Dipartimento di Informatica
Università del Piemonte Orientale "A. Avogadro"
Via Bellini 25/G, 15100 Alessandria
http://www.di.unipmn.it

Markov Decision Petri Net and Markov Decision Well-formed Net formalisms

Author: M. Beccuti (beccuti@mfn.unipmn.it), G. Franceschinis (giuliana.franceschinis@mfn.unipmn.it) and S. Haddad (haddad@lamsade.dauphine.fr)

TECHNICAL REPORT TR-INF-2007-02-01-UNIPMN
(February 2007)

The University of Piemonte Orientale Department of Computer Science Research Technical Reports are available via WWW at URL `http://www.di.unipmn.it/`. Plain-text abstracts organized by year are available in the directory

Recent Titles from the TR-INF-UNIPMN Technical Report Series

2006-03 *New challenges in network reliability analysis*, Bobbio, A., Ferraris, C., Terruggia, R., November 2006.

2006-03 *The Engineering of a Compression Boosting Library: Theory vs Practice in BWT compression*, Ferragina, P., Giancarlo, R., Manzini, G., June 2006.

2006-02 *A Case-Based Architecture for Temporal Abstraction Configuration and Processing*, Portinale, L., Montani, S., Bottrighi, A., Leonardi, G., Juarez, J., May 2006.

2006-01 *The Draw-Net Modeling System: a framework for the design and the solution of single-formalism and multi-formalism models*, Gribaudo, M., Codetta-Raiteri, D., Franceschinis, G., January 2006.

2005-06 *Compressing and Searching XML Data Via Two Zips*, Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S., December 2005.

2005-05 *Policy Based Anonymous Channel*, Egidi, L., Porcelli, G., November 2005.

2005-04 *An Audio-Video Summarization Scheme Based on Audio and Video Analysis*, Furini, M., Ghini, V., October 2005.

2005-03 *Achieving Self-Healing in Autonomic Software Systems: a case-based reasoning approach*, Anglano, C., Montani, S., October 2005.

2005-02 *DBNet, a tool to convert Dynamic Fault Trees to Dynamic Bayesian Networks*, Montani, S., Portinale, L., Bobbio, A., Varesio, M., Codetta-Raiteri, D., August 2005.

2005-01 *Bayesan Networks in Reliability*, Langseth, H., Portinale, L., April 2005.

2004-08 *Modelling a Secure Agent with Team Automata*, Egidi, L., Petrocchi, M., July 2004.

2004-07 *Making CORBA fault-tolerant*, Codetta Raiteri D., April 2004.

2004-06 *Orthogonal operators for user-defined symbolic periodicities*, Egidi, L., Terenziani, P., April 2004.

2004-05 *RHENE: A Case Retrieval System for Hemodialysis Cases with Dynamically Monitored Parameters*, Montani, S., Portinale, L., Bellazzi, R., Leonardi, G., March 2004.

2004-04 *Dynamic Bayesian Networks for Modeling Advanced Fault Tree Features in Dependability Analysis*, Montani, S., Portinale, L., Bobbio, A., March 2004.

2004-03 *Two space saving tricks for linear time LCP computation*, Manzini, G., February 2004.

2004-01 *Grid Scheduling and Economic Models*, Canonico, M., January 2004.

Markov Decision Petri Net and Markov Decision Well-formed Net formalisms

M. Beccuti, G. Franceschinis[1] and S. Haddad[2]

[1] Univ. del Piemonte Orientale,
giuliana.franceschinis@mfn.unipmn.it
[2] LAMSADE CNRS, Univ. Paris Dauphine
haddad@lamsade.dauphine.fr

February 2007

Abstract

In this work, we propose two high-level formalisms, *Markov Decision Petri Nets* (MDPNs) and *Markov Decision Well-formed Nets* (MDWNs), useful for the modeling and analysis of distributed systems with probabilistic and non deterministic features: these formalisms allow a high level representation of Markov Decision Processes. The main advantages of both formalisms are: a macroscopic point of view of the alternation between the probabilistic and the non deterministic behaviour of the system and a syntactical way to define the switch between the two behaviours. Furthermore, MDWNs enable the modeller to specify in a concise way similar components. We have also adapted the technique of the symbolic reachability graph, originally designed for Well-formed Nets, producing a reduced Markov decision process w.r.t. the original one, on which the analysis may be performed more efficiently. Our new formalisms and analysis methods are already implemented and partially integrated in the GreatSPN tool, so we also describe some experimental results.

Introduction

Markov Decision Processes (MDP). Since their introduction in the 1950's, Markov Decision process models have gained recognition in numerous fields including computer science and telecommunications [15]. Their interest relies on two complementary features. On the one hand, they provide to the modeler a simple mathematical model in order to express optimization problems in random environments. On the other hand, a rich theory has been developed leading to efficient algorithms for most of the practical problems.

Distributed Systems and MDPs. The analysis of distributed systems mainly consists in (1) a modeling stage with some high-level formalism (like Petri nets or process algebra), (2) the verification of properties expressed in some logic (like LTL or CTL) and (3) the computation of performance indices by enlarging the model with stochastic features and applying either (exact or approximate) analysis methods or simulations. In this framework, a MDP may be viewed as a model of a distributed system where it is possible to perform a non deterministic choice among the enabled actions (*e.g.*, the scheduling of tasks) while the effect of the selected action is probabilistic (*e.g.*, the random duration of a task). Then, with appropriate techniques, one computes the probability that a property is satisfied w.r.t. the "worst" or the "best" behavior [2, 6]. The time model that will be considered in this paper is discrete: each non deterministic choice is taken in a given decision epoch, after the probabilistic consequence of the choice has been performed, a new decision epoch starts.

Here the way we model distributed systems by MDPs is rather different. During a stage, the system evolves in a probabilistic manner until periodically a (human or automatic) supervisor takes the control in order to configure, adapt or repair the system depending on its current state before the next stage. In other words, usual approaches consider that the alternation between non deterministic and probabilistic behavior occurs at a microscopic view (*i.e.*, at the transition level) whereas our approach adopts a macroscopic view of this alternation (*i.e.*, at a stage level). It should be emphasized that, depending on the applications, one or the other point of view should be preferred and that the user should have an appropriate formalism and associated tools for both cases. For instance PRISM [11], one of the most used tools in this context, works at the microscopic level whereas the formalism of *stochastic transition systems* is

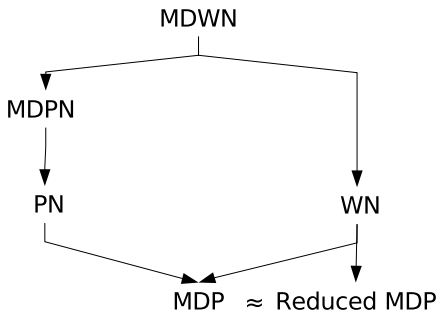


Figure 1: From nets to Markov decision processes

based on a macroscopic view [7]. The latter formalism is a slight semantical variation of generalized stochastic Petri nets [14] where the choice among the enabled immediate transitions is performed non deterministically rather than probabilistically. Despite its simplicity, this formalism has a serious drawback for the design process since the modeler has no mean to syntactically define the switches between the probabilistic behavior and the non deterministic one. Furthermore, the difference between the *distributed* feature of the probabilistic behavior and the *centralized* one of the non deterministic behavior is not taken into account.

Our contribution. In this work, we propose a high-level formalism in order to model distributed systems with non deterministic and probabilistic features. Our formalism is based on *Well-formed Petri Nets* (WN) [3]. First, we introduce *Markov Decision Petri nets* (MDPN): an MDPN is defined by three parts, a set of active components (*e.g.*, processes or machines), a probabilistic net and a non deterministic net. Every transition of the probabilistic net is triggered by a subset of components. When every component has achieved the activities related to the current probabilistic stage, the supervisor triggers the non deterministic transitions in order to take some decisions, either relative to a component or global. Every transition has an attribute (run/stop) which enables the modeler to define when the switches between the nets happen. The semantics of this model is designed in two steps: a single Petri net can be derived from the specification and its reachability graph can be transformed with some additional information, also specified at the MDPN level, into an MDP.

Distributed systems often present symmetries *i.e*, in our framework, many components may have a similar behavior. Thus, both from a modeling and an analysis point of view, it is interesting to look for a formalism expressing and exploiting behavioral symmetries. So we also define *Markov Decision Well-formed nets* (MDWN) similarly as we do for MDPNs. The semantics of a model is then easily obtained by translating a MDWN into a MDPN. Furthermore, we develop an alternative approach: we transform the MDWN into a WN,

then we build the symbolic reachability graph of this net [4] and finally we transform this graph into a reduced MDPN w.r.t. the original one. We prove that we can compute on this reduced MDP, the results that we are looking for in the original MDP. The different relations between the formalisms are shown in Fig. 1. Finally we have implemented our analysis method within the GreatSPN tool [5] and performed some experiments.

The paper is organized as follows: chapter 1 introduces the MDPN formalism, while chapter 2 the MDWN formalism. Finally in chapter 3 we conclude and give some perspectives .

Chapter 1

Markov Decision Petri Nets

In this chapter we are going to introduce a first high level formalism from which an MDP can be derived: Markov Decision Petri Nets (MDPN) based on Petri Nets (PN)[1] with priorities.

In section 1.1 we introduce the Markov Decision Process formalism and we show with an example that it may not be easy to model a realistic distributed system at the MDP level.

In section 1.2 we introduce the Markov Decision Petri Net formalism; while section 1.3 shows how to use the MDPN formalism in order to model and study a distributed system.

Finally, in section 1.4 we summarize and discuss the characteristics of this new formalism.

1.1 Introduction

In general we want to model systems composed by multiple active components whose behavior during a period is described in a probabilistic way and a centralized decision maker taking some decisions between execution periods (*e.g.*, assigning available resources to components).

A first example could be a system with two (anonymous) components that can be in service or out of service and a centralized decision maker that can choose between different repair policies.

The system can be summarized as follows:

- two similar components; that can be *down* or *up*. If a component is out of order then it is in the state *down* else is in the state *up*;
- a decision maker (or failure detection and recovery system) which must take for every component in every time unit one of the two following actions/decisions:

[1] we consider that the reader has already familiarity with the PN notation in any case more details can be found in [14]

  - repair a component in the state down;
  - do not repair a component (in the state down);

- a limited number of failures can be recovered in parallel (limited restore resources).

We assume that the system maintainer must pay a penalty when all the components are down and a repair cost for every assigned restore resource. Hence

$$C_{penalty}(state) = \begin{cases} C_{down} & if\ state = \ all\ components\ down \\ 0 & otherwise \end{cases}$$

where $C_{down} > 0$ is a fixed penalty cost, and

$$C_{repair}(action) = \begin{cases} C_{AssignRes} & if\ action = AssignResource \\ 0 & otherwise \end{cases}$$

where $C_{AssignRes} > 0$ is a fixed repair cost.

These systems can be efficiently modeled using the MDP; in fact the MDP formalism gives the possibility to perform a non deterministic choice among the enabled actions (e.g. repair/do not repair) while the effect of the selected action is probabilistic (e.g. the random duration of a task).

A (discrete time and finite) MDP is a dynamic system where the transitions between states are defined as follows: let $s \in S$ be the current state, first an action a is selected non deterministically among the subset of actions currently enabled (denoted A_s), then the new state is obtained by sampling from a probability distribution depending on s and a (denoted, $p(\cdot|s,a)$). An MDP includes rewards associated with state transitions; here, we choose a slightly restricted definition of the rewards that do not depend on the destination state but only on the source state and the chosen action (denoted, $r(s,a)$). Starting from such elementary rewards, different kinds of global rewards may be associated with a finite or infinite execution thus raising the problem to find an optimal strategy w.r.t. a global reward. For the sake of simplicity, we restrict the global rewards to be either the *total reward* or the *average reward* (depending on the nature of the analysis, finite horizon i.e. transient or infinite horizon i.e. steady state). The next definitions formalize these concepts.

Definition 1 (MDP) *An MDP $\mathcal{M}$ is a tuple $\mathcal{M} = \langle S, A, p, r\rangle$ where:*

- *S is a finite set of states,*
- *A is a finite set of actions defined as $\bigcup_{s\in S} A_s$ where A_s is the set of enabled actions in state s,*
- *$\forall s \in S, \forall a \in A_s, p(\cdot|s,a)$ is a (transition) probability distribution over S such that $p(s'|s,a)$ is the probability to reach s' from s by triggering action a,*

- $\forall s \in S, \forall a \in A_s, r(s,a) \in \mathbb{R}$ *the reward associated with state* s *and action* a.

A finite (resp. infinite) execution of an MDP is a finite (resp. infinite) sequence $\sigma = s_0 a_0 \dots s_n$ *(resp.* $\sigma = s_0 a_0 \dots$*) of alternating states and actions, s.t.* $\forall i, s_i \in S \wedge a_i \in A_{s_i}$ *and* $p(s_{i+1}|s_i, a_i) > 0$.

The total reward of such an execution is defined by $trw(\sigma) = \sum_{i=0}^{n-1} r(s_i, a_i)$ *(resp.* $trw(\sigma) = \lim_{n\to\infty} \sum_{i=0}^{n-1} r(s_i, a_i)$ *provided the limit exists) and its average reward is* $arw(\sigma) = (1/n) \sum_{i=0}^{n-1} r(s_i, a_i)$ *(resp.* $arw(\sigma) = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} r(s_i, a_i)$ *provided the limit exists).*

We denote $SEQ^*(\mathcal{M})$ (resp. $SEQ^\infty(\mathcal{M})$) the set of finite (resp. infinite) sequences. A strategy st is a mapping from $SEQ^*(\mathcal{M})$ to A fulfilling $st(s_0 a_0 \dots s_n) \in A_{s_n}$. Hence a strategy discards non determinism, and the behavior of $\mathcal{M}$ w.r.t. st is a stochastic process $\mathcal{M}^{st}$ defined as follows: assume that the current execution is $s_0 a_0 \dots s_n$ then $a_n = st(s_0 a_0 \dots s_n)$ and the next state s_{n+1} is randomly chosen w.r.t. distribution $p(\cdot|s_n, a_n)$. Consequently, the reward of a random sequence of $\mathcal{M}^{st}$ is a random variable and the main problem in the MDP framework is to maximize or minimize the mean of this random variable by choosing an appropriate strategy allowing to obtain this maximum/minimum when it exists. In this case we can define the optimal total reward as follows:

$$\mathcal{V}_n^*(s) = \max_{a \in A_s}((r(s,a) + \sum_{s' \in S} p(s'|s,a) \cdot \max_{a' \in A_{s'}} \mathcal{V}_{n-1}(s', a'))$$

$$(resp. \lim_{n\to\infty} \mathcal{V}_n^*(s) \; provided \; the \; limit \; exists)$$

In the framework of finite MDPs, efficient solution techniques have been developed like *value iteration*, *policy iteration*, *modified policy iteration* and *linear programming* [15].

As an example the MDP representing the system introduced at the beginning of this section is shown in Fig. 1.1.

Table 1.1 shows the MDP states and the MDP actions possible in every state,while table 1.2 shows the possible transition probabilities.

Finally the reward of the model depending on the current state and on the action selected can be defined as follows:

$$r(s,a) = C_{repair}(a) + C_{penalty}(s)$$

This means that if the system is in the states 0 or 1 and the decision maker selects the action $AssignRes$ then the maintainer will have to pay C_{down} for the current state and $C_{AssignRes}$ for the future restore operation.

The four possible strategies[2] are shown in table 1.3. It is important to observe that the possible strategies are actually only three because when we select the strategies $\langle NoAssignRes, AssignRes\rangle$ and $\langle NoAssignRes, NoAssignRes\rangle$

[2]Only in the states 0 and 1 a choice between the possible actions can be made. In all the other states we have only one possible action

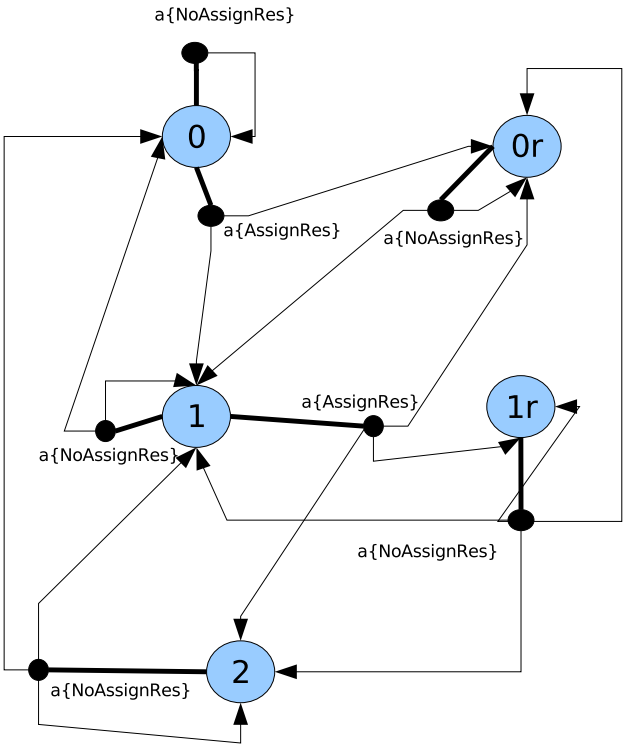


Figure 1.1: Symbolic representation of the MDP modeling the system with two identical components that can fail and be repaired

State	Description	Actions
0	all components are down	AssignRes,NoAssignRes
$0r$	a component is down while a component is on repair	NoAssignRes
1	a component is up while the other is down	AssignRes,NoAssignRes
$1r$	a component is up while the other is on repair on it	NoAssignRes
2	all components are up	NoAssignRes

Table 1.1: MDP states description and the MDP actions for state

Transition Probabilities	Value
$p_t(0\|0, NoAssignRes)$	1
$p_t(0r\|0, AssignRes)$	$1 - p_{repair}$
$p_t(1\|0, AssignRes)$	p_{repair}
$p_t(0r\|0r, NoAssignRes)$	$1 - p_{repair}$
$p_t(1\|0r, NoAssignRes)$	p_{repair}
$p_t(0\|1, NoAssingRes)$	p_{fault}
$p_t(1\|1, NoAssignRes)$	$1 - p_{fault}$
$p_t(0r\|1, AssignRes)$	$p_{fault}(1 - p_{repair})$
$p_t(1\|1, AssignRes)$	$p_{fault}p_{repair}$
$p_t(1r\|1, AssignRes)$	$(1 - p_{fault})(1 - p_{repair})$
$p_t(2\|1, AssignRes)$	$(1 - p_{fault})p_{repair}$
$p_t(0r\|1r, NoAssignRes)$	$p_{fault}(1 - p_{repair})$
$p_t(1\|1r, NoAssignRes)$	$p_{fault}p_{repair}$
$p_t(1r\|1r, NoAssignRes)$	$(1 - p_{fault})(1 - p_{repair})$
$p_t(2\|1r, NoAssignRes)$	$(1 - p_{fault})p_{repair}$
$p_t(0\|2, NoAssignRes)$	$p_{fault}p_{fault}$
$p_t(1\|2, NoAssignRes)$	$2p_{fault}(1 - p_{fault})$
$p_t(2\|2, NoAssignRes)$	$(1 - p_{fault})(1 - p_{fault})$

Table 1.2: MDP transition probabilities

	State 1	
State 0	$\langle AssignRes, AssignRes\rangle$	$\langle AssignRes, NoAssignRes\rangle$
	$\langle NoAssignRes, AssignRes\rangle$	$\langle NoAssignRes, NoAssignRes\rangle$

Table 1.3: MDP strategies

we obtain the same average reward in steady state: the two strategies are equivalent because the system will eventually reach the state 0 and it will always stay in it independently from the action selected for the state 1.

It is easy to observe that the MDP generation of this small system is a rather easy task, but if the system has more components and resources with different behaviors, then its modeling at the MDP level can be very hard. For instance if we change the previous example considering the two different components (different fault probabilities and repair probabilities) the corresponding MDP is much more complicated (the number of states becomes 8 as shown in table 1.4). Table 1.4 shows the MDP states and its possible actions in every state for this case. Table 1.5 instead shows all the possible strategies (we have used the label *NoAssRes* for the actions pair NoAssignResComp$_1$, NoAssignResComp$_2$,*AssResComp*$_1$ for the actions pair AssignResComp$_1$, NoAssignResComp$_2$ and *AssResComp*$_2$ for the actions pair NoAssignResComp$_1$, AssignResComp$_2$).

This is the reason of the introduction of Markov Decision Petri Net formalism (MDPN); we want to hide part of this complexity using a higher-level model. In fact the MDPN formalism does not give at the specification level the detail of the states space while it is possible at the specification level to define a complex decision or complex probabilistic behavior as a composition of simpler decisions or probabilistic behaviors.

1.2 Markov Decision Petri Net formalism

A Markov Decision Petri Net is composted by two different parts:

- the probabilistic one;
- the non deterministic one.

so that it is possible to distinguish clearly the probabilistic behavior of the system from the non deterministic one.

This allows to design the two parts separately and then compose them automatically. In the rest of this section we will use the scheme in Fig. 1.2 in order to explain these two parts.

In the rest of this section we will use the scheme in Fig. 1.2 in order to explain these two parts.

The probabilistic part models the probabilistic behavior of the system and can be seen as composition of n (controllable/non-controllable) components,

State	Description	Actions
0	all components are down	AssignResComp_1, AssignResComp_2, NoAssingResComp_1, NoAssingResComp_2
$0r1$	the component 2 is down while the component 1 is on repair	NoAssingResComp_1, NoAssingResComp_2
$0r2$	the component 1 is down while the component 2 is on repair	NoAssingResComp_1, NoAssResingComp_2
$1up1$	the component 1 is up while the other is down	AssignResComp_1, AssignResComp_2, NoAssingResComp_1, NoAssingResComp_2
$1up2$	the component 2 is up while the other is down	AssignResComp_1, AssignResComp_2, NoAssingResComp_1, NoAssingResComp_2
$1r1$	the component 1 is up and the other is on repair	NoAssingResComp_1, NoAssingResComp_2
$1r2$	the component 2 is up and the other is on repair	NoAssingResComp_1, NoAssingResComp_2
2	all components are up	NoAssingResComp_1, NoAssingResComp_2

Table 1.4: States and actions for state of the MDP modeling a system with two different components

Strat.	**0**	**1up1**	**1up2**
1	NoAssRes	NoAssRes	NoAssRes
2	NoAssRes	NoAssRes	AssResComp_1
3	NoAssRes	NoAssRes	AssResComp_2
4	NoAssRes	AssResComp_1	NoAssRes
5	NoAssRes	AssResComp_1	AssResComp_1
6	NoAssRes	AssResComp_1	AssResComp_2
7	NoAssRes	AssResComp_2	NoAssRes
8	NoAssRes	AssResComp_2	AssResComp_1
9	NoAssRes	AssResComp_2	AssResComp_2
10	AssResComp_1	NoAssRes	NoAssRes
11	AssResComp_1	NoAssRes	AssResComp_1
12	AssResComp_1	NoAssRes	AssResComp_2
13	AssResComp_1	AssResComp_1	NoAssRes
14	AssResComp_1	AssResComp_1	AssResComp_1
15	AssResComp_1	AssResComp_1	AssResComp_2
16	AssResComp_1	AssResComp_2	NoAssRes
17	AssResComp_1	AssResComp_2	AssResComp_1
18	AssResComp_1	AssResComp_2	AssResComp_2
19	AssResComp_2	NoAssRes	NoAssRes
20	AssResComp_2	NoAssRes	AssResComp_1
21	AssResComp_2	NoAssRes	AssResComp_2
22	AssResComp_2	AssResComp_1	NoAssRes
23	AssResComp_2	AssResComp_1	AssResComp_1
24	AssResComp_2	AssResComp_1	AssResComp_2
25	AssResComp_2	AssResComp_2	NoAssRes
26	AssResComp_2	AssResComp_2	AssResComp_1
27	AssResComp_2	AssResComp_2	AssResComp_2

Table 1.5: All the strategies of the MDP with two *different* components, that can fail and be repaired.

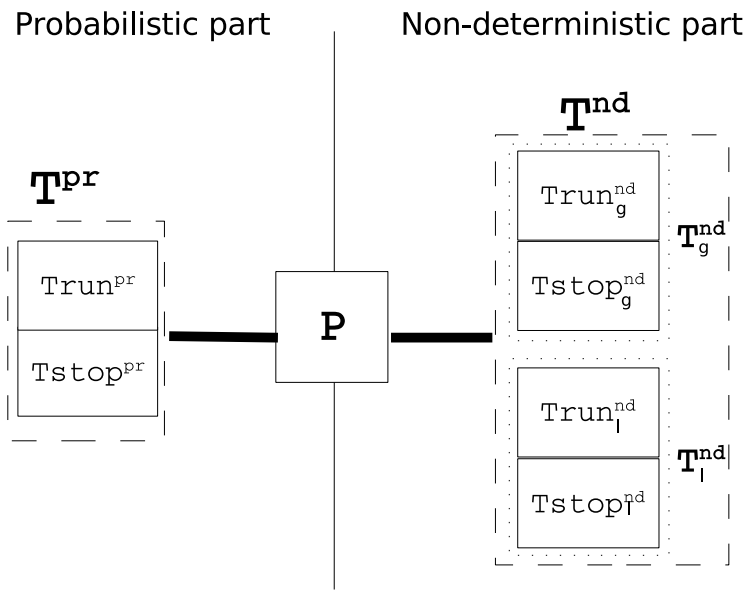


Figure 1.2: The MDPN schema

	Global decision	**Local decision**
Run	$Trun_g^{nd}$	$Trun_l^{nd}$
Stop	$Tstop_g^{nd}$	$Tstop_l^{nd}$

Table 1.6: Schema of the non deterministic transitions

that can interact; instead the non deterministic part models the non deterministic behavior of the system where the decisions must be taken (we shall call this part *the decision maker*). Hence the global system behavior can be described as an alternating sequence of probabilistic and non deterministic phases.

The probabilistic behavior of a component is characterized by two different types of actions/transitions $Trun^{pr}$ and $Tstop^{pr}$. The $Trun^{pr}$ transitions represent intermediate steps in a probabilistic behavior phase and can involve several components (synchronized in that transition), while the $Tstop^{pr}$ ones always represent the conclusion of the probabilistic phase of at least one component.

In the non deterministic part, the decisions can be defined at the system level (T_g^{nd}, *global decision*) or at the component level (T_l^{nd}, *local decision*). The sets T_g^{nd} and T_l^{nd} are again partitioned in $Trun_g^{nd}$ and $Tstop_g^{nd}$, and $Trun_l^{nd}$ and $Tstop_l^{nd}$. This is shown in table 1.6.

Like in the probabilistic part, a transition $t \in Trun^{nd}$ where $Trun^{nd} = Trun_g^{nd} \cup Trun_l^{nd}$, can never conclude the non deterministic phase of the sys-

tem or a component, while a transition $t \in Tstop^{nd}$ where $Tstop^{nd} = Tstop_g^{nd} \cup Tstop_l^{nd}$, always concludes the non deterministic phase of the global system or of a specific component.
It is important to introduce some additional terminology and assumptions: 1) a component that is subject to *local* non deterministic choice will be called *controllable* component, otherwise it will be called *non controllable* component; 2) a given probabilistic phase ends when all the components have finished their (current) probabilistic phase, while a given non deterministic phase ends when the decision maker has taken a decision at the global system level (if any is specified) and for every controllable component; 3) our model has a discrete time semantics; one time unit elapses (i.e. a new *decision epoch* starts) at each entrance in a non deterministic phase after having left a probabilistic phase.

The formal definition of the MDPN formalism follows.

Definition 2 (Markov Decision Petri Net (MDPN)) *A Markov Decision Petri Net (MDPN) is a tuple*

$$\mathcal{N}_{MDPN} = \langle Comp^{pr}, Comp^{nd}, N_{PN}^{pr}, N_{PN}^{nd} \rangle$$

where:

- $Comp^{pr}$ *is a finite non empty set of component identifiers;*
- $Comp^{nd} \subseteq (Comp^{pr} \cup \{id_s\})$ *is a finite non empty set of the controllable component identifiers plus the global system (id_s) identifier;*
- $N^{pr} = \langle P, T^{pr}, I^{pr}, O^{pr}, H^{pr}, prio^{pr}, weight^{pr}, act, m_0 \rangle$ *is a Petri Net with priorities and weights associated with the transitions, and with the additional function act.*
  The set T^{pr} is partitioned into two subsets: $T^{pr} = Trun^{pr} \uplus Tstop^{pr}$
  $act : T^{pr} \rightarrow 2^{Comp^{pr}}$ is a function that associates with every transition $t \in T^{pr}$ a set of components[3].
- $N^{nd} = \langle P, T^{nd}, I^{nd}, O^{nd}, H^{nd}, prio^{nd}, obj, m_0 \rangle$, *is a Petri Net with transition priorities and with the additional function obj.*
  The set of transitions is partitioned into two subsets: $T^{nd} = Trun^{nd} \uplus Tstop^{nd}$
  $obj : T^{nd} \rightarrow Comp^{nd}$ is a function that associates with every transition $t \in T^{nd}$ an element of $Comp^{nd}$ (This element is a controllable component, and it is the "object" to which the action (decision) represented by transition t refers).

Furthermore, the following constraints must be fulfilled:

- $T^{pr} \cap T^{nd} = \emptyset$;
- $\forall id \in Comp^{pr}, act^{-1}(\{id\}) \cap Tstop^{pr} \neq \emptyset$;

[3]These components are the "actors" that synchronize in transition t.

- $\forall id \in Comp^{nd}, obj^{-1}(\{id\}) \cap Tstop^{nd} \neq \emptyset$;

In the rest of the report we will call N^{pr} the **probabilistic part** and N^{nd} the **decision maker**.

Observations: 1) all the components and the decision maker share the same set of places; 2) a transition $t \in T^{pr}$ can be used to synchronize two or more components; 3) we cannot have a component without probabilistic behavior.

Let us now introduce the rewards associated with the MDPN net; two types of reward functions are possible: the state reward and the transition reward. These are then combined through a third function to obtain a global reward.

Definition 3 (MDPN reward functions)

- $rs : \mathbb{N}^P \rightarrow \mathbb{R}$ *is a function defining for every net marking its reward value.*
- $rt : T^{nd} \rightarrow \mathbb{R}$ *is a function defining for every transition its reward value.*
- $r_g : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ *is a function which is not decreasing w.r.t its second parameter (the reason is to allow efficient analysis and will be clarified later in this section), and which is needed to obtain a global reward function for the derived MDP; the first parameter is a state reward, while the second is a reward associated with a non deterministic (complex) action.*

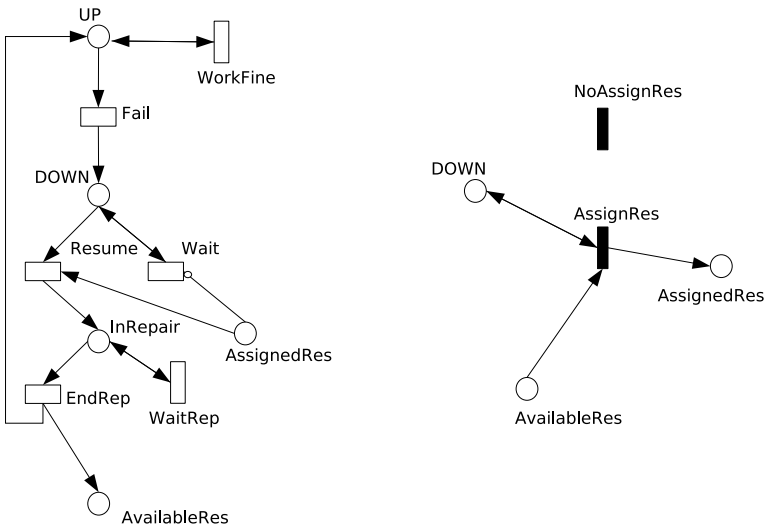
An example of probabilistic and non deterministic subnets is shown in Fig. 1.3. Fig. 1.3 in the left part shows the probabilistic behavior of a component which can work fine or fail; while Fig. 1.3 in the right part shows how the decision maker implements the possible ways of assigning the resources to a component (e.g. for repairing a component).

The $Tstop^{pr}$ transitions are *WorkFine*, *Fail*, *Wait*, *EndRep*, while the only $Trun^{pr}$ transition is *Resume* The non deterministic transitions are all $Tstop^{nd}$ transitions. This simple model can be easily complicated with more components in a modular way as shown in Fig. 1.11.

1.2.1 MDPN semantics

The MDPN semantics is given in two steps: the first step defines how to compose the probabilistic part and the decision maker and to derive from such composition a unique PN. The second step consists in generating the (finite) RG of the PN obtained in the first step and then in deriving an MDP from it.

From MDPN to PN: before describing how to compose the probabilistic part with the decision maker, we explain the semantics of the additional places $Stop_i^{pr}$, Run_i^{pr}, $Stop_i^{nd}$, Run_i^{nd}, $Stop_0^{nd}$ and $Run_0^{nd}$and the additional non deterministic transitions $PrtoNd$ and $NdtoPr$, that will be introduced during the composition phase.



$Trun^{pr} = \{Resume\}$, while all the other probabilistic transitions belong to $Tstop^{pr}$. All the non deterministic transition belong to $Tstop^{nd}$.

Figure 1.3: An example MDPN with only one (controllable) component.

Places $Stop_i^{pr}$, Run_i^{pr}, $Stop_i^{nd}$, Run_i^{nd}, $Stop_0^{nd}$ and Run_0^{nd} will be used to regulate the interaction among the components, the global system and the decision maker. It is important to observe that we have a place Run_i^{pr}, $Stop_i^{pr}$ for every component where i identifies the component, while we insert the places Run_0^{nd} and $Stop_0^{nd}$ if the decision maker takes same global decision and the pair of places Run_i^{nd} and $Stop_i^{nd}$ for every controllable component $i \in Comp^{nd}$.

The non deterministic transitions $PrtoNd$ and $NdtoPr$ are used to ensure that the decision maker takes a decision for every component in every time unit: the former triggers a non deterministic phase when all the components have finished their probabilistic phase, the letter triggers a probabilistic phase when the decision maker has finished the non deterministic phase.

Fig. 1.4 shows how these elements are connected together.

The formal definition of the PN that is built from the components submodel, the decision maker and the additional places and transitions described above follows.

$$N^{comp} = \langle P^{comp}, T^{comp}, I^{comp}, O^{comp}, H^{comp}, prio^{comp}, weight^{comp}, m_0 \rangle,$$

where:

Places $P^{comp} = P \cup_{i \in Comp_{pr}} \{Run_i^{pr}, Stop_i^{pr}\} \cup_{i \in Comp_{nd}} \{Run_i^{nd}, Stop_i^{nd}\}$

Transitions $T^{comp} = T^{pr} \cup T^{nd} \cup \{PrtoNd, NdtoPr\}$

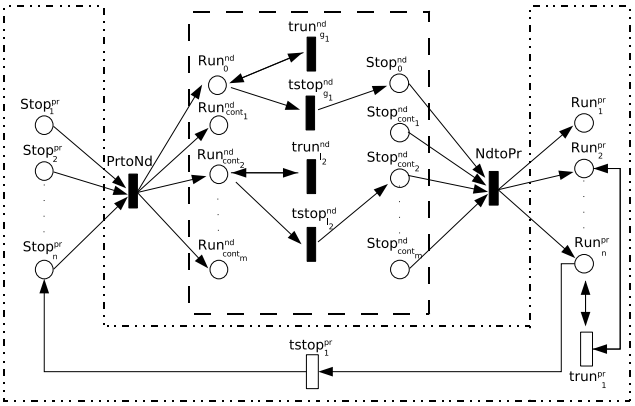


Figure 1.4: Arcs connecting the places $Stop_i^{pr}$, Run_i^{nd} and the transition $PrtoNd$; arcs connecting the places $Stop_i^{nd}$, Run_i^{nd} and the transition $NdtoPr$

priorities $\forall t \in T^{nd}, prio(t) = prio^{nd}(t)$
$\forall t \in T^{pr}, prio(t) = prio^{pr}(t)$
$prio(PrtoNd)$ =any natural number, $prio(NdtoPr)$ =any natural number, (actually these values are irrelevant, as we shall see later).

weights $\forall t \in T^{pr}, weight^{comp}(t) = weight^{pr}(t)$, for all others $t \in T^{comp}$ the weight is not defined.

I,O,H

- $\forall p \in P, t \in T^{nd} : I^{comp}(t,p) = I^{nd}(t,p), O^{comp}(t,p) = O^{nd}(t,p), H^{comp}(t,p) = H^{nd}(t,p)$
- $\forall p \in P, t \in T^{pr} : I^{comp}(t,p) = I^{pr}(t,p), O^{comp}(t,p) = O^{pr}(t,p), H^{comp}(t,p) = H^{pr}(t,p)$
- $\forall t \in T^{pr} s.t. i \in act(t) : I^{comp}(t, Run_i^{pr}) = 1$
- $\forall t \in Tstop^{pr}\ s.t. i \in act(t) : O^{comp}(t, Stop_i^{pr}) = 1$
- $\forall t \in Trun^{pr}\ s.t. i \in act(t) : O^{comp}(t, Run_i^{pr}) = 1$
- $\forall t \in T^{nd} s.t. i \in act(t) : I^{comp}(t, Run_i^{nd}) = 1$
- $\forall t \in Tstop^{nd}\ s.t. i \in act(t) : O^{comp}(t, Stop_i^{nd}) = 1$
- $\forall t \in Trun^{nd}\ s.t. i \in act(t) : O^{comp}(t, Run_i^{nd}) = 1$
- $\forall i \in Comp_{pr} : I^{comp}(PrtoNd, Stop_i^{pr}) = 1$
- $\forall i \in Comp_{nd} : O^{comp}(PrtoNd, Run_i^{nd}) = 1$
- $\forall i \in Comp_{pr} : O^{comp}(NdtoPr, Run_i^{pr}) = 1$
- $\forall i \in Comp_{nd} : I^{comp}(NdtoPr, Stop_i^{nd}) = 1$
- for all the other pairs t, p not mentioned in the above definition of I, O and H: $I^{comp}(t,p) = 0, O^{comp}(t,p) = 0, H^{comp}(t,p) = 0$;

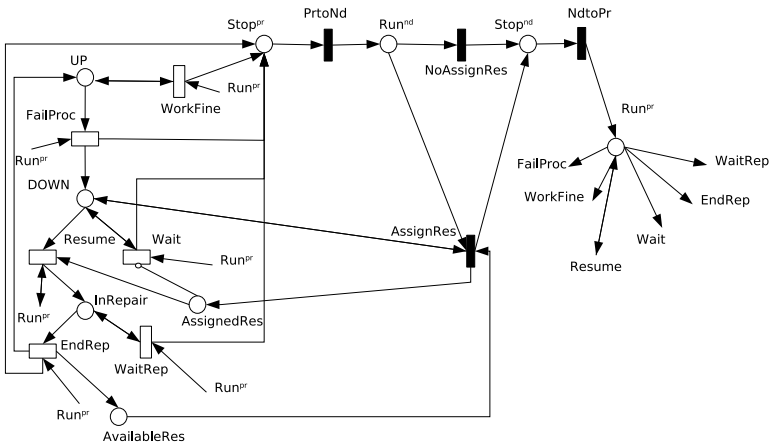


Figure 1.5: An example of PN obtained from the composition of the component and the decision maker in Fig. 1.3

the initial marking m_0 is equal to that of (both) the PN N^{nd} and N^{pr} $\forall p \in P$, $\forall i \in Comp_{nd} : m_0(Run_i^{nd}) = 1$; , $m_0(Stop_i^{nd}) = 0$; $\forall i \in Comp_{nd} : m_0(Run_i^{pr}) = 0$; $m_0(Stop_i^{pr}) = 0$;

Fig. 1.5 shows the PN obtained from the composition of the component and the decision maker in Fig. 1.3

RG semantics and transitions sequence reward Considering the RG obtained from the composed PN we observe that the reachability set (RS) can be partitioned into two subsets: the non deterministic states (RS_{nd}), in which only non deterministic transitions are enabled, and the probabilistic states (RS_{pr}), in which only probabilistic transitions are enabled. Notice that by construction it is not possible to have both nondeterministic and probabilistic transitions enabled in any marking of the PN obtained from an MDPN: in fact a probabilistic transition can be enabled only if there is at least one place Run_i^{pr} with marking $m(Run_i^{pr}) > 0$, while a non deterministic transition can be enabled only if there is at least one place Run_i^{nd} with marking $m(Run_i^{nd}) > 0$. Initially only the Run_i^{nd} places are marked; only when all the tokens in the Run_i^{nd} places have migrated to the $Stop_i^{nd}$ places (through the firing of some transition in $Tstop^{nd}$), the transition $NdtoPr$ can fire, removing all tokens from the $Stop_i^{nd}$ places and putting one token in each Run_i^{pr} place. Similarly, transition $PrtoNd$ is enabled only when all tokens have moved from the Run_i^{pr} to the $Stop_i^{pr}$ places; the firing of $PrtoNd$ brings the tokens back in each Run_i^{nd} place. It is thus clear that it can never be the case that a place Run_i^{pr} and a place Run_i^{nd} are simultaneously marked.

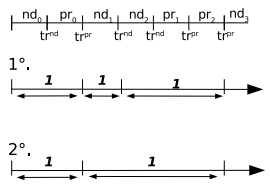


Figure 1.6: An example showing the two different time semantics of the evolution produced by an MDPN

Observe that any *path* in the RG can be partitioned into (maximal) sub-paths traversing only states of the same type, so that each path can be described as an alternating sequence of non deterministic and probabilistic sub-paths. Each probabilistic sub-path can be substituted by a single "complex" probabilistic step and assigned a probability based on the weights of the transitions firing along the path. The nondeterministic sub-paths can be interpreted according to two alternative semantics: the first one considers each non deterministic marking in the path (and thus in the RG) as an observable state, that is a state where the system spends one time unit (in other words a state where a new decision epoch starts); the second semantics considers a path through non deterministic states as a single complex action and the only state where time is spent is the first one in the sequence (that is the state that triggers the "complex" decision multi-step).

In the former case, every non deterministic state in the RG will translate in a MDP state while in the latter case, the non deterministic paths will be substituted by "complex" actions in the MDP to be generated, and only the first state in each path will appear as a state in the MDP (the others states in the path are *vanishing*, borrowing the terminology from the literature on GSPN / SWN).

Fig. 1.6 illustrates through one example the two different timing semantics applied to the same path.

In the rest of paper we decide to adopt the second semantics since it easily simulates the first one, while the converse simulation remains an open issue.

For instance to simulate the first semantics having chosen the second one we only need to add a dummy probabilistic step after each non deterministic transition firing, that does not change the global state but serves only as a time advance probabilistic action (that brings back the control to the non deterministic part with probability one). In our MDWN framework this could be achieved by introducing a dummy time advance subnet activated after each firing of a T^{nd} transition with no effect on the overall state. The subnet could contain a single place Pdummy that should inhibit all the transitions in T^{nd}, and a single probabilistic transition Tdummy, that should have Pdummy as single input place and no output places.

Let us now define the reward function for a sequence of non deterministic transitions, $\sigma \in (T^{nd})^*$; abusing notation we use the same name $rt()$ for the

reward function for single transitions and for transition sequences.

Definition 4 (The transition sequence reward $rt(\sigma)$) *The transition reward for a non deterministic transition sequence is defined as follows:*

$$rt(\sigma) = \sum_{t \in T^{nd}} rt(t)|\sigma|_t$$

where $|\sigma|_t$ is the number of occurrences of non deterministic transition t in σ

Observation: The above definition of $rt(\sigma)$ assumes that the firing order in such a sequence is irrelevant w.r.t. the reward. Otherwise stated, a sequence of non deterministic transitions *produces* a set of decisions, that do not need to be implemented in a defined order.

Generation of the MDP from the RG of the PN derived from the MDPN In this subsection we describe how to derive an MDP from the RG of the PN previously obtained.

The MDP can be obtained from the RG (shown in Fig. 1.7) of the PN model in two steps (as showing in Fig. 1.7):

- build from the RG the RG_{nd} where all the transition sequences passing only through non deterministic states are reduced to one non deterministic step.
- build the $RG_{MDP} \approx MDP$ from the RG_{nd} where all probabilistic paths are removed.

Before proceeding to the transformation of the RG into a MDP some requirements must be checked on it:

- in the probabilistic states part, it is required that there is not any terminal strongly connected component (*i.e.*, there is not any set of absorbing probabilistic states);
- it is required that the RG does not contain any deadlock state.

We do not require anything on the presence of loops in the RG because they will be managed by the Bellman and Ford algorithm as described later.

To efficiently derive the RG_{nd} from the RG it is necessary to decide if the solution of our problem must maximize or minimize the reward function. Then every sub-graph in the RG corresponding to a set of sequences of non deterministic states starting with the same non deterministic state and ending with a probabilistic state is substituted in the RG_{nd} by a new sub-graph obtained from the previous one connecting directly the first non deterministic state nd to every probabilistic state pr (in the initial sub-graph) with an arc labeled $\sigma_{nd,pr}$; where $\sigma_{nd,pr}$ is the non deterministic transitions sequence between nd and pr with the maximum/minimum rt . In the rest of paper we will call $\sigma_{nd,pr}$ a non

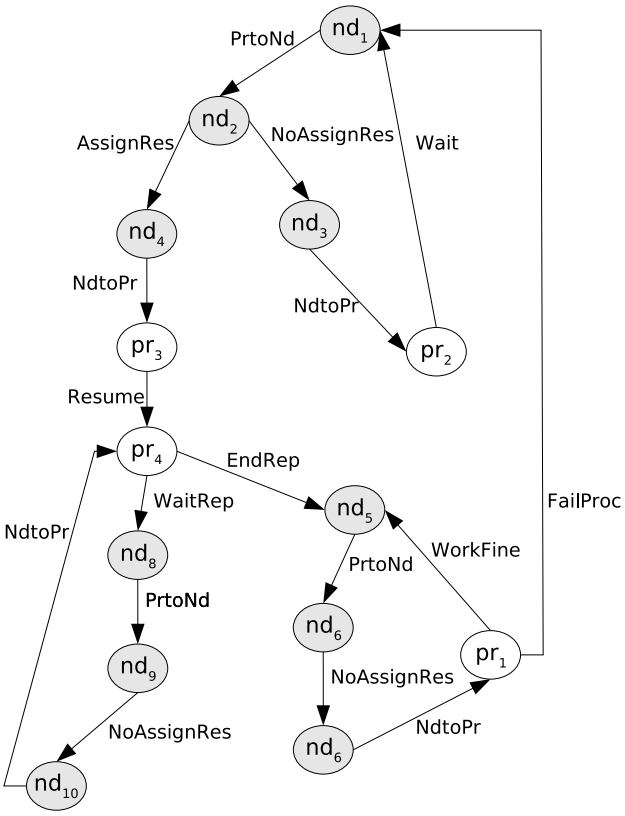


Figure 1.7: The RG of the PN model in Fig. 1.5

Figure 1.8: The steps from the RG to RG_{MDP}

Algorithm 1 function BellmanFord(list vertexes, list edges, vertex source)

```
1: for vertex v in vertexes do
2:    if v is source then
3:       v.distance := 0
4:    else
5:       v.distance := infinity
6:    v.predecessor := null
7: for i from 1 to size(vertexes) do
8:    for each edge uv in edges do
9:       u := uv.source
10:      v := uv.destination
11:      if v.distance > u.distance + uv.weight then
12:         v.distance := u.distance + uv.weight
13:         v.predecessor := u
14: for edge uv in edges do
15:   u := uv.source
16:   v := uv.destination
17:   if v.distance > u.distance + uv.weight then
18:      error "Graph contains a cycle"
```

deterministic macro-transition and T_{macro} will be the set containing all the non deterministic transitions sequence.

The minimum sequence is calculated using the Bellman and Ford algorithm for a single-source shortest paths in a weighted digraph (algorithm 1) where the transition reward corresponds to the cost function associated with the arcs, while the maximum is calculated using still the Bellman and Ford algorithm but the cost function must be set to the opposite of the transition reward. It is important to observe that if we want to use the Bellman and Ford algorithm in order to find the maximum/minimum path between nd and pr, $rt(\sigma)$ *must be defined as sum of transition rewards*(this motivates the definition of $rt(\sigma)$).

So far we have not discussed what happens in presence of non deterministic transition loops; first of all we must distinguish between two types of loops:

- negative loops where the reward function decreases;
- positive loops where the reward function increases.

This distinction is necessary because when we are interested to maximize the reward function the negative loops are not a problem, these paths will be not considered by the Bellman and Ford algorithm; while if the Bellman and Ford algorithm finds a positive loop the RG_{nd} cannot be obtained.

The same thing happens when we want to minimize the reward function, but in this case only the positive loops will create problems.

For example if we want to maximize the reward function then the non deterministic markings nd_2, nd_3, nd_4, nd_6, nd_7, nd_9 and nd_{10} will not appear in

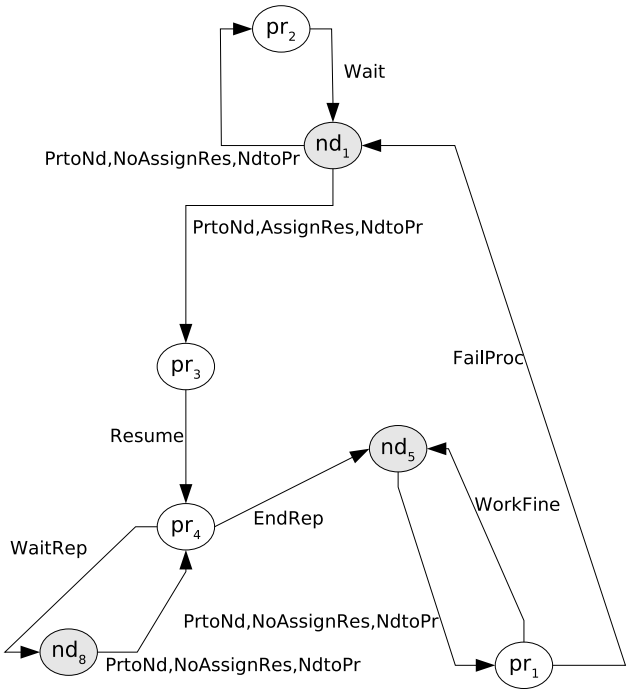


Figure 1.9: The RG_{nd} obtained from the RG in Fig. 1.7

the RG_{nd} obtained from the RG (Fig. 1.7). Therefore the path $\langle nd_1 \overset{PrtoNd}{\longrightarrow} nd_2 \overset{NoAssignRes}{\longrightarrow} nd_3 \overset{NdtoPr}{\longrightarrow} pr_2\rangle$ will be substituted by the path $\langle nd_1 \overset{\sigma_{nd_1,pr_2}}{\longrightarrow} pr_2\rangle$ where σ_{nd_1,pr_2} is the transitions sequence $\langle nd_1 \overset{PrtoNd}{\longrightarrow} nd_2 \overset{Noassign Res}{\longrightarrow} nd_3 \overset{NdtoPr}{\longrightarrow} pr_2\rangle$ with maximum rt. In the same way the path $\langle nd_1 \overset{PrtoNd}{\longrightarrow} nd_2 \overset{AssignRes}{\longrightarrow} nd_4 \overset{NdtoPr}{\longrightarrow} pr_3\rangle$ will become $\langle nd_1 \overset{\sigma_{nd_1,pr_3}}{\longrightarrow} pr_3\rangle$ where σ_{nd_1,pr_3} is the transitions sequence $\langle nd_1 \overset{PrtoNd}{\longrightarrow} nd_2 \overset{AssignRes}{\longrightarrow} nd_4 \overset{NdtoPr}{\longrightarrow} pr_3\rangle$ with maximum rt. It is important to observe that in this case there are not non deterministic sequences starting and ending with the same states; such that it is simpler to compute the minimum/maximum sequence.

The RG_{nd} for the example is shown Fig. 1.9.

The values of transitions firings sequence reward function for the non deter-

ministic macro-transitions in Fig. 1.9 are:

$$\begin{aligned} rt(\sigma_{nd_1,pr_2}) &= rt(PrtoNd) + rt(NoAssignRes) + rt(NdtoPr) \\ rt(\sigma_{nd_1,pr_3}) &= rt(PrtoNd) + rt(AssignRes) + rt(NdtoPr) \\ rt(\sigma_{nd_5,pr_1}) &= rt(PrtoNd) + rt(NoAssignRes) + rt(NdtoPr) \\ rt(\sigma_{nd_8,pr_4}) &= rt(PrtoNd) + rt(NoAssignRes) + rt(NdtoPr) \end{aligned}$$

It is important to observe that all the non deterministic transition firing sequences start with the transition $PrtoNd$ and end with the transition $NdtoPr$. These transitions have always $rt()$ equal to zero so that we can discard them in the following way:

$$\begin{aligned} rt(\sigma_{nd_1,pr_2}) &= rt(NoAssignRes) \\ rt(\sigma_{nd_1,pr_3}) &= rt(AssignRes) \\ rt(\sigma_{nd_5,pr_1}) &= rt(NoAssignRes) \\ rt(\sigma_{nd_8,pr_4}) &= rt(NoAssignRes) \end{aligned}$$

The final step takes in input the RG_{nd} and returns RG_{MDP}. First of all the probabilistic paths are reduced in the same way as the vanishing paths are reduced in the GSPN [14] then the transition probability function is computed.

For example the path $nd_5 \xrightarrow{\sigma_{nd_5,pr_1}} pr_1 \xrightarrow{Work} nd_5$ and $nd_5 \xrightarrow{\sigma_{nd_5,pr_1}} pr_1 \xrightarrow{FailProc} nd_1$ become $nd_5 \xrightarrow{\sigma_{nd_5,pr_1}} nd_5$ with probability $p(nd_5|nd_5, \sigma_{nd_5,pr_1}) = 1 - p_{fault}$ and $nd_5 \xrightarrow{\sigma_{nd_5,pr_1}} nd_1$ with probability $p(nd_1|nd_5, \sigma_{nd_5,pr_1}) = p_{fault}$.

Every path starting with a non deterministic state followed by a sequence of probabilistic states ending with a non deterministic state is substituted in the RG_{MDP} by a path containing only the two non deterministic states. These states will be connected by an arc labeled with the same label of the first non deterministic arc in such path, so that an MDP action corresponds to a non deterministic macro-transition.

In order to define the general way to compute the transition probability function of the MDP it is useful to introduce matrix $\mathbb{P}$.

Definition 5 (Transition probability matrix) *The matrix P represents the transition probability matrix and can be decomposed as follows:*

$$\mathbb{P} = (\sum_{n=0}^{\infty} (P^{(pr,pr)})^n \circ P^{(pr,nd)}) \tag{1.1}$$

$P^{(pr,pr)}$ represents the probability of moving from a probabilistic marking to a probabilistic marking pr without hitting any intermediate non deterministic marking and $P^{(pr,nd)}$ represents the probability of moving from probabilistic to non deterministic markings.

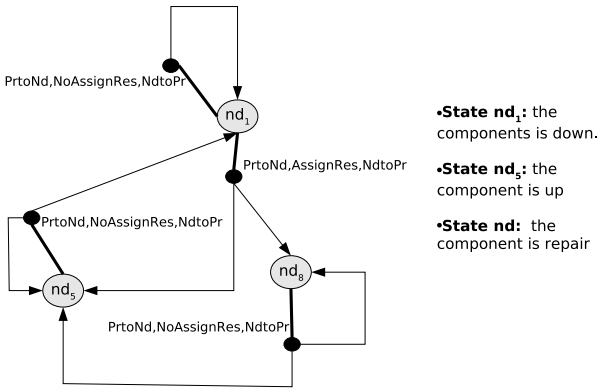


Figure 1.10: The MDP obtained from the MDPN in Fig. 1.3

In the computation of $\sum_{n=0}^{\infty}(P^{(pr,pr)})^n$, two possibilities may arise. The first corresponds to the situation in which there are no loops involving only probabilistic states. This means that for any probabilistic state $pr_i \in RS_{pr}$ there is a value $n_{0,i}$ such that any sequence of transition firings of length $n \geq n_{0,i}$ starting from such state must reach a non deterministic state $nd_j \in RS_{nd}$. In this case

$$\exists n_0 \ : \ \sum_{k=0}^{\infty}(P^{(pr,pr)})^k = \sum^{n_0})_{k=0}(P^{(pr,pr)})^k$$

The second corresponds to the situation in which there are possibilities of loops among probabilistic states, so that there is a possibility to remain trapped within a set of probabilistic states. In this case if there is at least a path that allows to exit from the loop arriving in a non deterministic state then:

$$\sum_{n=0}^{\infty}(P^{(pr,pr)})^n = [I - P^{(pr,pr)}]^{-1}$$

In conclusion equation (1.1) can be rewritten in this way:

$$I\!P = \begin{cases} (\sum_{k=0}^{n_0}(P^{(pr,pr)})^k) \circ P^{(pr,nd)} & if\ there\ are\ no\ loops \\ ([I - P^{(pr,pr)}]^{-1}) \circ P^{(pr,nd)} & if\ there\ are\ loops \end{cases}$$

from which we can conclude that the probability distribution $dist(nd, \sigma)$ with $nd \xrightarrow{\sigma} pr$ is given by the row vector $I\!P[pr]$.

The reward function for every action is instead computed by the following formula:

$$r(nd, \sigma) = r_g(rs(nd), rt(\sigma))$$

Probability	Value
$pr(nd_1 \mid nd_1, \sigma_{nd_1,pr_2})$	1
$pr(nd_8 \mid nd_1, \sigma_{nd_1,pr_3})$	p_{repair}
$pr(nd_5 \mid nd_1, \sigma_{nd_1,pr_3})$	$1 - p_{repair}$
$pr(nd_1 \mid nd_5, \sigma_{nd_5,pr_1})$	p_{fault}
$pr(nd_5 \mid nd_5, \sigma_{nd_5,pr_1})$	$1 - p_{fault}$
$pr(nd_8 \mid nd_8, \sigma_{nd_8,pr_4})$	p_{repair}
$pr(nd_5 \mid nd_8, \sigma_{nd_8,pr_4})$	$1 - p_{repair}$

Table 1.7: Transition probability function of the MDP in Fig. 1.10

Reward	Value
$r(nd_1, \sigma_{nd_1,pr_2})$	$r_\sigma(\sigma_{nd_1,pr_1}) + r_s(nd_1)$
$r(nd_1, \sigma_{nd_1,pr_3})$	$r_\sigma(\sigma_{nd_1,pr_3}) + r_s(nd_1)$
$r(nd_5, \sigma_{nd_5,pr_1})$	$r_\sigma(\sigma_{nd_5,pr_1}) + r_s(nd_5)$
$r(nd_8, \sigma_{nd_8,pr4})$	$r_\sigma(\sigma_{nd_8,pr4}) + r_s(nd_8)$

Table 1.8: Reward function of the MDP in Fig. 1.10

Now it should be clear why r_g must be not decreasing in its second parameter, in fact only under this condition we can discard some paths with lower reward when applying the Bellman and Ford algorithm in the derivation of the MDP from the MDPN Reachability Graph.

In Fig. 1.10 the graphical representation of the MDP obtained from the MDPN in Fig. 1.3 is shown and in tables 1.7 and 1.8 its transition probability function and its reward function are shown.

1.3 A more complex example

In this section we are going to show and solve an MDPN model of the system described in the section 1.1 but with two components and one resource. In Fig. 1.11 the probabilistic part and the non deterministic part are shown. It is important to remark that in this example the decision maker does not take any global decision since the decision process in this case can be decomposed into a set of decision local to the single components.

An example of the derived MDP assuming that we want to minimize the total reward is shown in Fig. 1.12 where:

- $\sigma_1 = \{NoAssigRes_2, NoAssigRes_1\}$;
- $\sigma_2 = \{NoAssigRes_2, AssignRes_1\}$;
- $\sigma_3 = \{AssignRes_2, NoAssigRes_1\}$;

and its states are described in table 1.10.

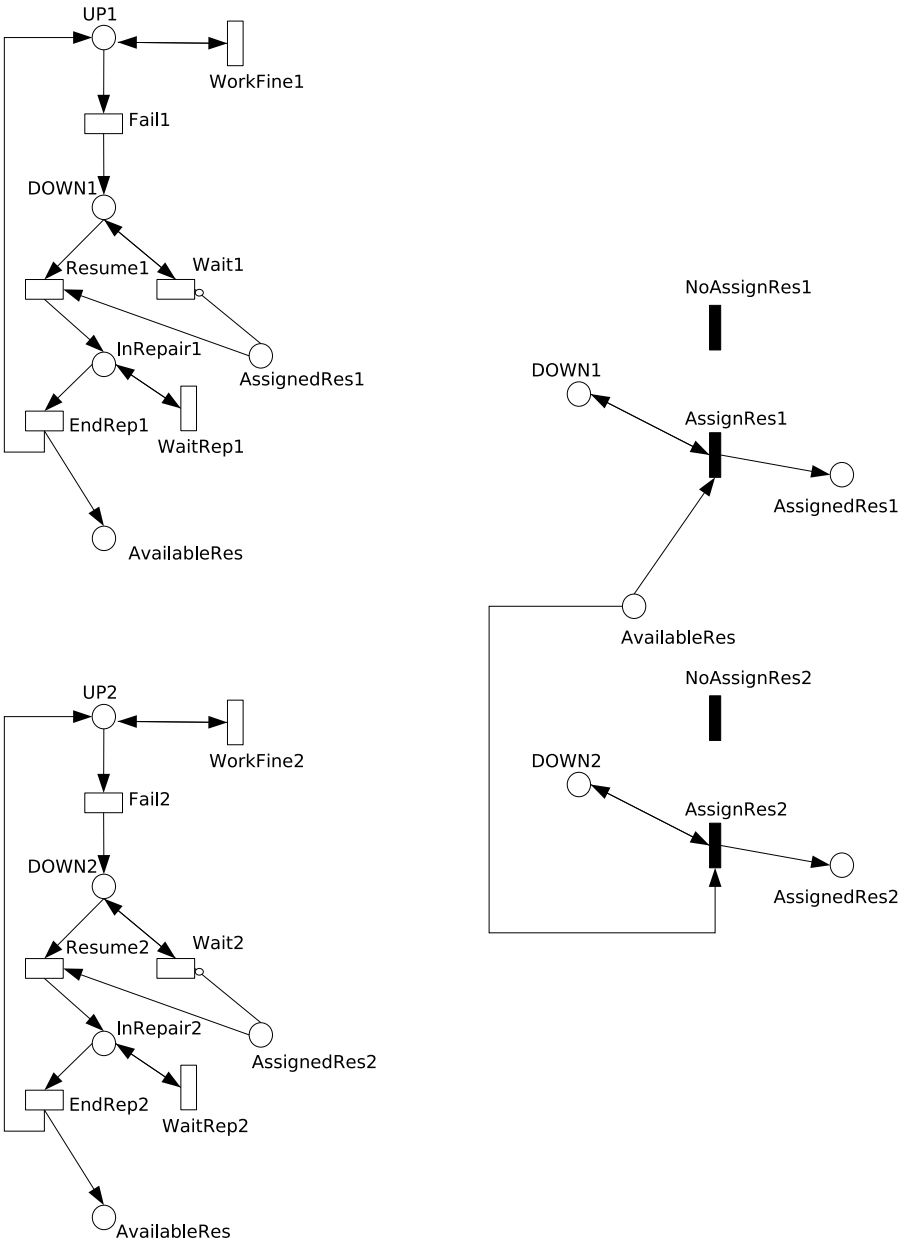


Figure 1.11: The probabilistic and non deterministic part

	|Probabilistic|	**|Non deterministic|**	**|Total|**
RG	32	48	80
RG_{nd}	32	8	40
RG_{MDP}	0	8	8

Table 1.9: States of the MDP in Fig. 1.12

ID	**Marking**
nd_1	$DOWN_1(1)Stop_1^{pr}(1)AvailableRes(1)UP_2(1)Stop_2^{pr}(1)$
nd_8	$DOWN_1(1)Stop_1^{pr}(1)AvailableRes(1)DOWN_2(1)Stop_2^{pr}(1)$
nd_{17}	$UP_1(1)Stop_1^{pr}(1)AvailableRes(1)UP_2(1)Stop_2^{pr}(1)$
nd_{22}	$UP_1(1)Stop_1^{pr}(1)AvailableRes(1)DOWN_2(1)Stop_2^{pr}(1)$
nd_{29}	$InRepair_1(1)Stop_1^{pr}(1)DOWN_2(1)Stop_2^{pr}(1)$
nd_{34}	$InRepair_1(1)Stop_1^{pr}(1)UP_2(1)Stop_2^{pr}(1)$
nd_{39}	$DOWN_1(1)Stop_1^{pr}(1)InRepair_2(1)Stop_2^{pr}(1)$
nd_{44}	$UP_1(1)Stop_1^{pr}(1)InRepair_2(1)Stop_2^{pr}(1)$

Table 1.10: States of the MDP in Fig. 1.12

The number of states of this MDP (table 1.9) is higher than that of the MDP shown in section 1.1, because in the MDPN formalism the components can not be considered indistinguishable; however if we merge together the states nd_{29} and nd_{39}, and the states nd_{34} and nd_{44} then we obtain exactly the MDP presented in section 1.1 (this is the reason for introducing the Markov Decision Well-formed net).

It is important to observe that the MDP derived in this section and the MDP presented in section 1.1 are equivalent and we can derive from them the same optimal strategies.

For example if we set:

- $p_{fault} = 0.3$;
- $p_{repair} = 0.6$;
- $C_{penalty} = 100$

then we can observe that the optimal strategy is:

- $\forall 0 \leq C_{repair} \leq \frac{17}{16} C_{penalty}$ we must repair a component every time that is down;
- $\forall \frac{17}{16} C_{penalty} < C_{repair} \leq \frac{13}{4} C_{penalty}$ we must repair only when all the components are down[4];
- $\forall C_{repair} > \frac{13}{4} C_{penalty}$ we must never repair a component.

These results are summarized in table 1.11.

[4]Only a component will be repaired at a time

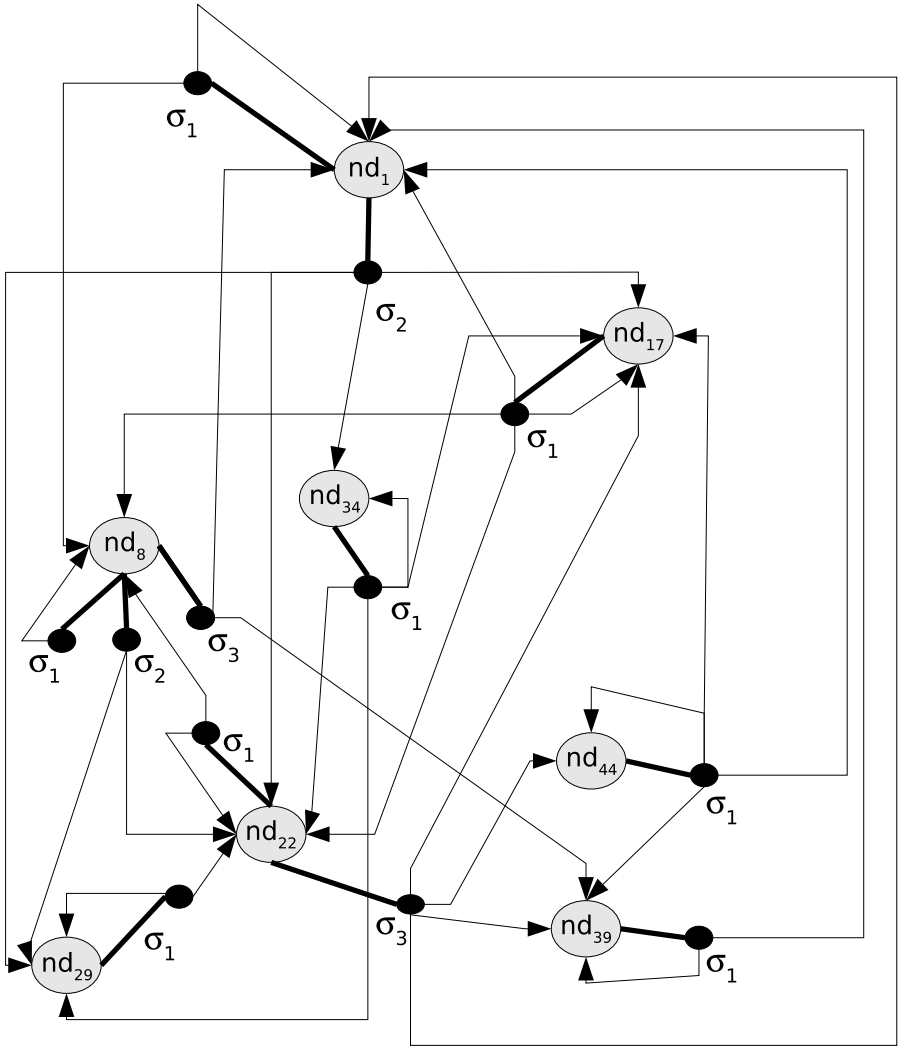


Figure 1.12: The derived MDP

C_{repair}	Optimal policy	Optimal policy value
0	nd_8:assign, nd_1, nd_{22}:assign	15.0685
50	nd_8:assign, nd_1, nd_{22}:assign	33.5616
70	nd_8:assign, nd_1, nd_{22}:assign	42.8082
87.5	nd_8:assign, nd_1, nd_{22}:assign	47.4315
100	nd_8:assign, nd_1, nd_{22}:assign	52.0548
106.25	nd_8:assign, nd_1, nd_{22}:assign	54.3664
112.5	nd_8:assign,nd_1, nd_{22}:not assign	55.8333
150	nd_8:assign, nd_1, nd_{22}:not assign	63.3333
200	nd_8:assign, nd_1, nd_{22}:not assign	73.3333
300	nd_8:assign, nd_1, nd_{22}:not assign	93.3333
325	nd_8:assign, nd_1, nd_{22}:not assign	98.3333
337.5	nd_8:not assign,nd_1, nd_{22}:not assign	100
350	nd_8:not assign, nd_1, nd_{22}:not assign	100

Table 1.11: Optimal strategies

1.4 Discussion

We have introduced a high level formalism for defining an MDP and we have described how starting from an MDPN model to automatically obtain the corresponding MDP. Unfortunately this formalism has a drawback: by definition, the components are identified and always distinguished in the state representation, even if they have similar behavior (*i.e.*, even if one component is an exact copy of another component). This can have an impact both at the level of the model description (which could become difficult to read when several components are present), and at the level of the state space size. A solution may be the use of a higher-level formalism for modeling the components and the decision maker nets. The Well-formed nets (WNs) formalism is a good candidate. First, it is a high level Petri Net formalism that allows to represent systems in a concise way thanks to the possibility of associating information with tokens and of parameterizing transition firings. Furthermore it is also the support of efficient analysis techniques that take into account the symmetries of the system.

Chapter 2

Markov Decision Well-formed net

In this chapter the Markov Decision Well-formed Net formalism is introduced in order to give an higher level formalism to express an the MDP. Section 2.1 describes the syntax and the semantics of the Markov Decision Well-formed Nets (MDWN). The unfolding algorithm, the RG and the SRG technique are also presented in this section.

Section 2.2 shows some experiments.

In this chapter we consider that the reader has already familiarity with the WN notation, in every case a short presentation on the WN is shown in appendix A.

2.1 Markov Decision Well-formed Net formalism

2.1.1 WN informal introduction

WNs are an high-level Petri net formalism whose syntax has been the starting point of several efficient analysis methods. Below, we describe the main features of WNs. The reader can refer to [3] and to the appendix A for a formal definition.

In a WN (and more generally in high-level Petri nets) a color domain is associated with places and transitions. The colors of a place label the tokens contained in this place, whereas the colors of a transition define different ways of firing it. In order to specify these firings, a color function is attached to every arc which, given a color of the transition connected to the arc, determines the number of colored tokens that will be added to or removed from the corresponding place. Finally the initial marking is defined by a multi-set of colored tokens in each place.

A color domain is a Cartesian product of color classes, this may be viewed as primitive domains. Classes can have an associated (circular) order expressed by

means of a successor function. The Cartesian product defining a color domain is possibly empty (e.g., for a place which contains neutral tokens) and may include repetitions (e.g., a transition which synchronizes two colors inside a class). A class can be divided into static subclasses. The colors within a class represent objects with the same nature (processes, resources, etc.), whereas the colors inside a static subclass have the same potential behavior (batch processes, interactive processes, etc.).

A color function is built by standard operations (linear combination, composition, etc.) on basic functions. There are three basic functions: a projection which selects an item of a tuple and is denoted by a typed variable (e.g., p, q); a synchronization/diffusion that is a constant function which returns the multiset composed by all the colors of a class or a subclass and is denoted S_{C_i} ($S_{C_{i,k}}$) where C_i ($C_{i,k}$) is the corresponding (sub)class; and a successor function which applies on an *ordered* class and returns the color following a given color.

Transitions and color functions can be guarded by expressions. An expression is a boolean combination of atomic predicates. An atomic predicate either identifies two variables $[p = q]$ or restricts the domain of a variable to a static subclass.

Examples of arc functions, transition guards, color domains can be seen in the MDWN model of Fig. 2.1 and Fig. 2.3. The details about the WN notation can be found in [3].

The constraints on the syntax of WN allow to automatically exploit the behavioral symmetries of the model and perform state-space based analysis on a more compact RG: the symbolic reachability graph (SRG). The SRG construction lies on the *symbolic marking* concept, namely a compact representation for a set of equivalent ordinary markings. A symbolic marking is a symbolic representation, where the actual identity of tokens is forgotten and only their distributions among places are stored. Tokens with the same distribution and belonging to the same static subclass are grouped into a so-called dynamic subclass. Starting from an initial symbolic marking, the SRG can be constructed automatically using a symbolic firing rule [3].

Various behavioral properties may be directly checked on the SRG. Furthermore, this construction leads also to efficient quantitative analysis, e.g. the performance evaluation of Stochastic WNs (SWNs) [3] (a SWN is obtained from a WN by associating an exponentially distributed delay with every transition, which may depend only on the static subclasses to which the firing colors belong).

2.1.2 Markov Decision Well-formed Net definition

A MDWN, like an MDPN, is composed by two distinct parts: the probabilistic one and the non deterministic one, and also in this case the set of transitions in each part is partitioned into $Trun$ and $Tstop$, as shown in Fig. 1.2. Each part of a MDWN is a WN model: the two parts share the same set of color classes. A MDWN comprises a special color class, say C_0, representing the system components: its cardinality $|C_0|$ gives the total number of compo-

nents in the system. This class can be partitioned into several static subclasses $C_0 = (\biguplus_{k=1}^{m} C_{0,k}) \uplus (\biguplus_{k=m+1}^{n_0} C_{0,k})$ such that colors belonging to different static subclasses represent components with different behavior and the first m static subclasses represent the controllable components while the others represent the non-controllable components.

Every transition in an MDWN has an associated color domain, defining its color instances. A color instance is expressed as a tuple of elements, each one with a type (a basic color set, possibly C_0). Some elements in a transition color instance have type C_0 and represent a component identifier (we call such elements the *component parameters*) and are used to specify the system components involved in the transition instance firing.

Definition 6 (Markov Decision Well-formed Net (MDWN)) *A Markov Decision Well-formed (MDWN) is a tuple*

$$\mathcal{N}_{MDWN} = \langle N_{WN}^{pr}, N_{WN}^{nd}, synctype, dyn, static \rangle,$$

where:

- $N_{WN}^{pr} = \langle P, T^{pr}, \mathcal{C}, cd^{pr}, I^{pr}, O^{pr}, H^{pr}, \phi, prio, weight^{pr}, m_0^{pr} \rangle$, *is a WN with weights associated with the transitions;*
- $N_{WN}^{nd} = \langle P, T^{nd}, \mathcal{C}, cd^{nd}, I^{nd}, O^{nd}, H^{nd}, \phi, prio, m_0^{nd} \rangle$, *is a WN;*
- $synctype : T^{pr} \cup T^{nd} \rightarrow \{Some, Allbut\}$ *is a function which associates with every transition a label, s.t.* $\forall t \in Tstop^{pr} \cup T^{nd} \Rightarrow synctype(t) = Some$. *This function is used to identify whether a transition involves only a given number of components or all components in the system except a given subset;*
- $dyn(t)$, *where* $t \in T^{pr} \cup T^{nd}$ *and* $cd(t) = \bigotimes_{i \in \{0,\ldots,n\}} C_i^{e_i}$, *is a subset of* $\{1, \ldots, e_0\}$*;*
- $static(t)$, *where* $t \in T^{pr} \cup T^{nd}$, *is a subset of* $\{1, \ldots, n_0\}$ *where* n_0 *represents the number of static subclasses in* C_0*;* $static(t)$ *is meaningful only if* $synctype(t) = Allbut$.

Furthermore, the following constraints must be fulfilled:

- $T^{pr} \cap T^{nd} = \emptyset$*;*
- $T^{pr} = Trun^{pr} \uplus Tstop^{pr} \wedge T^{nd} = Trun^{nd} \uplus Tstop^{nd}$*;*
- $\forall t \in T^{pr} \wedge synctype(t) = Some \Rightarrow dyn(t) \neq \emptyset$*;*
- $\forall t \in T^{pr} \wedge synctype(t) = Allbut : \forall i \in dyn(t), \phi(t) \Rightarrow (x_i \in \cup_{j \in static(t)} C_{0,j})$ *moreover* $\sum_{j \in static(t)} |C_{0,j}| > |dyn(t)|$, *where* x_i *represents the projection of the color of* t *on the i-th component parameter; this constraint is needed for consistency of transitions of type "Allbut" (the subset of* not *involved component cannot exceed the size of the subclass they belong to);*

- $\forall t \in T^{nd} \Rightarrow 0 \leq |dyn(t)| \leq 1$; *this is needed to ensure that each "stop" non deterministic transition involves only one (controllable) component;*

- $\forall C_{0,k}$ *there exists at least one transition* $t \in Tstop^{pr}$ *such that: either* $synctype(t) = Some \wedge \exists i \in dyn(t)$ *such that* $\phi(t) \Rightarrow x_i \in C_{0,k}$ *or* $synctype(t) = Allbut \wedge C_{0,k} \in static(t)$, *moreover if* $k \leq m$ *there exists a similar* $t \in Tstop^{nd}$.

- *all component parameters of any transition must be assigned different elements from* C_0 *in each transition instance: this should be enforced by the transition guard.*

In the rest of the report we will call N^{pr}_{WN} the **probabilistic part** and N^{nd}_{WN} the **decision maker**, that is the **non deterministic part**.

Now we introduce the rewards associated to the MDWN net; two types of reward functions are possible: the place reward and the transition reward.

Before introducing the place reward we must define the set $\widetilde{C}$.

Definition 7 ($\widetilde{\mathcal{C}}$) $\widetilde{\mathcal{C}}$ *is the set of sets* $\{\widetilde{C}_i\}_{i \in I}$ *with* $I = \{0, \ldots, n\}$. *While* $\widetilde{C}_i$ *is the set* $\{1, \ldots, n_i\}$ *where* n_i *is the number of static subclasses in* C_i.

We can always map the color class $\mathcal{C}$ on the set $\widetilde{\mathcal{C}}$ such that the definition of the $\widetilde{cd}$ function immediately follows:

Definition 8 ($\widetilde{cd}$) *The function* $\widetilde{cd}(p)$ *is defined as follows:*

$$\widetilde{cd} \stackrel{def}{=} \widetilde{(\bigotimes_{i \in I} C_i^{e_i})} = \bigotimes_{i \in I} \widetilde{C}_i^{e_i}$$

For instance if $C_0 = C_{0,1} \cup C_{0,1} \cup C_{0,3}$ where $C_{0,1} = \{comp_1, comp_2\}, C_{0,2} = \{comp_3\}, C_{0,3} = \{comp_4\}$, then $\tilde{C}_0 = \{C_{0,1}, C_{0,1}, C_{0,3}\}$, and if $p \in P$ with $cd(p) = C_0 \times C_0 \times C_0$ then $\widetilde{cd}(p) = \widetilde{C}_0 \times \widetilde{C}_0 \times \widetilde{C}_0$, moreover if $c = \langle comp_1, comp_2, comp_3 \rangle \in cd(p)$ then $\widetilde{c} = \langle 1, 1, 2 \rangle \in \widetilde{cd}(p)$.

It is important to observe that an unique $\widetilde{c}$ corresponds to every c.

Definition 9 (MDWN reward functions)

- $rs : \bigotimes_{p \in P} I\!N^{\widetilde{cd}(p)} \rightarrow I\!R$ *is a function which returns for every colored marking a reward value.*

- $\forall t \in T^{nd}, rt[t] : cd(t) \rightarrow I\!R$ *is a vector which associates with every transition a function defining the reward value of its instances; two instances may be assigned a different reward value only if there exists a standard predicate capable to distinguish the two.*

- $r_g : I\!R \times I\!R \rightarrow I\!R$ *is defined as in MDPN.*

An example of MDWN is shown in Figs. 2.1 and 2.2. In this model we are assuming that there are several instances of three component types: Proc, Mem and ResCtr (grouped in banks, each with one instance of each component): rather than replicating the same subnet several times, we use colored tokens to represent several instances on the same net structure. Instead there is only one instance of on other component the global memory. Class C_0 comprises four static subclasses $\{P, M, GM, R\}$, one for each component type. The cardinality of the processor (P), local memory (M) and resource control (R) subclasses corresponds to the number of banks in the system, while the cardinality of the global memory subclass is one. Arcs in Figs. 2.1 and 2.2 are annotated with very functions (tuples of projections) and all the variables appearing in the functions in this example are component parameters. The guards on the arcs include a term in the form $d(x) = CompType$ to force parameter x to range within static subclass $CompType$. The additional terms ϕ_{xyz}, ϕ_{xz}, ϕ_{yz} are not detailed here, but are used to associate components in the same bank: in fact the probabilistic part of the model must correctly synchronize components of type Proc, Mem and ResCtr belonging to the same bank (the model represents a situation where only one resource is assigned to each bank at a time, and it can be used to resume all failed components in the bank).

2.1.3 MDWN semantics

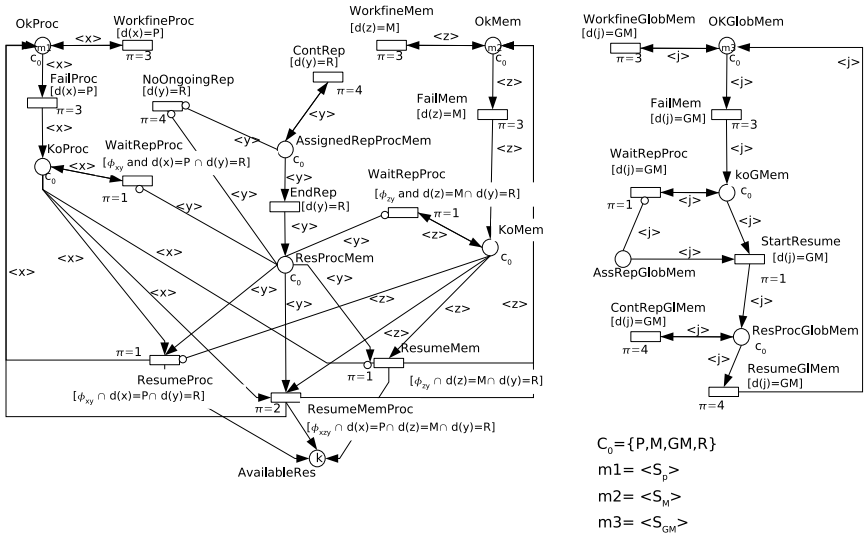
In this section we are going to describe how it is possible to obtain from an MDWN model the corresponding MDP model. The two possible methods are shown in Fig. 1

The first method requires to unfold the MDWN in order to obtain an equivalent MDPN and to derive from this an MDP, but this is not very efficient in fact it will multiply the number of places, transitions and arcs, moreover if the number of components is high the cost for computing the results will be high.

Instead the second method derives directly from an MDWN model an MDP. This second method can be decomposed in two steps: the first step defines how to compose the probabilistic part and the decision maker and to derive from such composition a unique WN. The second step consists in generating the (finite) RG of the WN obtained in the first step and then in deriving an MDP from it. In this way there is no need to produce the intermediate, potentially huge MDPN, but the cost for computing the result can still be high. In the last part of this section we show how the properties of WN can be extended to MDWN so that a smaller MDP can be directly derived from the Symbolic Reachability Graph of the corresponding WN, gaining in solution efficiency.

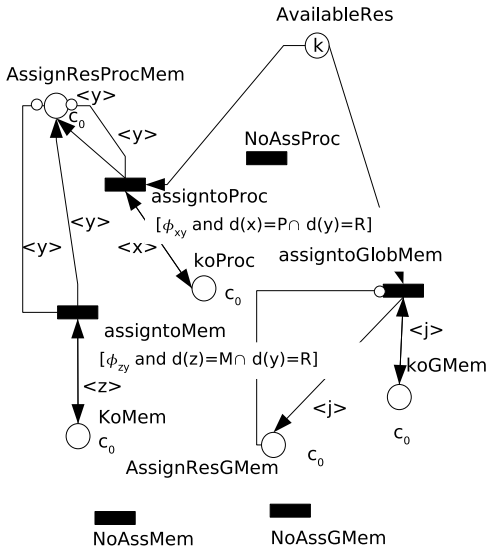
From MDWN to MDPN Given an MDWN with a finite color domains, it is always possible to derive an equivalent MDPN applying an unfolding algorithm[1]

[1]It is important to observe that while the unfolding of a MDWN is unique, the inverse operation of folding a MDPN to obtain a more compact, colored representation of the same model, may lead to several alternative MDWN models, depending on the point of view of the folding and the desired degree of compacting.



$Trun^{pr} = \{FailProc, FailMem, EndRep, StartResume\}$ all other transitions belong to $Tstop^{pr}$; all variables are component parameters. Transition priorities are denoted $\pi = prio(t)$ in the figure. P, M, GM and R are the Proc, Mem, GlobMem and ResCtr subclasses respectively.

Figure 2.1: Probabilistic part of the multiprocessor system MDWN model



all the non deterministic transition belong to $Tstop^{nd}$. P, M, GM and R are the Proc, Mem, GlobMem and ResCtr subclasses respectively.

Figure 2.2: Non deterministic part of the multiprocessor system MDWN model

Definition 10 (Unfolding of MDWN) *The MDPN resulting from the unfolding of a MDWN is defined as follows:*

- $Comp^{pr}$ *is the color class* C_0*;*
- $Comp^{nd}$ *is the set* $(\biguplus_{i=0}^{m} C_{0,i}) \cup \{id_s\}$*;*
- $N_{PN}^{pr} = \langle P', T^{pr\prime}, I^{pr\prime}, O^{pr\prime}, H^{pr\prime}, prio^{pr\prime}, weight^{pr\prime}, Comp, m_0' \rangle$ *where:*
  - $P' = \{\langle p, c\rangle, p \in P, c \in cd(p)\}$*;*
  - $T^{pr\prime} = \{\langle t, c\rangle, t \in T^{pr}, c \in cd_\phi(t)\}$*;*
  - $I^{pr\prime}[\langle p, c\rangle \langle t, c'\rangle] = (I^{pr}[p, t](c'))[c]$*;*
  - $O^{pr\prime}[\langle p, c\rangle \langle t, c'\rangle] = (O^{pr}[p, t](c'))[c]$*;*
  - $H^{pr\prime}[\langle p, c\rangle \langle t, c'\rangle] = (H^{pr}[p, t](c'))[c]$*;*
  - $prio^{pr\prime}[\langle t, c\rangle] = prio^{pr}[t](c)$*;*
  - $weight^{pr\prime}[\langle t, c\rangle] = weight^{pr}[t](c)$*;*
  - $act'(\langle t, c\rangle)$*:*
    - $\forall t \in T^{pr}, synctype(t) = Some \Rightarrow \bigcup_{i \in dyn(t)} c_{0,i}$*;*
    - $\forall t \in T^{pr}, synctype(t) = Allbut \Rightarrow \bigcup_{j \in static(t)} C_{0,j} \backslash (\biguplus_{i \in dyn(t)} c_{0,i})$*; where* $c_{0,i}$ *denotes the projection of color* c *on the* i*-th occurrence of* C_0 *in* $cd(t)$
  - $m_0^{pr\prime}[\langle p, c\rangle] = m_0^{pr}[p, c]$.
- $N_{PN}^{nd} = \langle P', T^{nd\prime}, I^{nd\prime}, O^{nd\prime}, H^{nd\prime}, prio^{nd\prime}, weight^{nd\prime}, Comp, m_0' \rangle$ *where:*
  - $T^{nd\prime} = \{\langle t, c\rangle, t \in T^{nd}, c \in cd(t)\}$*;*
  - $I^{nd\prime}[\langle p, c\rangle \langle t, c'\rangle] = (I^{nd}[p, t](c'))[c]$*;*
  - $O^{nd\prime}[\langle p, c\rangle \langle t, c'\rangle] = (O^{nd}[p, t](c'))[c]$*;*
  - $H^{nd\prime}[\langle p, c\rangle \langle t, c'\rangle] = (H^{nd}[p, t](c'))[c]$*;*
  - $prio^{nd\prime}[\langle t, c\rangle] = prio^{pr}[t](c)$*;*
  - $weight^{nd\prime}[\langle t, c\rangle] = weight^{nd}[t](c)$*;*
  - $obj(\langle t, c\rangle)$*:*
    - $\forall t \in T^{nd} \Rightarrow \bigcup_{i \in dyn(t)} \{c_{0,i}\}$*;*

The reward functions r_m' *and* r_t' *of the unfolded MDPN are defined as follows:*

- $rs'[\langle p, c\rangle] = rs[p](\widetilde{c})$*;*
- $rt'[\langle t, c\rangle] = rt[t](c)$*;*
- $r_g' = r_g$.

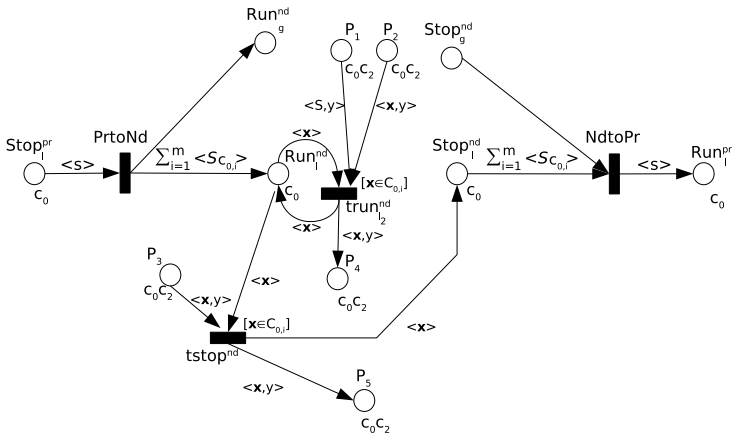


Figure 2.3: arcs connecting places $Stop^{pr}$, Run_l^{nd}, Run_g^{nd}, and transition $PrtoNd$ and their functions; arcs connecting places $Stop_l^{nd}$, $Stop_g^{nd}$, Run_l^{nd} and transition $NdtoPr$ and their function; example of connection of the decision maker to places Run^nd and $Stop^nd$: component parameters are highlighted in boldface in the arc functions.

From MDWN to composed WN Before describing the second method we must explain the use of the places $Stop_l^{pr}$, Run_l^{pr}, $Stop_l^{nd}$, Run_l^{nd}, $Stop_g^{nd}$, Run_g^{nd} and the non deterministic transitions $PrtoNd$ and $NdtoPr$, that are introduced during the composition phase.

The places $Stop^{pr}$, Run^{pr}, $Stop_l^{nd}$, Run_l^{nd}, $Stop_g^{nd}$ and Run_g^{nd} are used in order to regulate the interaction among the components, the global system and the decision maker. The color domain of the places $Stop^{pr}$, Run^{pr}, $Stop_l^{nd}$ is C_0, that is they will contain colored tokens representing the components; while Run_g^{nd}, $Stop_g^{nd}$ are neutral; The non deterministic transitions $PrtoNd$ and $NdtoPr$ are used to assure that the decision maker takes a decision for every component in every time unit.

The schemes, describing how the places $Stop^{pr}$, Run^{pr}, $Stop_l^{nd}$, Run_l^{nd}, $Stop_g^{nd}$ and Run_g^{nd} and the transitions $PrtoNd$ and $NdtoPr$ are connected, are shown in the Fig. 2.3. Observe that the basic schema is instead the same already defined for MDPN but now the arcs are annotated with function $\langle S \rangle$ (input) and $\sum_{i=1}^{m} \langle S_{C_{0,i}} \rangle$ (output) meaning that all components must synchronize in that point.

Now we are going to describe how to derive a unique WN composing the probabilistic part with the non deterministic part.

Places Run^{pr} and $Stop^{pr}$, introduced above, are connected with its run/stop transitions of N^{pr} in the same way as for MDPNs, similarly places Run_l^{nd} and $Stop_l^{nd}$ Run_g^{nd} and $Stop_g^{nd}$ introduced above are connected to the run/stop transitions of N^{nd} as for MDPNs, but now the arcs must be annotated with the following functions:

- $\forall t \in T^{pr} \cup T_l^{nd}$, if $synctype(t) = Some \Rightarrow \langle \sum_{i \in dyn(t)} x_{0,i} \rangle$, where $x_{0,i}$ denotes the i-th component of type C_0 in the color domain of t;
- $\forall t \in Trun^{pr}$, if $synctype(t) = Allbut \Rightarrow \langle \sum_{j \in static(t)} S_{0,j} - \sum_{i \in dyn(t)} x_{0,i} \rangle$

Observe that the arcs connecting transitions T_g^{nd} and places Run_g^{nd}, $Stop_g^{nd}$ are not annotated with any function because these places have *neutral* color (i.e. they contain plain black tokens).

Once the composed WN is built, its RG can be constructed and transformed into a MDP following the same two steps already explained for MDPN.

An efficient analysis technique. We have justified the introduction of the MDWN formalism in order to cope the limitations of the MDPN, but for the moment we have only shown how the MDWN works out the first problem: the MDWN is more convenient than the MDPN for its compactness, readability.

For instance we can observe that the MDWN model in Fig. 2.1 can be extended to n identical components only changing the number of elements in the C_0; while in the MDPN formalism this would require to introduce n component nets.

Anyway the number of MDP states is the same for both formalisms, such that the MDPs obtained are equivalent (they have same number of states).

In this section we are going to describe how to obtain a Lumped MDP from the SRG[2] of the composed WN and we will prove that the optimal reward, computed on the MDP $\mathcal{M}$ obtained from the RG, can be computed directly on the Lumped MDP $\mathcal{M}'$ imposing the following assumptions:

- **Stationary rewards and transition probabilities**; $r(s,a)$ and $p(\cdot|s,a)$ do not vary from decision epoch to decision epoch;
- **Bounded rewards** $\forall s \in S$ and $\forall a \in A_s, |r(s,a)| < \infty$;
- **Discrete and finite state space** S is discrete and finite .

First of all we observe that the following properties are satisfied:

1. $\forall m, m' \in \widehat{\mathbf{m}}, rs(m) = rs(m')$
2. $\forall \langle t, c \rangle, \langle t, c' \rangle, \in \langle t, \widehat{c} \rangle, rt(\langle t, c \rangle) = rt(\langle t, c' \rangle)$
3. $\forall \sigma, \sigma' \in \widehat{\sigma}, rt(\sigma) = rt(\sigma')$

In fact it is easy to prove that these properties of the reward functions rs, rt are satisfied by the symmetries constrains imposed on their definitions.

Afterwords the reward functions rs, rt can be easily extended for the SRG as follows:

Definition 11 (MDWN reward functions)

[2]The SRG is constructed following the standard algorithm described in [3]

- *rs is extended to the symbolic markings as follows:*

$$\forall m \in \widehat{\boldsymbol{m}}, rs(\widehat{\boldsymbol{m}}) \stackrel{def}{=} rs(m)$$

- *rt is extended to the symbolic instance as follows:*

$$\forall \langle t, c\rangle \in \langle t, \widehat{c}\rangle, rt(\langle t, \widehat{c}\rangle) \stackrel{def}{=} rt(\langle t, c\rangle)$$

- *rt is extended to the symbolic instance sequence as follows:*

$$\forall \sigma \in \widehat{\sigma}, rt(\widehat{\sigma}) \stackrel{def}{=} rt(\sigma)$$

The Lumped MDP is obtained again in two steps:

- build from the SRG the SRG_{nd} where all the transition sequences passing only through non deterministic symbolic markings are reduced to one non deterministic step;
- build the SRG_{MDP} from the SRG_{nd} where all probabilistic paths are substituted by single probabilistic arcs.

These two steps are computed exactly in the same way as for the RG, the only difference is in the computation of the transition probability function. The probability of each "symbolic" probabilistic step is obtained from first computing the weight of the corresponding symbolic transition instance and then normalizing it w.r.t all the enabled symbolic transition instances in the source symbolic marking; the weight of the symbolic transition instances is defined as the product of its multiplicity by the weight of any ordinary transition instance represented by it (Appendix A).

Now let us prove that the optimal reward, computed on the MDP $\mathcal{M}$ obtained from the RG, can be computed directly on the Lumped MDP $\mathcal{M}'$.

We prove this showing that a Lumped MDP $\mathcal{M}'$ is *stochastically bisimilar* w.r.t the MDP $\mathcal{M}$. This requires that every ordinary marking $m \in S_{\mathcal{M}}$ is stochastically bisimilar to the symbolic marking $\widehat{\mathbf{m}} \in S_{\mathcal{M}'}$ where $m \in \widehat{\mathbf{m}}$:

- $r_g(rs(\widehat{\mathbf{m}}), rt(\widehat{\sigma})) = r_g(rs(m), rt(\sigma))$ where $\widehat{\sigma}$ is an action enabled in $\widehat{\mathbf{m}}$ and σ is an action enabled in m and $\sigma \in \widehat{\sigma}$;
- $\forall \widehat{\mathbf{m}}' \in S_{\mathcal{M}'}\ p(\widehat{\mathbf{m}}'|\widehat{\mathbf{m}}, \widehat{\sigma}) = \sum_{m' \in \widehat{\mathbf{m}}', \sigma \in \widehat{\sigma}} p(m'|m, \sigma)$;
- $\forall \sigma \in \widehat{\sigma}$ and $\sigma \in A_m, \widehat{\sigma} \in A_{\widehat{\mathbf{m}}},\ \ \sigma \in \widehat{\sigma}' \Rightarrow \widehat{\sigma} = \widehat{\sigma}'$.
- $\forall \widehat{\mathbf{m}}, \widehat{\mathbf{m}}' \in S_{\mathcal{M}'}, \forall \widehat{\sigma} \in A_{\widehat{\mathbf{m}}} : p(\widehat{\mathbf{m}}'|\widehat{\mathbf{m}}, \widehat{\sigma}) > 0 \Rightarrow \forall m \in \widehat{\mathbf{m}}, \exists \sigma \in \widehat{\sigma} : p(m'|m, \sigma) > 0$ where $m' \in \widehat{\mathbf{m}}'$;

The demonstration that the stochastic bisimilarity holds could be immediately deduced. The first point is satisfied by the definition 11, the second by the procedure to compute the probability of the probabilistic transition sequences of the SRG; while the last two points are satisfied by the SRG definition.

This notion of equivalence lead us to the following theorem on optimal value equivalence.

Theorem 1 (Optimal value equivalence) *Let $\mathcal{M}'$ be the lumped MDP stochastically bisimilar to the MDP $\mathcal{M}$ then:*

$$\mathcal{V}_n^*(m) = \mathcal{V}_n^*(\widehat{\boldsymbol{m}})$$

where $m \in \widehat{\boldsymbol{m}}$ and $\sigma \in \widehat{\sigma}$.
Proof:
Let us define the m-step optimal action value function recursively for all the state-action pairs in $\mathcal{M}$

$$\mathcal{V}_n(m,\sigma) = r_g(rs(m), rt(\sigma)) + \sum_{m' \in S} p(m'|m,\sigma) \cdot \max_{\sigma' \in A_{m'}} \mathcal{V}_{n-1}(m', \sigma')$$

and in $\mathcal{M}'$

$$\mathcal{V}_n(\widehat{\boldsymbol{m}},\widehat{\sigma}) = r_g(rs(\widehat{\boldsymbol{m}}), rt(\widehat{\sigma})) + \sum_{\widehat{\boldsymbol{m}}' \in S} p(\widehat{\boldsymbol{m}}'|\widehat{\boldsymbol{m}},\widehat{\sigma}) \cdot \max_{\widehat{\sigma}' \in A_{\widehat{\boldsymbol{m}}'}} \mathcal{V}_{n-1}(\widehat{\boldsymbol{m}}', \widehat{\sigma}')$$

Now we prove by induction on m that the theorem is true.
(1)For case $m = 0$.
We have $\mathcal{V}_0(m,\sigma) = r_g(rs(m), rt(\sigma))$ and $\mathcal{V}_0(\widehat{\boldsymbol{m}},\widehat{\sigma}) = r_g(rs(\widehat{\boldsymbol{m}}), rt(\widehat{\sigma}))$
by the definition 11

$$r_g(rs(m), rt(\sigma)) = r_g(rs(\widehat{\boldsymbol{m}}), rt(\widehat{\sigma}))$$

(2)Let us assume that $\mathcal{V}_{j-1}(m,\sigma) = \mathcal{V}_{j-1}(\widehat{\boldsymbol{m}},\widehat{\sigma})$ for all values of $j-1$ less than n. Now we have:

$$\mathcal{V}_j(\widehat{\boldsymbol{m}},\widehat{\sigma}) = r_g(rs(\widehat{\boldsymbol{m}}), rt(\widehat{\sigma})) + \sum_{\widehat{\boldsymbol{m}}' \in S} p(\widehat{\boldsymbol{m}}'|\widehat{\boldsymbol{m}},\widehat{\sigma}) \cdot \max_{\widehat{\sigma}' \in A_{\widehat{\boldsymbol{m}}'}} \mathcal{V}_{j-1}(\widehat{\boldsymbol{m}}', \widehat{\sigma}')$$

by the definition 11:

$$\mathcal{V}_j(\widehat{\boldsymbol{m}},\widehat{\sigma}) = r_g(rs(m), rt(\sigma)) + \sum_{\widehat{\boldsymbol{m}}' \in S} p(\widehat{\boldsymbol{m}}'|\widehat{\boldsymbol{m}},\widehat{\sigma}) \cdot \max_{\widehat{\sigma}' \in A_{\widehat{\boldsymbol{m}}'}} \mathcal{V}_{j-1}(\widehat{\boldsymbol{m}}', \widehat{\sigma}')$$

by the stochastic bisimilarity definition and by induction hypothesis:

$$\begin{aligned}
\mathcal{V}_j(\widehat{\boldsymbol{m}},\widehat{\sigma}) &= r_g(rs(m), rt(\sigma)) + \sum_{\widehat{\boldsymbol{m}}' \in S} \sum_{m' \in \widehat{\boldsymbol{m}}'} p(m'|m,\sigma) \max_{\sigma' \in A_{m'}} \mathcal{V}_{j-1}(m', \sigma') \\
&= r_g(rs(m), rt(\sigma)) + \sum_{m' \in S} p(m'|m,\sigma) \max_{\sigma' \in A_{m'}} \mathcal{V}_{j-1}(m', \sigma') \\
&= \mathcal{V}_j(m,\sigma).
\end{aligned}$$

Since the reward is bounded then:

$$\begin{aligned}
\max_{\widehat{\sigma} \in A_{\widehat{m}}} \{\mathcal{V}_n(\widehat{\boldsymbol{m}},\widehat{\sigma})\} &= \max_{\sigma \in A_m} \{\mathcal{V}_n(m,\sigma)\} \\
\mathcal{V}_n^*(\widehat{\boldsymbol{m}}) &= \mathcal{V}_n^*(m).
\end{aligned}$$

□

We have shown that the optimal reward computed from Lumped MDP $\mathcal{M}'$ is equivalent to the optimal reward computed by MDP obtained from the RG technique but we have not discussed about the relation between the optimal policies in the two models. We must observe that the optimal policy computed by the Lumped MDP can be always translated to get an optimal policy for the MDP. From an ordinary marking m it is always possible to deduce the corresponding symbolic marking $\widehat{\mathbf{m}}$ and then the optimal associated symbolic action from which it possible to obtain the corresponding optimal ordinary action.

It is important to observe that several optimal ordinary actions may correspond to a symbolic action: in this case we can select randomly one of these possible optimal actions because they are all equivalent. In fact it is possible to show that $\forall m, m' \in \widehat{\mathbf{m}}$, m is stochastically bisimilar to m', so that $\mathcal{V}_n^*(m,\sigma) = \mathcal{V}_n^*(m',\sigma')$ with $\sigma, \sigma' \in \widehat{\sigma}$ and $\widehat{\sigma} \in A_{\widehat{\mathbf{m}}}$.

We can observe that in literature the concept of stochastically bisimilarity has already been used to obtain an MDP size reduction (i.e. in [10, 16]), but all these methods require to build all the MDP state space and then to reduce it, while our method builds directly the Lumped MDP without generating all the state space.

Furthermore the possibility of abstracting details through the symbolic transition allows to obtain a more effective reduction than the method proposed in [10] since we consider as equivalent, actions that are "similar" but not identical.

2.2 Experiments discussion

In this section we will present an example modeling a multiprocessor system where each processor has a local memory; the system includes also a global shared memory that can be used by any processor when its local memory fails. Each processor, local memory and global shared memory can fail independently; however we consider recoverable failures, that can be solved by restarting/reconfiguring the failed component.

The system includes an automatic failure detection system that is able to detect and perform a reconfiguration of the failed component (e.g. by resetting it). The failure detection and recovery system can handle a limited number k of failures in parallel.

Notice that if a local memory M_i and the global shared memory GM are both failed at the same time, the processor P_i cannot perform any useful work, even if it is not failed; moreover we assume that when both the processor P_i and its local memory M_i are simultaneously failed, they are reset together (this is considered as a single reset operation). The components in this system are: n processors, n local memories and one global shared memory, moreover it is convenient from a modeling point of view to introduce a logical "reset" component associated with each "physical" component modelling a reset procedure.

The MDWN probabilistic part of this system is depicted in Fig. 2.1.

The decision maker corresponds to the automatic failure detection and recovery system; it may represent any possible recovery strategy; it is modeled in such a way that any association of up to k recovery resources to any subset of failed components at a given time can be realized by the model. It is depicted in Fig. 2.2.

Unfortunately the use of the standard predicates ϕ_{xyz}, ϕ_{xz}, ϕ_{yz} for this model diminishes the effectiveness of the reduction induced by the SRG technique.

The effectiveness of the reduction induced by the SRG technique is instead higher if we consider the new model in Figs. 2.4 and 2.5; where we use two color classes $\{C_0, C_1\}$ for representing the system components where $C_0 = \{P, M, GM, R\}$, $C_1 = \{1, \ldots, n\}$ and P, M, GM, R are four cardinality one static subclasses. A processor is unequivocally identified by means of pair from $P \times C_1$, in the same way the local memory is identified by means of pair from $M \times C_1$.

This requires to extend the MDWN definition given in section 2.1 to allow that a component identifier be represented with a tuple of colors. This does not affect any other definition and does not require any change in the RG/SRG technique; we have used a single class component identifiers in the MDWN definition in order to make the formalization simpler.

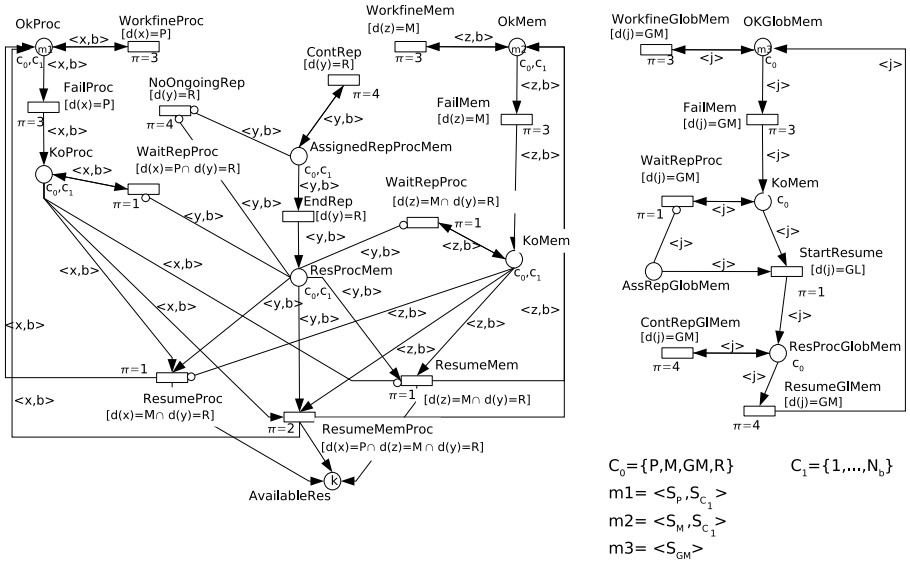


Figure 2.4: Probabilistic part of the multiprocessor system MDWN model with two color classes $\{C_0, C_1\}$ for representing the system components

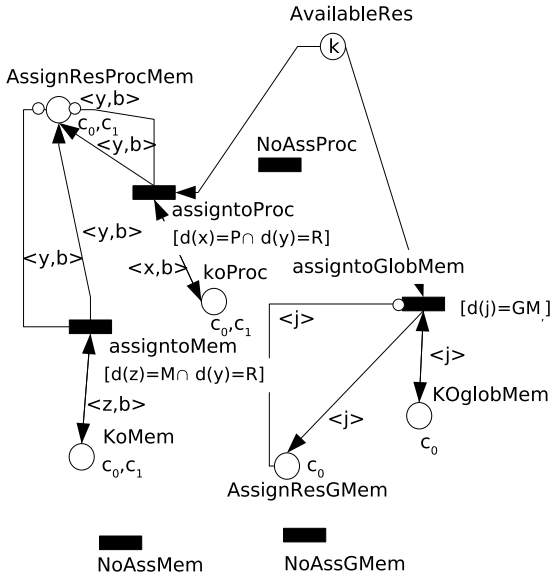


Figure 2.5: Non deterministic part of the multiprocessor system MDWN model with two color classes $\{C_0, C_1\}$ for representing the system components

	Proc=2,Mem=2,Res=2			Proc=3,Mem=3,Res=2			Proc=4,Mem=4,Res=2		
	Prob.	Non det.	Time	Prob.	Non det.	Time	Prob.	Non det.	Time
RG	19057	21031	13s	755506	863886	1363s	26845912	31895848	>13h
RG_{nd}	19057	441	9s	755506	4078	2833s			
RG_{MDP}	0	441	2s	0	4078	250s			
SRG	9651	10665	9s	132349	150779	284s	1256220	1478606	5032s
SRG_{nd}	9651	219	3s	132349	831	222s	1256220	2368	12795s
SRG_{MDP}	0	219	1s	0	831	28s	0	2360	518s
RG *prio*	19057	5235	9s	755506	103172	983s	26845912	1863024	>13h
RG_{nd} *prio*	19057	411	4s	755506	4078	1830s			
RG_{MDP} *prio*	0	411	2s	0	4078	246s			
SRG *prio*	9651	2697	6s	132349	18904	187s	1256220	96044	3270s
SRG_{nd} *prio*	9651	219	2s	132349	831	75s	1256220	2368	1560s
SRG_{MDP} *prio*	0	219	1s	0	831	26s	0	2360	234s

Table 2.1: Results for the example modeling a multiprocessor system. It is important to observe that the RG for Proc=4 and Mem=4 is not computed because it requires a lot of time; its size is computed indirectly by the SRG

The reduction in the size of the model state space of the final MDP are reported in table 2.2; it shows the number of states and the computation time respectively of the $RG, RG_{nd}, RG_{MDP}, SRG, SRG_{nd}$ and SRG_{MDP} for different numbers of banks. The computation was performed with an AMD Athlon 64 2.4Ghz of 4Gb memory capacity. In particular the first, the second and the third lines report the number of states and computation time of the RG, the RG_{nd} and the RG_{mdp}, while the following three lines show the number of states and the computation time obtained using the SRG technique. It is easy to observe how the SRG technique wins both in terms of space and time gain with respect to the RG technique.

A further reduction of the number of states for this model can be achieved associating different priorities to the system transitions such that the interleavings between the non deterministic/probabilistic actions are reduced. For instance the last six lines in table 2.2 show the reduction in terms of non deterministic states and time computation obtained imposing an order on the decision maker choices. (First the decision maker must take all the decisions for the processors then for the memories and in the end for the global memory).

It is important to observe that it is not always possible to use this trick since the actions must be independent; the priorities in practice must not reduce the set of possible strategies. Our tool solves the MDPs using the library *graphMDP* developed at the Ecole Nationale Suprieure de l'Aronautique et de l'Espace Toulouse (ONERA-Toulouse http://www.cert.fr/dcsd/cd/teichteil). The optimal strategy is expressed as a set of optimal actions, such that for every system state an optimal action is given.

For example let us consider a model with two processors and memories, with two recovery resources, with probability of processor fault 0.01, memory fault 0.05, global memory fault 0.001, continuing the processor recovery 0.20, continuing the memory recovery 0.05, and continuing the global memory recovery 0.05, with cost of the processor recovery 200, memory recovery 100 and global memory recovery 150, and with penalty 500 if the number of active processors is one and 1000 if it is zero. In this case we can observe that if a fault happens and there is a free recovery resource then the recovery of this fault must start immediately moreover the global memory recovery is preferred with respect the processor recovery and the memory recovery.

After having obtained the optimal strategy we would like to synthesize a new model without non determinism implementing it (this could be achieved by substituting the decision maker part with a new probabilistic part implementing the decisions of the optimal strategy): classical Markov chain analysis techniques could then be applied to this model, moreover the new net would constitute a higher level (hopefully easier to interpret) description for the optimal strategy. Unfortunately this is not always easy (especially when the number of states is large), but this is an interesting direction of future research.

Chapter 3

Related work and final consideration

3.1 Related work

In this section we are going to compare our formalism with two other high level formalisms for MDP: the PRISM language and the Stochastic Transition System (STS) proposed in [7].

The PRISM language [11] is a state-based language based on the Reactive Modules formalism of Alur and Henzinger [1]. A system is modeled by PRISM language as composition of modules(components) which can interact with each other. Every model contains a number of local variables used to define it state in every time unit, and the local state of all modules determines the global state. The behavior of each module is described by a set of commands; such that a command is composed by a guard and a transition. The guard is a predicate over all the (local/nonlocal) variables while a transition describes how the local variable will be update if the its guard is true.

The composition of the modules is defined by a process-algebraic expression: parallel composition of modules, action hiding and action renaming.

Comparing the MDPN formalism with the PRISM language we can observe that they have the same expressive power: we can define local or global non-deterministic actions and the reward function on the states and/or on the actions in both formalisms; such that it is possible to translate MDPN model directly in a PRISM model. The main difference is that using the MDPN formalism one can define complex probabilistic behaviors and complex non-deterministic actions as a composition of simpler behaviors or actions.

If we compare the PRISM language with the MDWN then we can see that the MDWN has two other advantages: a parametric description of the model and an efficient analysis technique making it possible to automatically take advantage of intrinsic symmetries of the system. In fact the PRISM language has a limited possibility for parametrization. In order to cope with this problem in [8] it was

presented a syntactic pre-processor called eXtended Reactive Modules (XRM) which can generate RM models giving to the users the possibility of describing the system using for instance: *for* loops, *if* statements.

Instead several techniques proposed in order to reduce the states explosion problem in PRISM i.e. in [9] were based on the minimization of the RG with respect to bisimulation; but this requires to build all the state space and then to reduce it; hence our method gives the possibility of managing models with a bigger number of states. It generates directly the Lumped MDP without building all the state space.

A direct comparison between our formalisms and the STS is not possible, because the STSs are an high level formalism for modeling the continuous time MDPs. It extends the Generalized Stochastic Petri Net by introducing transitions with an unspecified delay distributions and by the introducing the possibility of non-deterministic choice among enabled immediate transitions. In every way we can observe that the STS has the same problems of GSPN formalism; that make its utilization less advantage with respect to the WN. It is also important to observe that there are no tools supporting this formalism.

3.2 Conclusion and Future Work

We have introduced MDPNs, based on Petri nets, and MDWNs, based on Well-formed nets, in order to model and analyze distributed systems with probabilistic and non deterministic features. From a modeling point of view, these models support a macroscopic point of view of alternation between the non probabilistic behavior and the non deterministic one of the system and a syntactical way to define the switch between the two behaviors. Furthermore, MDWNs enable the modeler to specify in a concise way similar components. From an analysis point of view, we have adapted the technique of the symbolic reachability graph producing a reduced Markov decision process w.r.t. the original one, on which the analysis may be performed. Our methods are already implemented and integrated in the GreatSPN tool and we have described some experimental results.

The MDPN and MDWN formalisms open up two different research areas for future work. One topic will be the adaptation of the Extended SRG technique [12, 13] for the MDWN model. In fact in the systems with mostly symmetric behavior and occasional local asymmetric behavior (partially symmetric systems) the SRG approach loses most of its efficiency. In these systems the Extended SRG (ESRG) can be used obtaining more states space reduction than using the SRG.

Another important topic will be to explore the (symbolic) optimal policy in order to synthesize it so that a new model without non determinism behaviors can be automatically obtained and studied with classical Markov chain analysis techniques.

Bibliography

[1] R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[2] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *15th Int. Conf. of Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513, Bangalore, India, 1995. Springer.

[3] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, nov 1993.

[4] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured Petri nets. *Theoretical Computer Science*, 176(1–2):39–65, 1997.

[5] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: Graphical editor and analyzer for timed and stochastic petri nets. *Performance Evaluation, special issue on Performance Modeling Tools*, 24(1-2):47–68, November 1995.

[6] L. de Alfaro. Temporal logics for the specification of performance and reliability. In *14th Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *LNCS*, pages 165–176, Hansestadt Lbeck, Germany, 1997. Springer.

[7] L. de Alfaro. Stochastic transition systems. In *9th International Conference on Concurrency Theory*, volume 1466 of *LNCS*, pages 423–438, Nice, France, 1998. Springer.

[8] K. Demaille, S. Peyronnet, and B. Sigoure. Modeling of sensor networks using XRM. In *2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, Paphos, Cyprus, 2006.

[9] H. Garavel and H. Hermanns. On combining functional verification and performance evaluation using CADP. In *In FME 2002: International Symposium of Formal Methods Europe,*, pages 10–429, Copenhagen, Denmark, 2000.

[10] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 2003.

[11] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 441–444, Vienna, Austria, 2006. Springer.

[12] L. Capra, C. Dutheillet, G. Franceschinis, and J-M. Ilié. Exploiting Partial Symmetries for Markov Chain Aggregation. *Electronic Notes In Theoretical Computer Science*, 39 (3), 2000.

[13] M. Beccuti, S. Baarir, G. Franceschinis, and J-M. Iliè. Efficient lumpability check in partially symmetric systems. In *tool paper for the 3st IEEE International Conference on Quantitative Evaluation of Systems (QEST'06)*, Riverside, CA, USA, September 2006. IEEE Computer Society.

[14] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, John Wiley and Sons, 1995. Download http://www.di.unito.it/∼greatspn.

[15] M. L. Puterman. *Markov Decision Processes. Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[16] B. Ravindran and A. G. Barto. Symmetries and Model Minimization of Markov Decision Processes. Technical report, Computer Science Technical Report 01-43 University of Massachusetts, Amherst, MA, 2001. http://www-anw.cs.umass.edu/∼ravi/TR01-43.ps.

Appendix A

WN background: definitions and notation

The Well-formed Net (WN) formalism [3] was inspired by the Colored Petri Net (CPN) formalism and has the same modeling power of CPNs (but unlike CPNs it includes transition priorities and inhibitor arcs). However its color annotations syntax is peculiar: it was defined with the aim of developing efficient analysis techniques able to automatically exploit the behavioral symmetries embedded in the model.

Definition 12 *A Well-formed Net is a ten-tuple*

$$\mathcal{N} = \langle P, T, \mathcal{C}, cd, I, O, H, \phi, prio, m_0 \rangle$$

P and T are the place and transition sets; transition input, output and inhibitor arcs are defined by I, O, H, which define also their color annotations (called arc expressions); m_0 is the initial (colored) marking. *prio* defines the transition priorities (assigning a priority level $prio(t) \in \mathbb{N}$ to each transition). The other elements correspond to the model color annotations, described next.

Basic color classes. $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a set of pairwise disjoint *basic color classes*. C_i is a finite not empty set and it can be partitioned into n_i disjoint subsets $C_{i,j}, j = 1, \ldots, n_i$ called *static subclasses*; $|C_i|$ denotes the cardinality n_i of the partition. A basic color class C_i may be (circularly) ordered, the order is induced by a successor function: the successor of element c is denoted $!c$.

Color domain. cd defines the color domains of places and transitions, which are Cartesian products of basic color classes. The color domain of a node k is denoted $cd(k) = C_1^{e_1} \times C_2^{e_2} \times \ldots \times C_n^{e_n}$, where $e_i \in \mathbb{N}$ is the number of occurrences of class C_i in $cd(k)$ and its value depends on the considered node.

Elementary Function. I, O and H are defined on $T \times P$ and can map a pair t, p to an empty function, meaning that there is no input, output or inhibitor arc connecting t and p, or to an arc function $f : cd(t) \rightarrow Bag(cd(p))$, defined next.

Let us consider the color domain $cd = C_1^{e_1} \times C_2^{e_2} \times \ldots \times C_n^{e_n}$: an *elementary color function* is a linear mapping from cd to $Bag(C_i)$ (for some $i \in 1, \ldots, n$) chosen among the following functions:
- the projection denoted X_i^l defined as: $X_i^l(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n}, \ldots) \mapsto c_i^l$
- the successor denoted $!X_i^l$ defined as: $!X_i^l(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n} \ldots) \mapsto !c_i^l$
- the diffusion function (also called synchronization function, depending if it annotates an output or an input arc), which is constant, denoted S_i and defined as follows: $S_i(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n} \ldots) \mapsto \sum_{\forall c \in C_i} c$

Notice that in practice the symbols X_i^l used above to denote the projection function are substituted by names of transition variables (representing the transition parameters) in the models; each variable has a type C_i. The variable-based notation makes the model more readable, since variables can be given meaningful names.
The diffusion(synchronization) function can be restricted to a static subclass, denoted $S_{i,j}$ and defined as follows: $S_{i,l}(\ldots, c_1^{j_1}, \ldots, c_2^{j_2}, \ldots, c_n^{j_n} \ldots) \mapsto \sum_{\forall c \in C_{i,l}} c$

Due to the linear property of the the elementary functions, they can be just defined on the single elements of the domain. Abusing notation, the elementary functions and their linear extension are usually denoted in the same way.

Class function. A color function f on class C_i, also called C_i class-function, is a linear combination of elementary functions (with same domain and codomain):

$$f_i = \sum_j \alpha_j \cdot X_i^j + \sum_{q=1}^{|C_i|} \beta_q \cdot S_{i,q} + \sum_j \gamma_j \cdot !X_i^j$$

The coefficients $\beta_q \in \mathbb{N}$, $\alpha_j, \gamma_j \in \mathbb{Z}$ must satisfy the following constraint: if f_i^- and f_i^+ are respectively the multisets of elements with negative and positive coefficients in the formula above (so that $f_i = f_i^+ - f_i^-$), then it must hold $f_i^- \subseteq f_i^+$.

Arc function. An arc function F on an input, output or inhibitor arc, connecting transition t and place p, is a sum so defined:

$$F = \sum_k \lambda_k \cdot \bigotimes_{i=1}^{n} \bigotimes_{j=1}^{e_i} f_i^{j,k}$$

where $f_{i,j}^k : cd(t) \to Bag(C_i)$, $\lambda_k \in \mathbb{N}$ and e_i is the number of occurrences of class C_i in $cd(p)$. The symbol $\bigotimes$ denotes the Cartesian product quantifier, in the text we shall also use the alternative representation $\langle f_1^1, f_1^2, \ldots f_n^{e_n'} \rangle$, briefly called *function tuple* (or simply *tuple*).

Transition and color function guards. A *guard* is a Boolean expression defined on a transition color domain whose basic terms are *standard predicates*. Standard predicates allow to compare color elements from the same color class C_i, and can take the following form:
- $[X_i^j = X_i^k](c)$, it evaluates to *true* iff the j^{th} component of type C_i in c is equal to the k^{th} component of same type;

- $[d(X_i^j) = C_{i,h})](c)$, it evaluates to *true* iff the j^{th} component of type C_i in c belongs to $C_{i,h}$;
- $[d(X_i^j) = d(X_i^k)](c)$, it evaluates to *true* iff the j^{th} and k^{th} components of type C_i in c belong to the same static subclass.

In WNs the ϕ function associates a guard with each transition, the default guard is $[true]$. Moreover, guards may be employed in arc expressions. Let F be a function tuple and g a guard, then the guarded tuple is defined as

$$[g]F \stackrel{def}{=} \text{ if } g(c) \text{ then } F(c) \text{ else } \emptyset$$

A guarded arc function is then a linear combination of guarded tuples. We use the notation $cd_\phi(t)$ denots the subset of $cd(t)$ satisfies ϕ_t

Definition 13 (Marking) *A marking m is expressed as a distribution of colored tokens in the all places.*

Definition 14 (Transition instance) *A transition instance $\langle t, c\rangle$ in m is a binding c of the transition variables to the objects in the appropriate color class.*

The evolution of an WN system is defined through a firing rule applied to a given transition instance $\langle t, c\rangle$. The new marking obtained from the transition instance firing satisfies: $\forall p \in P, m'(p) = m(p) - I(p,t)[c] + O(p,t)[c]$.

Now we introduce some definitions and properties concerning the SRG technique.

Definition 15 (Color permutation) *Let $\xi = \{s = \langle s_1, \ldots, s_h, s_{h+1}, s_n\rangle\}$ be a subgroup of the permutation on $C_1, \ldots, C_n$ such that:*

- $\forall 0 < i \leq h$ *s_i is a permutation on C_i such that* $\forall 0 < k \leq n_i, C_{i,k} = s_i(C_{i,k})$
- $\forall h < i \leq n$ *s_i is a rotation on C_i such that* $\forall 0 < k \leq n_i, C_{i,k} = s_i(C_{i,k})$

We recall the definition of marking permutation:

Definition 16 (marking permutation) *Let $m(p)$ be the marking of a place p, and $s \in \xi$. Then $m(p)' = s.m(p)$ is a marking defined by applying the s to each object tuple c in $m(p)$.*

The firing propriety is preserved with respect to the color permutation:

Property 1 *The firing property is preserved by applying a permutation both on the markings and the transition instance. $\forall m$ ordinary, $\forall t \in T$, $\forall c \in cd_\phi(t)$ and $\forall s \in \xi$*

$$m[t, c\rangle m' \Leftrightarrow s.m[t, s.c\rangle s.m'$$

In this way we introduce also the definition of symbolic marking .

Definition 17 (Symbolic Marking) *Let Eq be the equivalence relation defined by:*

$$mEqm' \Leftrightarrow \exists s \in \xi, m' = s.m$$

An equivalence class of Eq is called symbolic marking ($\widehat{\boldsymbol{m}}$).

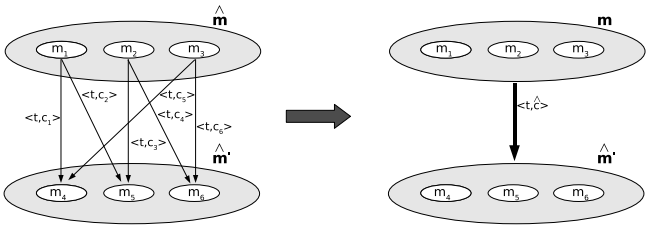


Figure A.1: Grouping of ordinary transition instance firings in a symbolic transition instance firing

In order to build the SRG directly starting from a symbolic initial marking, without building the RG and then grouping markings into equivalence classes, a symbolic firing rule was defined on the symbolic marking in [3]. This requires to introduce the concept of a symbolic instance $\langle t, \widehat{c}\rangle$. A symbolic transition instance represents groups of ordinary transition instances as depicted in Fig. A.1 such that the symbolic instance firing $\widehat{\mathbf{m}} \overset{\langle t,\widehat{c}\rangle}{\rightarrow}$ stands for all the ordinary firing instances that can be obtained by valid assignments of objects to $\widehat{c}$

Its cardinality $|\widehat{\mathbf{m}} \overset{\langle t,\widehat{c}\rangle}{\rightarrow}|$ corresponds to the number of ordinary firings from each single ordinary marking $m \in \widehat{\mathbf{m}}$ that are represented by the symbolic firing instance (e.g. the cardinality of the symbolic firing instance in Fig. A.1 is two).

Appendix B

Tool: description and use

This is a prototype package that uses the GreatSPN package[1] in order to draw the models (MDPN/MDWN) and to generate the (S)RG of these models, and the GraphMDP library[2] in order to solve the derived MDP.

Initially the user draws the components and the decision maker net (MDPN/ MDWN) through the GUI of GreatSPN[3] and composes them using algebra tool. After that, the $WNRG$ or the $WNSRG$ program[4] executed with the option $-r$ generates the file .dot containing the (S)RG of the model in the dot format[5]. The model rewards rs and rt are defined in the file .reward following the grammar in Fig. B.1. The $RG2RRG$ program taking in input the .dot file and the .reward file returns the .RG1 and .RG2 files containing the RG_{nd} and the RG_{MDP} in the dot format and the .xml file containing the derived MDP in XML format (the DTD of the .xml file is shown in Fig. B.2)

This .xml file is given in input to the MDP_solve program that solves the MDP returning the optimal strategy and the optimal (average) reward.

The table B.1 shows the input and output files for every program while the table B.2 describes these files.

[1] GreatSPN is a software package for the modeling, validation, and performance evaluation of distributed systems using Generalized Stochastic Petri Nets and their colored extension: Stochastic Well-formed Nets. It is available for free for universities and non-profit organizations (http://www.di.unito.it/ greatspn/)

[2] GraphMDP is a library for modeling, drawing and solving a Markov Decision Process (http://www.cert.fr/dcsd/cd/teichteil/)

[3] For the moment the probabilistic transitions are drawn as timed transitions and those deterministic as immediate transitions

[4] the $WNRG$ computes the Reachability Graph of the model, while the $WNSRG$ compute the Symbolic Reachability Graph of the model

[5] It is important to observe that RG can be drawn using the dot program: dot ⟨net_name⟩.dot -Tps -o ⟨net_name⟩.ps

Reward = TransitionReward | PlaceReward | FormulaReward
TransitionReward = **T** ⟨transition_name⟩ value
PlaceReward = **P** ⟨place_name⟩ value
FormulaReward = **F** value (Arguments)
Arguments = ⟨place_name⟩ op value | ⟨place_name⟩ op value **and** Arguments
op = > | = | <
value = double

Figure B.1: The .reward file grammar

```
<!DOCTYPE MDP[
    <!ELEMENT MDP (STATES,ACTIONS)>
        <!ATTLIST MDP
            stationary (yes | no)#REQUIRED
            Infinite (true | false) #REQUIRED
            InfiniteCriteriumType (DISCOUNTED
                                   |AVERAGE_REWARD) #IMPLIED
            DecompositionAlgorithmType NMTOKEN #IMPLIED
            Horizon NMTOKEN #IMPLIED
            Optimization_algorithm (POLICY_ITERATION
                                    | VALUE_ITERATION
                                    | LINEAR_PROGRAMMING) #IMPLIED>
        <!ELEMENT STATES (STATE)+ >
            <!ELEMENT STATE (LABEL)>
                <!ELEMENT LABEL (#PCDATA)>
        <!ELEMENT ACTIONS (ACTION)+ >
            <!ELEMENT ACTION (LABEL,TRANSITIONS)>
                <!ELEMENT LABEL (#PCDATA)>
            <!ELEMENT TRANSITIONS (TRANSITION)+>
                <!ELEMENT TRANSITION (STARTING_STATE,ENDING_STATE,
                                      PROBABILITY,REWARD)>
                    <!ELEMENT STARTING_STATE (#PCDATA)>
                    <!ELEMENT ENDING_STATE (#PCDATA)>
                    <!ELEMENT PROBABILITY (#PCDATA)>
                    <!ELEMENT REWARD (#PCDATA)>
] >
```

Figure B.2: DTD of the .xml file

Input files	**Program**	**Output files**
.def and .net	$WNRG/WNSRG$ -r	.dot
.def, .net, .dot and .reward	$RG2RRG$	.RG1, .RG2 and .xml
.xml	MDP_solve	

Table B.1: Input and output files for some modules

Extension	**Format**	**Description**
.dot	ASCII	model (S)RG in dot format
.RRG1	ASCII	model $(S)RG_{nd}$ in dot format
.RRG2	ASCII	model $(S)RG^{MDP}$ in dot format
.reward	ASCII	model reward
.xml	ASCII	model MDP in XML format

Table B.2: List of files