic probabilistic transitions in it (can be seen as the non-deterministic branching transitions are assigned with weights). Our next task is to check the functional and performance properties in this model.

5.3 Model Checking on the resulting IMC

The stochastic process algebras (e.g. IMC) extends the classical process algebras with adding stochastic timing information. Rather than only functional verification is allowed, this semantic model allows the evaluation of various system aspects [13] (cf. 2.1):

- functional behavior (e.g. deadlocks)
- temporal behavior (e.g. throughput)
- combined properties (e.g. duration of certain event sequence)

5.3.1 Functional Verification

For functional verification we use the Action Computation Tree Logic (ACTL) [35, 34]. This logic is implemented using eXecutable Temporal Language (XTL) in CADP. XTL is a functional programming language which can implement various modal and temporal logics (e.g., HML, CTL, LTAC or μ -calculus) and these operators in the logics are evaluated over LTS [32]. XTL can even access the data values contained in the states, labels, transitions and sets, the predefined and use-defined functions are also provided to manipulate these data values. For example the set of deadlock states can be expressed by XTL expression:

```
{ S:state where succ (S) = empty }
```

Three basic predicates which we may use in our verification over actions and states are defined in the basic XTL. These predicates are shown in the following figure.

```
def INIT () : stateset = { init of state } end_def
def TAU () : labelset = { L: label where
                        not (visible (L)) } end_def
macro EVAL_A (A) = { L : label where L -> [ A ] } end_macro
```

Four primitive ACTL operators (cf. table 5.1) are used for any set of events A and formulas F, G , the semantics of these operators over the LTS \mathcal{I} are described as follows. Here we assume the given LTS $\mathcal{I} = (S, Act, \longrightarrow, s_0)$ and for each state $s \in S$ and $a_i \in Act$, let the $Path(s) = \{s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \dots\}$ be the execution sequences starting at s . The semantics of a state formula f (denoted as $\llbracket f \rrbracket$), is the set of states (labels) satisfying it in \mathcal{I} , similar a action formula a (denoted as $\llbracket a \rrbracket$) is the set of labels satisfying it in \mathcal{I} . Four derived notations from the major ACTL is also used for functional verification. With the aid of ACTL and the functions predefined in XTL, we can check some properties of our generated model. But a barrier exists in our model, since our model is now an IMC with Markovian transitions, we should turn it back to an LTS when we do functional verification. We do not choose the direct generated LTS at the first step of state space generation, since the action urgency and non-determinism are not applied to generated the LTS. So now we need to rename the Markovian transitions back to normal transitions and use the label pattern match function in XTL. The Markovian transitions in the model presents exactly the executions of processes in SADF specification, so we rename them back to the form “Execution_P” to represent the execution of a process “P”. The operators are used later in the second case study in Section 5.4.2.

XTL syntax	math syntax	semantics of operator
EX_A(A, F)	$EX_A F$	$\{s \mid \exists s' \xrightarrow{\alpha} s'. \alpha \in \llbracket A \rrbracket \wedge s' \in \llbracket F \rrbracket\}$
AX_A(A, F)	$AX_A F$	$\{s \mid \exists \alpha, s' : s \xrightarrow{\alpha} s' \wedge \forall \alpha, s' : s \xrightarrow{\alpha} s' \implies \alpha \in \llbracket A \rrbracket \wedge s' \in \llbracket F \rrbracket\}$
EU_A(F, A, G)	$E[F_A UG]$	$\{s \mid \exists p \in Path(s) \wedge \exists k > 0 : (\forall i = 0 \dots (k-1), p(i) \xrightarrow{\alpha} p(i+1) \implies (\alpha \in \llbracket A \rrbracket \cup \{\tau\}) \wedge p(i) \in \llbracket F \rrbracket) \wedge p(k) \in \llbracket G \rrbracket\}$
AU_A(F, A, G)	$A[F_A UG]$	$\{s \mid \forall p \in Path(s) \wedge \exists k > 0 : (\forall i = 0 \dots (k-1), p(i) \xrightarrow{\alpha} p(i+1) \implies (\alpha \in \llbracket A \rrbracket \cup \{\tau\}) \wedge p(i) \in \llbracket F \rrbracket) \wedge p(k) \in \llbracket G \rrbracket\}$

Table 5.1: Four major logic operators in ACTL

XTL syntax	math syntax	semantics of operator
Box(A, F)	$[A]F$	$\neg EX_A \neg F$
AG_A(A, F)	$AG_A F$	$\neg E[\text{tt}_A U \neg F]$
AG(F)	$AG F$	$AG_{\text{tt}} F$

Table 5.2: Four derived logic operators in ACTL

5.3.2 Performance Evaluation

Recall that the IMC we obtained at the end of model generation contains only Markovian transitions and probabilistic transitions. To analyze the temporal behavior of the resulting IMC, we need to eliminate the probabilistic transitions. Only after this step, we can obtain an equivalent CTMC to the original IMC with probabilistic transitions and do performance evaluations. The algorithm to eliminate the probabilistic transitions is described in [39] and implemented by the tool BCG_TRANSIENT and BCG_STEADY.

The basic idea to eliminate the probabilistic transitions is to collapse a Markovian transition with all immediately following probabilistic transitions (cf. Figure 5.9).

The standard elimination algorithm for probabilistic transitions is described as follows [39].

First the states of the transition system are divided into two categories.

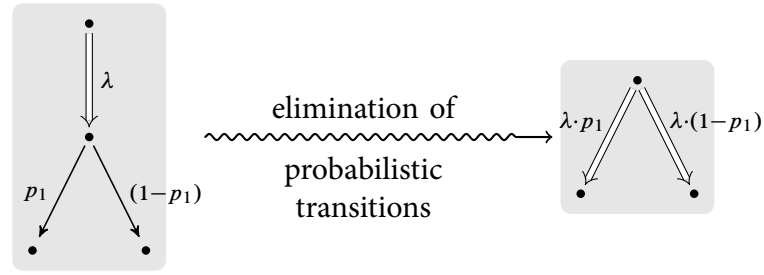


Figure 5.9: An example to collapse a Markovian transition with two immediately following probabilistic transitions

Definition 5.2 (Tangible and vanishing states) For a state s in the transition system with both Markovian transitions and probabilistic transitions, if at least one (possibly labelled) probabilistic transition goes out of s , then the state s is called a vanishing state. Otherwise, i.e., if there is no probabilistic transition goes out of s , then s is called a tangible state.

Then the Markovian transitions and probabilistic transitions in the transition system are translated into a matrix with the following shape:

$$\underline{Q} = \left(\begin{array}{c|c} \underline{M}_{tt} & \underline{M}_{tv} \\ \hline \underline{P}_{vt} & \underline{P}_{vv} \end{array} \right)$$

The \underline{M}_{tt} and \underline{M}_{tv} contain the Markovian transitions (from tangible) to tangible and vanishing states respectively, and \underline{P}_{vt} and \underline{P}_{vv} contain the probabilistic transitions (from vanishing) to tangible and vanishing states respectively. And a characteristic matrix is defined as:

$$\underline{B} = \{b_{ij}\} = (\underline{I} - \underline{P}_{vv})^{-1} \underline{P}_{vt},$$

where b_{ij} is the probabilities to reach the tangible state j from a vanishing state i via probabilistic transitions only. Then the transition rate matrix \underline{Q}' , which does not contain any probabilistic transitions is:

$$\underline{Q}' = \underline{M}_{tt} + \underline{M}_{tv}\underline{B}$$

Extended the algorithm with respect to the Markovian transitions together with action names (corresponding to the Markovian transitions of the form “Execution_P; rate % f ”, and the set of actions together with Markovian transition is denoted as Act_M), we get matrices ${}^a\underline{M}_{tt}$ and ${}^a\underline{M}_{tv}$ for each action $a \in Act_M$. The sum of rates of transitions

labelled with a leading from state i to state j is represented by the entry ${}^a m_{ij}^{tt}$ in the matrix ${}^a \underline{M}_{tt}$. Then we have:

$$\underline{Q} = \left(\begin{array}{c|c} {}^a \underline{M}_{tt} & \underline{M}_{tv} \\ \hline {}^a \underline{P}_{vt} & \underline{P}_{tt} \end{array} \right) \quad {}^a \underline{Q}' = \underline{M}_{tt} + \underline{M}_{tv} \underline{B}$$

Since in our model we have $\underline{P}_{vv} = 0$ (no probabilistic transition ends in a vanishing state),

$$\underline{B} = (\underline{I} - \underline{0})^{-1} \underline{P}_{vt}.$$

Then we have

$${}^a \underline{Q}' = \{ {}^a q_{ij} \} = {}^a \underline{M}_{tt} + {}^a \underline{M}_{tv} \underline{P}_{vt}.$$

By applying this algorithm on our IMC model, we can obtain a transition system (almost) without probabilistic transitions, which is equivalent to the original one. The only probabilistic transitions that cannot be removed are the probabilistic transitions emanating from the initial state (since no Markovian transition goes into the initial state). The possible solution is to give an initial weight distribution on these states, to which the probabilistic transitions from initial states are pointing, according to the probabilistic branching during temporal analysis.

The benefit of the above mentioned approach is that this algorithm computes an equivalent CTMC without any probabilistic transitions compared with the original system. Then the properties (e.g. reachability) that can be evaluated on CTMC are also available for our model. One performance property we are going to check is the time until one process finishes its first execution in the SADF specification.

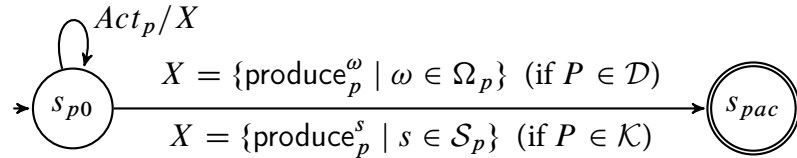
The handicap here is that logic like CSL cannot be used in our case, since we lack the tool to export the CTMC without probabilistic transitions out from the CADP toolset. On the other hand, the BCG_TRANSIENT or BCG_STEADY tools only allow us to do transient or steady analysis on the generated CTMC. If we want to know the expected time until one process finishes its first execution in the resulting CTMC model, the first problem is to find the execution sequences out from the generated IMC. Since the manner of parallel composition in IMCs forces the actions in the synchronization set to synchronize (TCSP-like). We use an IMC automaton (i.e. an extended IMC with an additional accepting state) to parallel compose with an IMC automaton derived from the original IMC representing the SADF model to obtain the traces which lead to the first execution of that process. More formally, we extend the IMC to an IMC automaton which has a set of accepting states. Let an IMC automaton $\mathcal{A} = (S, Act, \longrightarrow, \Longrightarrow, s_0, F)$, only the set F is defined as a nonempty set of states which are assigned as *accepting states*, other definitions are the same as in IMC. Further, for parallel composition of two IMC automata, say $\mathcal{A}_1 = (S_1, Act_1, \longrightarrow_1, \Longrightarrow_1, s_{0,1}, F_1)$ and $\mathcal{A}_2 = (S_2, Act_2, \longrightarrow_2, \Longrightarrow_2, s_{0,2}, F_2)$,

the parallel composition of \mathcal{A}_1 and \mathcal{A}_2 w.r.t. set G is defined as an IMC automaton $\mathcal{A}_1 \parallel_G \mathcal{A}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \longrightarrow, \Longrightarrow, (s_{0,1}, s_{0,2}), F)$. In the definitions, (initial) states and transition relations in the parallel composition of the two IMC automata are the same as the case of IMCs (cf. Definition 2.11), only the additional accept set F is defined as $F = F_1 \times F_2$. Now, we let $\mathcal{I} = (S, Act, \longrightarrow, \Longrightarrow, s_0)$ be the IMC representing a SADF graph G , then the corresponding IMC automaton \mathcal{A}_I of \mathcal{I} is defined as $\mathcal{A}_I = (S, Act, \longrightarrow, \Longrightarrow, s_0, S)$ (i.e., all the states in the \mathcal{I} are accepting states in \mathcal{A}_I). Then our goal is to find an automaton $\mathcal{A}_P = (S_p, Act_p, \longrightarrow_p, \Longrightarrow_p, s_{p0}, F_p)$, such that $\mathcal{A}_I \parallel_G \mathcal{A}_P = \mathcal{A}^f$, where in the automaton \mathcal{A}^f , all the paths from the initial state (s_0, s_{ac}) to the accepting (final) states represent the execution sequences until the the process P finishes its first execution (to get the expected time until the first execution of P , we can use the resulting IMC automata \mathcal{A}^f to parallel compose with another IMC automaton in a similar way).

We give the IMC automata $\mathcal{A}_P = (S_p, Act_p, \longrightarrow_p, \Longrightarrow_p, s_{p0}, F_p)$ and the synchronization set G as $\mathcal{A}_I = (S, Act, \longrightarrow, \Longrightarrow, s_0, S)$ follows:

$$\bullet \longrightarrow = \begin{cases} s_{p0} \xrightarrow{\text{produce}_p^\omega} s_{pac} & \text{if } P \in \mathcal{D}, \omega \in \Omega_p; \\ s_{p0} \xrightarrow{\text{produce}_p^s} s_{pac} & \text{if } P \in \mathcal{K}, \omega \in \mathcal{S}_p; \\ s_{p0} \xrightarrow{\alpha} s_{p0} & \text{if } P \in \mathcal{D}, \alpha \in Act / \{\text{produce}_p^\omega \mid \omega \in \Omega_p\}; \\ & \text{if } P \in \mathcal{K}, \alpha \in Act / \{\text{produce}_p^s \mid s \in \mathcal{S}_p\}, \end{cases}$$

- $G = Act_p = Act$
- $S_p = \{s_{p0}, s_{pac}\}$,
- $F_p = \{s_{pac}\}$,
- $\Longrightarrow = \emptyset$.



Here we choose the action produce-action of process P as the synchronization action to lead the states to the accept (final) states, since we know direct after an execution

of the process P (i.e. the Markovian transition of P) the produce-action is followed. These produce-actions are hidden later and removed by applying stochastic branching bisimulation, and only the Markovian transitions are kept. If we apply our approach to the example in Figure 4.11 and parallel compose the resulting IMC automata (cf. Figure 5.11), we get the IMC automata, whose traces from the initial state to accepting states represent the first execution of kernel B . The IMC automata we obtained is shown in Figure 5.10.

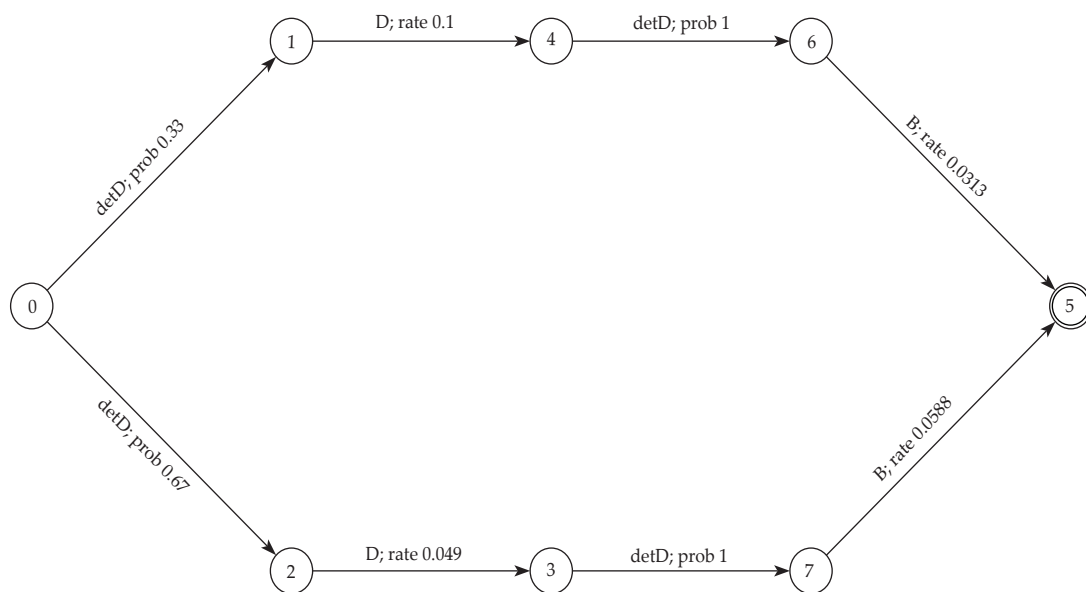


Figure 5.10: The IMC automata to represent the first execution of process B according to the example in Figure 4.11

5.4 Case Studies

In last section, we have shown how to generate the IMC of SADF specifications using the toolset CADP. In this section, we will discuss the results of state space of two examples, the example in Figure 4.11 and the a MPEG-4 decoder shown in Figure 4.1.

	Generation of state space	After sto. branching reduction	Remove of non-determinism	Reduction factor
1.Approach- separation of channels with processes				
# states	4081	18	15	226.72
# transitions	11785	32	21	368.28
2.Approach- integration of channels into processes				
# states	288	44	15	4.42
# transitions	155	26	21	6.55

Table 5.3: The results of state space in first case study

5.4.1 1. Case Study

In the first example, we compare the state space generated by two different approaches.

1. approach. Direct generation with separation of channels with processes.
2. approach. Direct generation with integration of channels into processes.

With the 1.approach, we get an IMC of 4081 states and 11785 transitions. Compare to the 2.approach, the state space is about 26-time larger than the 2.approach. The reason is that if we separate the channels with processes, the number of components increase from 3 to 8 (3 processes and 5 channels) compared with the 2.approach. Naturally this increase of the components for parallel composition causes a large state space. On the other hand, the separation also makes the system more complex. In the SADF model, the channels do not act as only the medium to store the information, but also need to notify the availability of token to the corresponding process, furthermore the channels do not know what the in which sub(scenario) the process is going to operate. The process must first communicate with the control channels, which connect through its control ports, and send the (sub)scenario to the corresponding data channels, and then wait for the notification from these channels. This cause an immense synchronization between the channels and processes. But after we hide these synchronization, all these transitions for synchronization turn into τ -transitions and generally the stochastic branching reduction can remove these transitions. In the 2.approach, we integrate the channels as variables in the processes, which fetch the tokens from the channels. This setting allows us to use *guarded* transitions to enable the execution of the processes when the tokens are enough. This is why the reduction factor of 1.approach is much higher than the 2.approach. The next step is to remove the non-determinism caused by the conflict between probabilistic transitions and Markovian transitions. Both approaches yield the same IMC with only 15 states and 21 transitions shown in Figure 5.11. In the first case

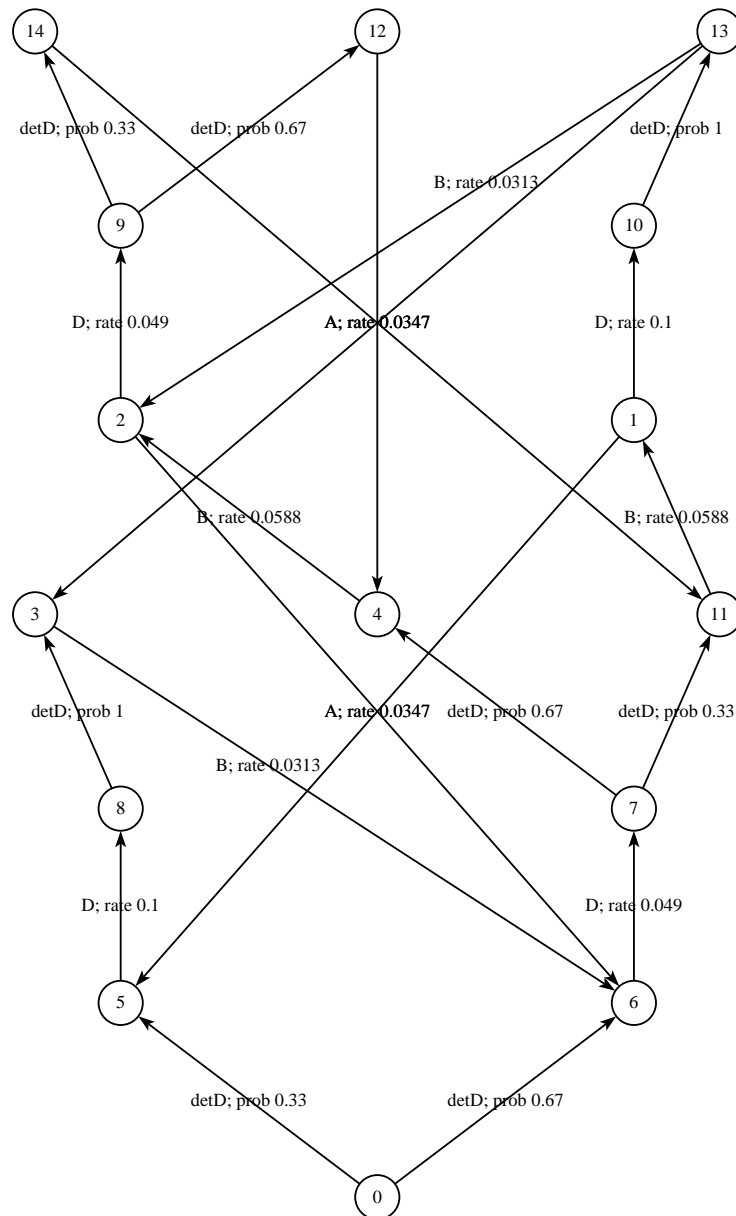


Figure 5.11: The resulting IMC of example in Figure 4.11

study, our aim is to compare two approaches of IMC semantics definitions. The 1. approach causes much synchronization than the second one. It encounters a much quicker state explosion problem, but if the compositional aggregation strategy is applicable in our state space generation phase, this problem will be alleviated by this strategy, since the relation between the components are looser than the 2. approach. The internal transitions between the components (processes and channels) can be reduced by branching bisimulation as soon as there is no further synchronization needed on these actions. In the 2. approach, the synchronization are now directly between the processes, and normally all actions used for synchronization are only allowed to be hidden until the whole system is generated, and the only reduction technique in composition aggregation can be used during each parallel composition is strong bisimulation. In the second case study, we only employ the 2. approach.

5.4.2 2. Case Study

In the second case study, we analyzed a more practical and complex model, the MPEG-4 decoder. In following experiments, the SADF model contains a whole scenario set, i.e., the processes can operate in the following (sub)scenarios: $I, P_0, P_{30}, P_{40}, P_{50}, P_{60}, P_{70}, P_{80}, P_{99}$ (9 scenarios in total). Two scenarios I and P_{99} contribute more to the growth of state space than other scenarios. The reason is that the every time FD finishes its firing, it is possible that FD produces 99 scenario tokens valued with I or P_{99} to the channels $CC_{fd,vld}$ and $CC_{fd,idet}$. Our goal is to generate the IMC of the MPEG-4 decoder with 1 and 3 initial tokens in the data channel $DC_{rc,mc}$ and $DC_{rc,fd}$ respectively, which contains the (sub)scenario $I, P_0, P_{30}, P_{40}, P_{50}, P_{60}, P_{70}, P_{80}, P_{99}$. To generate the system directly, we encountered the state explosion problem and could not generate the whole state space. In our machine, 192 GB RAM is used up without obtaining the resulting IMC. As a result, we first try to generate the system with only 1 and 2 initial tokens in the data channel $DC_{rc,fd}$. Here, we let N to denote the number of initial data tokens in the data channel $DC_{rc,fd}$. The results is shown in the table 5.4. We observed that the IMC with

N	Generation of state space		After reduction & removal of non-det.		Reduction factor of states
	states	transitions	states	transitions	
1	121430	230538	20664	40588	5.88
2	11843682	28429668	748813	2056149	15.82
3	?	?	?	?	?

Table 5.4: The results of state space with 9 scenarios ($I, P_0, P_{30}, P_{40}, P_{50}, P_{60}, P_{70}, P_{80}, P_{99}$) in the IMC

only 2 initial tokens in the data channel $DC_{rc,fd}$, the resulting IMC has more than 10 million states after the state space generation, and after reduction and removal of non-determinism on this IMC, we have about 700000 states with the reduction factor about 15. To have a hypothesis about the trend of growth of this IMC with 3 initial tokens

N	Generation of state space		After reduction & removal of non-det.		Reduction factor of states
	states	transitions	states	transitions	-
1	59406	122865	5051	99902	11.76
2	454420	1127321	24951	49503	18.21
3	1635890	4351159	64453	128308	25.38

(a) The results of only scenario I in the MPEG-4 decoder

N	Generation of state space		After reduction & removal of non-det.		Reduction factor of states
	states	transitions	states	transitions	-
1	66978	135994	7003	13686	9.56
2	1298878	3258351	83586	202554	15.54
3	10311656	28178676	477036	1234026	21.61

(b) The IMC state space when scenarios I, P_0, P_{60} in the MPEG-4 decoder

N	Generation of state space		After reduction & removal of non-det.		Reduction factor of states
	states	transitions	states	transitions	-
1	89166	174562	12580	24680	7.09
2	3853042	9395552	259145	679213	14.87
3	56867106	154377391	2661313	7527029	21.37

(c) The IMC state space when scenarios $I, P_0, P_{30}, P_{60}, P_{99}$ in the MPEG-4 decoder

N	Generation of state space		After reduction & removal of non-det.		Reduction factor of states
	states	transitions	states	transitions	-
1	62013	107649	15714	30785	3.95
2	7309391	17097494	554135	1511645	13.19
3 ²	44376497	118676755	2462606	6968605	18.02

(d) The IMC state space when scenarios $P_x (x = 30, 40, 50, 60, 70, 80, 99)$ in the MPEG-4 decoder**Table 5.5:** The conflict between probabilistic transitions

in $DC_{rc,fd}$ with 9 scenarios, we go on with our experiments. We reduce the scenario numbers in the scenario set. Note that one factor also contributes the growth of memory consumption: for the generation of IMC with 2 initial tokens in the data channel $DC_{rc,fd}$, the maximal natural numbers we used in the IMC to represent the numbers

²With $N = 3$, the generation of IMC of P_x only with $P_{30}, P_{40}, P_{50}, P_{60}, P_{70}$ possible.

the tokens in the channels (for both data and control channels) are smaller than 254, but with 3 initial tokens in $DC_{rc,fd}$, the detector can fire 3 times sequentially and produces at most 297 (99×3) I or P_{99} scenario tokens into the control channel $CC_{fd,vld}$ and $CC_{fd,mc}$. To generate such IMCs correctly, we need to use the X_NATURAL library in CADP, in which the natural number is externally implemented as unsigned int (32 bits) in C (the details cf. [3]). To compare these results, we try to generate the IMCs with four different scenario combinations: 1) only scenario token I , 2) 3 scenarios (I, P_{30}, P_{60}), 3) 5 scenarios ($I, P_0, P_{30}, P_{60}, P_{99}$), 4) 7 scenarios with P_x ($x=30,40,50,60,70,80,99$). Since the x in P_x represents the number of macro blocks in the P frames, the more distinguished values of x are used, the more accurate the system is to be modeled. But with reduced scenarios in the model, we can still achieve to model the MPEG-4 decoder with less accuracy. Now, we increase the initial data tokens in $DC_{rc,fd}$ from 1 to 3 and try to generate the IMCs. The results are shown in the table 5.5. The limitation of state space generation before state space reduction is when there are 3 initial data tokens in $DC_{rc,fd}$ and with only 5 scenarios ($I, P_0, P_{30}, P_{60}, P_{99}$) or ($P_{30}, P_{40}, P_{50}, P_{60}, P_{70}$). The memory consumption in both cases is over 100 GB, and when only one additional scenario is added, the generation is not possible anymore.

With comparison of the reduction factors, we observed that for $N = 1$, the reduction factor is about 10, for $N = 2$, the reduction factor is about 15, for $N = 3$ we do not obtain an reduction factor at the end, but from the other scenario combinations, it suggest that the reduction factor when $N = 3$ is about 20. Hence, the reduction factor increases when the N increases.

Another aspect is from the Figure 5.12 (red. stands for reduction in this figure), we looked at only the number of states in IMC before and after reduction (together with removal of non-determinism) with different scenario combinations. The value of X -axis represents the number of scenarios in the scenario combinations, i.e., the scenario combination of only one scenario I is 1, the combination (I, P_0, P_{60}) is 3, the combination ($I, P_0, P_{30}, P_{60}, P_{99}$) is 5, and P_x ($x=30,40,50,60,70,80,99$) is 7, the full scenario set is 9. From the figure, we can see that the reduction by letting the system modulo bisimulation is efficient. But one issue reveals here is the reduction approach may not sufficient to alleviate the quick growth of state space in the original IMC, since the growing rate (angle) between the scenario combinations using both approaches are almost the same. The state spaces after reduction does not grow slower than the reduction before. Another observation is the increase of N has a very strong influence on growing rate of state space. Comparing the state space growth between the scenario combination 1 and 3 in X -axis, the angle of growth is steeper and steeper if N increases. The linear growth of N from 1 to 3 does not cause a linear growth of the state space, but much

²With $N = 3$, the generation of IMC of P_x only with $P_{30}, P_{40}, P_{50}, P_{60}, P_{70}$ possible.

quicker. In this figure, it suggest that the generation of IMC with full scenario set and $N = 3$ may possible, if the compositional aggregation approach can be used during the state space generation, since the IMC after reduction should have less than 10^8 states.

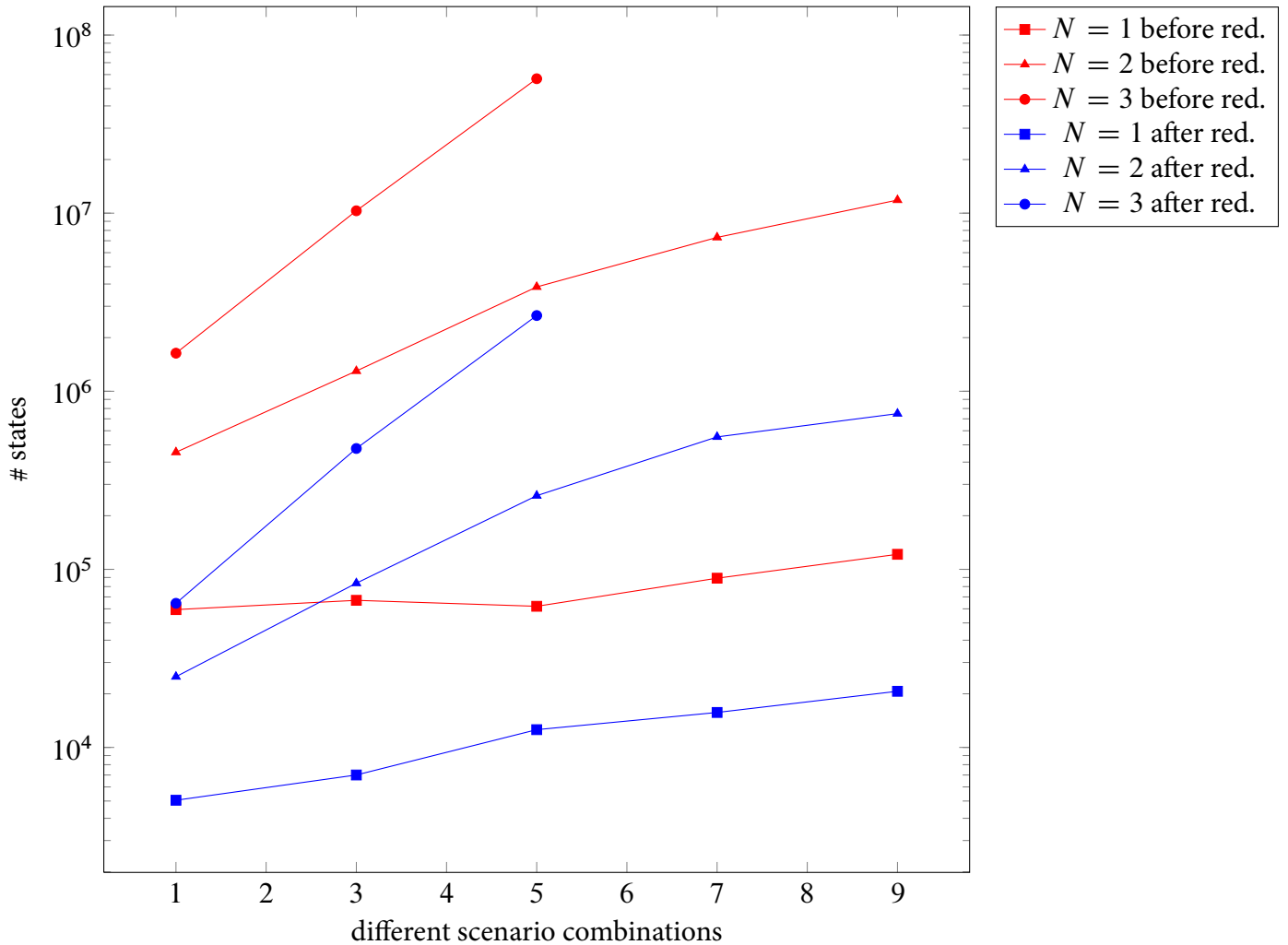


Figure 5.12: The state space of generated IMCs before and after reduction with different scenario combinations

Functional Properties

For the example of MPEG-4 decoder, we have checked following properties when $N = 1$:

1. Deadlocks: there should be no deadlock in the resulting IMC. Since the original SADF graphs are all bounded and deadlock-free [41], the IMCs we generate for these SADF graphs should also be bounded and deadlock-free. This property holds for all SADF models in our examples.
- * In the 2.case study, we check the properties on the generated IMC. Since we know that the Markovian transitions in the IMC represent the firings of processes, we first using the label matching function `EVAL_A(...)` to obtain the set of such transitions belonging to every processes in the SADF. Hence, for instance, the set `ERC` is all the Markovian transitions which are labelled with the `RC` in the SADF:

```
ERC   : labelset = {L:label where L ->[RC ?rate:real]},
EMC   : labelset = {L:label where L ->[MC ?rate:real]},
EIDCT : labelset = {L:label where L ->[IDCT ?rate:real]},
EVLD  : labelset = {L:label where L ->[VLD ?rate:real]},
```

2. After we obtained the set of transitions that indicates the firing of the processes. We can first check whether this property holds: initially, `ERC` (the firing of `RC`) will be eventually reached. This property should hold, but the result returns “false”. The reason is that the probabilistic self loop in the IMC caused by the scenario P_0 . Since the all the executions of processes in scenario P_0 cost no time, there exists self-loops of probabilistic transitions (The zero-time executions of processes are not modeled by interactive transitions in our IMC, they are simply ignored). That is why the result returns out “false”. If we use the set of `producerc` transitions to indicate the firing of `RC`, this property holds.

```
INIT implies AU_A_B (true, true, ERC, true)
```

3. The next property we check is : Between two consecutive `MC` firings, there must a `RC` firing. The should be evaluated to true. And, in our test, this property holds.

```
AG (Box (EMC, AG_A (not(ERC), Box (EMC, false))))
```

4. For `VLD` and `IDCT`, we check the property that: Initially, no `IDCT` firing can happen before a firing of `VLD`. This property holds.

```
INIT implies AG_A (not (EVLD), Box (EIDCT, false))
```

5. The property: after a firing of VLD , a firing of IDCT is eventually reachable, is evaluated as “false”. The reason is the same as the 2.property. If we use transitions labelled with produce_{vld} and produce_{idct} to indicate the firing of VLD and IDCT respectively, this property holds.

```
AG (Box (EVLd , AU_A_B (true , true , EIDCT , true)))
```

The time cost of evaluation each property using XTL is shown in following table.

Time cost of functional properties (in seconds)					
	P1	P2	P3	P4	P5
$N = 1$	1.4	22.5	2.6	1.3	12.2
$N = 2$	1.8	105032.	2693.	109.	54920.

Temporal Properties

1. The time until RC finishes its first execution A temporal property we want to check in our generated IMC is the probabilities of kernel RC finishes its first firing in x kCycle. We use the approach mentioned in the Section 5.3.2 with extended IMC automata to obtain an IMC, in which all the traces from the initial state to the accepting states indicates the execution sequence until the the process, say P , finishes its first firing. Here, we only applied the approach on the generated IMC with $N = 1$. One adaption is the transient analysis does not support more than one initial states which are probabilistically distributed. To let the probabilistic transitions emanating from the initial state to be merged into a Markovian transition, we manually add a warm-up “preparing” processing time (a Markovian transition with high rate) for the first firing of FD. This adaption makes sure the initial state is tangible. The result is shown in Figure 5.13.

It is no wonder that the probability of RC finishes its first firing at time zero is not equal to zero, but with a fixed probability 0.02. The reason is that the probabilistic self-loop transition caused by the scenario P_0 (details refer to the Section 4.3.1). The probability of FD sends scenario P_0 to each kernel is just 0.02 after each time FD has finished its firing. Since all the processes finishes their firing cost no time in P_0 , the probability of RC to finishes its firing at time zero is just 0.02. First after 1000 kCycle, the probability starts to increase due to VLD and IDCT have finished its firing in some scenarios and produce enough data tokens for MC and RC. Then after 2600 kCycle, the possibility of

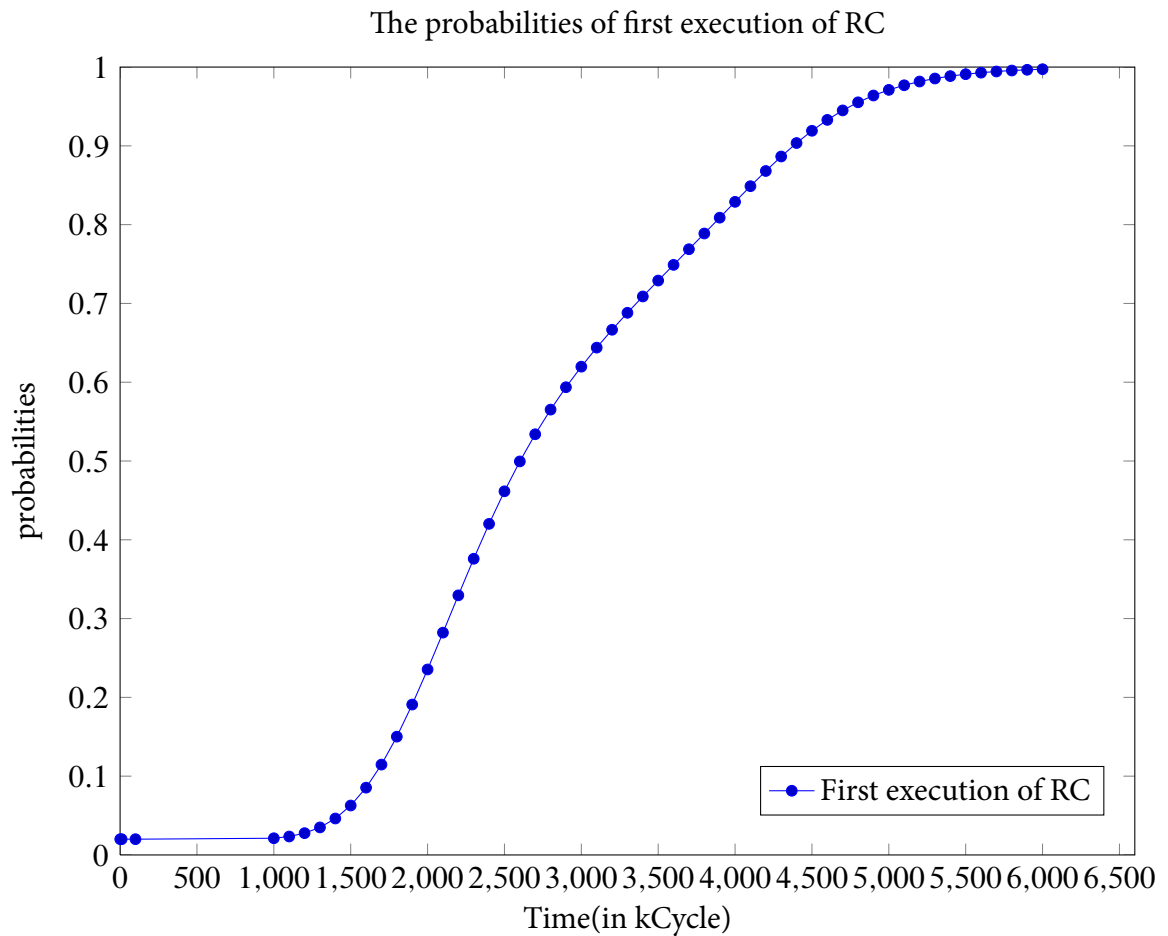


Figure 5.13: The probability of RC finishes its first firing in x time (in kCycle)

RC has finished its first firing is about 0.5. After 5500 kCycle, the probabilities are all above 0.99, i.e., almost 1.

2. The transition throughputs of processes³

The BCG_STEADY tool in CADP implements the Gauss/Seidel algorithm to compute the steady-state behavior of continuous-time Markov chains encoded in the BCG format [2]. Here, we use one option in this tool to compute the transition throughput of the processes. The transition throughput computes how much (a label attached on a Markovian transition) can be accomplished over a fixed time period. With BCG_STEADY tool, only

³Since the exact interpretation of the notion transition throughput is unavailable, the transition throughput in IMC may differ from the definition of throughput in SADF. The result obtained is for reference only.

	N=1	N=2	N=3 ⁴
VLD	0.0209017	0.0248695	0.0247597
IDCT	0.0209017	0.0248695	0.0247597
MC	0.000306018	0.000364109	0.000195987
RC	0.000348718	0.000414915	0.000326645

Table 5.6: The throughput of each kernel with $N = 1, 2, 3$

the throughput on the long run is computed. Recall that during the generation of IMC, the Markovian transitions have the form “A; rate % f ”, which represents the firing delay of the process A. Therefore, the transition throughput in the table 5.6 is interpreted as the throughput of each kernel in the MPEG-4 decoder. The time unit in the table is kCycle. One handicap here is the firings of processes in certain scenario which costs no time. The throughputs of processes *VLD* and *IDCT*, the processes *MC* and *RC* should be identical. It is slight different in our results, since the kernel *MC* costs no time to finish its firing in scenario *I* has no Markovian transition to represent such firing (no Markovian transition can represent an instantaneous firing). One remark on this computation is the unclarity that whether the probabilistic transitions are also taken into account during the computation. The result is provided here only for reference hence.

⁴The generated IMC is only with scenarios *I*, P_0 , P_{30} , P_{60} , P_{99} .

Chapter 6

Conclusion

In this thesis, we first introduced a natural semantic model called IMC, which takes the advantages of both classical process algebras and CTMCs. The orthogonal integration of these two models in IMC provides us a convenient way to model and analyze the systems. The classical process algebra framework in IMC supports the basic idea of constructing the system in an component-wise manner, which allows the system to be modeled by just parallel composing the components. On the other hand, the Markovian transitions representing the time passage in system specifications avoid complexity during modeling and the memoryless property of non-negative exponential distribution fits well with the semantics of parallel composition operator in IMC. Then, we introduced the model SADF, whose semantics we want to define in terms of IMC. The SADF model extends the traditional SDF model and allows to express the dynamism in modern streaming applications. The dynamism in SADF is achieved by using the (sub)scenarios to set the processes into certain operating contexts. The (sub)scenarios in which the process operates are determined by using the Markov chains embedded in the detectors. The original operational semantics of SADF is based on the timed probabilistic (labelled transition) system (TPS). We designed a semantic framework to translate the original SADF models in terms of IMC by using delays governed by exponential distributions to represent the firing of processes instead of constant firing time in SADF. This formal definition is implemented by using CADP toolset developed by the VASY team at INRIA Rhone-Alpes. We also analyzed the non-determinism in the resulting IMC and some issues encountered during the state space generation. Based on the generated state space of IMC, we can do both functional verification and performance evaluation on the system. We used the ACTL logic to check the functional properties (e.g., deadlock, certain execution sequence of processes) of the IMC representing the SADF model and the temporal property like time until one process finishes

its first execution is checked based on the transient analysis on CTMC with the aid of extension of IMCs to IMC automata.

6.1 Future Work

Since we can not export the IMCs from the CADP toolset in which the probabilistic transitions in the original IMC are eliminated, we can only analyze the steady and transient properties within the CADP toolset. Model checking with CSL logic [5] is not available here to check the properties like “the expected time between two successive execution of kernel RC ”. Therefore, one direction of future work is to export the IMC without probabilistic transitions (may be a CTMC) out of CADP, and analyze the resulting IMC. On the other hand, the IMC is a behavior-oriented model which lacks of state-oriented support. For instance, the time-average buffer occupancy can be computed if we can get the steady probability of each state in the extended CTMC and the current buffer numbers (variable values) of all channels in that states. Therefore, including the variable values in the states of IMC (extended CTMC) is also a need, when we want to check more properties of SADF. In our experiments, we have only consider the scenario decision in the SADF specification based on a discrete-time Markov chain in the detectors. If the scenario decision is changed to a non-determinism label transition system (FSM) [18], the inherent non-determinism reveals in our IMC model. An even more complex situation can happen when the execution (firing) times of processes are also probabilistic distributed over different time sample space [41]. Then, the corresponding IMC model bears internal transitions that cause non-determinism, Markovian transitions and probabilistic transitions. The algorithm used in 5.3.2 to eliminate the probabilistic transitions are no longer feasible, since non-determinism caused by τ -transitions may exist between the Markovian transition and probabilistic transition. Novel algorithm to eliminate the probabilistic transitions in the IMC is needed, then the tool IMCA [4] may used to do performance evaluation.

Bibliography

- [1] Bcg manual page. <http://www.inrialpes.fr/vasy/cadp/man/bcg.html>, . Accessed: 09/02/2010.
- [2] Bcg_steady manual page. http://fmt.cs.utwente.nl/tools/pdac/man_pages/bcg_steady.html, . Accessed: 30/04/2010.
- [3] Frequently Asked Questions / Forum Aux Questions about the CADP Software. <http://www.inrialpes.fr/vasy/cadp/faq.html#section7.15>. Accessed: 18/03/2010.
- [4] The Interactive Markov Chain Analyzer (IMCA) tool. <http://www-users.rwth-aachen.de/dennis.guck/imca/index.html>. Accessed: 02/04/2010.
- [5] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.
- [6] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle, editors. *Validation of Stochastic Systems - A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science*, 2004. Springer. ISBN 3-540-22265-0.
- [7] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-checking Techniques and Tools*. Springer-Verlag New York, Inc., New York, NY, USA, 1999. ISBN 3-540-41523-8.
- [8] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2007. ISBN 978-3-540-75595-1.

- [9] Mario Bravetti and Pedro R. D'Argenio. Tutte le Algebre Insieme: Concepts, Discussions and Relations of Stochastic Process Algebras with General Distributions. In Baier et al. [6], pages 44–88. ISBN 3-540-22265-0.
- [10] Ed Brinksma, Holger Hermanns, and Joost-Pieter Katoen, editors. *Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science, Berg en Dal, The Netherlands, July 3-7, 2000, Revised Lectures*, volume 2090 of *Lecture Notes in Computer Science*, 2001. Springer. ISBN 3-540-42479-2.
- [11] Peter Buchholz. Exact and Ordinary Lumpability in Finite Markov Chains. *Journal of Applied Probability*, 31(1):pp. 59–75, 1994. ISSN 00219002. URL <http://www.jstor.org/stable/3215235>.
- [12] Shing-Chi Cheung and Jeff Kramer. Enhancing Compositional Reachability Analysis with Context Constraints. In *SIGSOFT FSE*, pages 115–125, 1993.
- [13] Allan Clark, Stephen Gilmore, Jane Hillston, and Mirco Tribastone. Stochastic Process Algebras. In Marco Bernardo and Jane Hillston, editors, *SFM*, volume 4486 of *Lecture Notes in Computer Science*, pages 132–179. Springer, 2007. ISBN 978-3-540-72482-7.
- [14] Nicolas Coste, Hubert Garavel, Holger Hermanns, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. Ten Years of Performance Evaluation for Concurrent Systems using cadp. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (2)*, volume 6416 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2010. ISBN 978-3-642-16560-3.
- [15] Pepijn Crouzen and Holger Hermanns. Aggregation Ordering for Massively Compositional Models. In Luís Gomes, Victor Khomenko, and João M. Fernandes, editors, *ACSD*, pages 171–180. IEEE Computer Society, 2010. ISBN 978-0-7695-4066-5.
- [16] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. On Probabilistic Automata in Continuous Time. In *LICS*, pages 342–351. IEEE Computer Society, 2010. ISBN 978-0-7695-4114-3.
- [17] Hubert Garavel and Frédéric Lang. SVL: A Scripting Language for Compositional Verification. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *FORTE*, volume 197 of *IFIP Conference Proceedings*, pages 377–394. Kluwer, 2001. ISBN 0-7923-7470-3.

-
- [18] Marc Geilen and Sander Stuijk. Worst-case Performance Analysis of Synchronous Dataflow Scenarios. In Tony Givargis and Adam Donlin, editors, *CODES+ISSS*, pages 125–134. ACM, 2010. ISBN 978-1-60558-905-3.
- [19] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002. ISBN 3-540-44261-8.
- [20] Holger Hermanns and Joost-Pieter Katoen. The How and Why of Interactive Markov Chains. In Frank S. de Boer, Marcello M. Bonsangue, Stefan Hallerstede, and Michael Leuschel, editors, *FMCO*, volume 6286 of *Lecture Notes in Computer Science*, pages 311–337. Springer, 2009. ISBN 978-3-642-17070-6.
- [21] Holger Hermanns and Michael Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In *in Proc. of the 2nd Workshop on Process Algebras and Performance Modelling (PAPM '94)*, pages 71–87, 1994.
- [22] Holger Hermanns and Markus Siegle. Bisimulation Algorithms for Stochastic Process Algebras and Their BDD-Based Implementation. In Joost-Pieter Katoen, editor, *ARTS*, volume 1601 of *Lecture Notes in Computer Science*, pages 244–264. Springer, 1999. ISBN 3-540-66010-0.
- [23] Holger Hermanns, Ulrich Herzog, and Vassilis Mertsiotakis. Stochastic Process Algebras - Between LOTOS and Markov Chains. *Computer Networks*, 30(9-10): 901–924, 1998.
- [24] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. Towards Model Checking Stochastic Process Algebra. In Wolfgang Grieskamp, Thomas Santen, and Bill Stoddart, editors, *IFM*, volume 1945 of *Lecture Notes in Computer Science*, pages 420–439. Springer, 2000. ISBN 3-540-41196-8.
- [25] Ulrich Herzog. Formal Methods for Performance Evaluation. In Brinksma et al. [10], pages 1–37. ISBN 3-540-42479-2.
- [26] Joost-Pieter Katoen. Labelled Transition Systems. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 615–616. Springer, 2004. ISBN 3-540-26278-4.
- [27] Joost-Pieter Katoen and Pedro R. D'Argenio. General Distributions in Process Algebra. In Brinksma et al. [10], pages 375–430. ISBN 3-540-42479-2.

-
- [28] Joost-Pieter Katoen, Tim Kemna, Ivan S. Zapreev, and David N. Jansen. Bisimulation Minimisation Mostly Speeds Up Probabilistic Model Checking. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2007. ISBN 978-3-540-71208-4.
- [29] Jean-Pierre Krimm and Laurent Mounier. Compositional State Space Generation from Lotos Programs. In Ed Brinksma, editor, *TACAS*, volume 1217 of *Lecture Notes in Computer Science*, pages 239–258. Springer, 1997. ISBN 3-540-62790-1.
- [30] Frédéric Lang. Refined Interfaces for Compositional Verification. In Elie Najm, Jean-François Pradat-Peyre, and Véronique Donzeau-Gouge, editors, *FORTE*, volume 4229 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2006. ISBN 3-540-46219-8.
- [31] Edward A. Lee and David G. Messerschmitt. Synchronous Data Flow: Describing Signal Processing Algorithm for Parallel Computation. In *COMPCON*, pages 310–315. IEEE Computer Society, 1987. ISBN 0-8186-0764-5.
- [32] R. Mateescu and H. Garavel. XTL: A Meta-Language and Tool for Temporal Logic Model-Checking. In Tiziana Margaria, editor, *STTT'98*, 1998.
- [33] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. ISBN 3-540-10235-3.
- [34] Rocco De Nicola. Action and State-based Logics for Process Algebras. In Jos C. M. Baeten and Jan Friso Groote, editors, *CONCUR*, volume 527 of *Lecture Notes in Computer Science*, pages 20–22. Springer, 1991. ISBN 3-540-54430-5.
- [35] Rocco De Nicola and Frits W. Vaandrager. Action versus State based Logics for Transition Systems. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer, 1990. ISBN 3-540-53479-2.
- [36] Xavier Nicollin and Joseph Sifakis. An Overview and Synthesis on Timed Process Algebras. In Kim Guldstrand Larsen and Arne Skou, editors, *CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer, 1991. ISBN 3-540-55179-4.
- [37] G. D. Plotkin. *A Structural Approach to Operational Semantics*, 1981.
- [38] Gordon D. Plotkin. A Structural Approach to Operational Semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.
- [39] Michael Rettelbach. Probabilistic Branching in Markovian Process Algebras. *Comput. J.*, 38(7):590–599, 1995.

-
- [40] Ph. Schnoebelen. The Verification of Probabilistic Lossy Channel Systems. In Baier et al. [6], pages 445–466. ISBN 3-540-22265-0.
- [41] Bart D. Theelen, Marc Geilen, Twan Basten, Jeroen Voeten, Stefan Valentin Gheorghita, and Sander Stuijk. A Scenario-aware Data Flow Model for Combined Long-run Average and Worst-case Performance Analysis. In *MEMOCODE*, pages 185–194. IEEE, 2006.
- [42] B.D. Theelen, M.C.W. Geilen, S. Stuijk, S.V. Gheorghita, T. Basten, J.P.M. Voeten, and A.H. Ghamarian. Scenario-Aware Dataflow. Technical report esr-2008-08, Eindhoven University of Technology, Eindhoven (The Netherlands), July 2008.
- [43] Frederic Tronel, Frédéric Lang, and Hubert Garavel. Compositional Verification Using CADP of the ScalAgent Deployment Protocol for Software Components. In Elie Najm, Uwe Nestmann, and Perdita Stevens, editors, *FMOODS*, volume 2884 of *Lecture Notes in Computer Science*, pages 244–260. Springer, 2003. ISBN 3-540-20491-1.
- [44] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, Generative and Stratified Models of Probabilistic Processes. *Inf. Comput.*, 121(1):59–80, 1995.
- [45] Jeroen Voeten. Performance evaluation with temporal rewards. *Perform. Eval.*, 50(2/3):189–218, 2002.
- [46] Lijun Zhang and Martin R. Neuhäuser. Model Checking Interactive Markov Chains. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2010. ISBN 978-3-642-12001-5.

