Nondeterminism in Stochastic Modeling

Daulet Turetayev

Formal Methods and Tools Group, University of Twente
P.O. Box 217, NL-7500 AE Enschede, The Netherlands

Abstract. This paper is devoted to the notion nondeterminism in general and particularly in stochastic modeling. It discusses how stochastic modeling formalisms can be mapped to a corresponding configuration graph which is convenient for the investigation of nondeterminism phenomena. Algorithms for mapping stochastic models onto configuration graphs are introduced, and two approaches – well-specified check and weak bisimulation – for eliminating nondeterminism are reviewed and related.

1 Introduction

One can observe an ever accelerating boom of information technology, and consequently an impetuous development of computer science. Reality more and more makes new demands. Modern methods to design and handle computer science innovations become more and more complex, and embrace more and more system aspects. One such aspect is stochastics, through the prism of which many already existing design directions are reconsidered or completed. This paper considers stochastic modeling in the context of a phenomenon usually known as nondeterminism. The latter is treated frequently as something mystical. While on the one hand, one cannot just 'exclude' it from the investigations, from another point of view it prevents to get 'pure' stochastics in when modeling. This paper provides some insight into this problem.

1.1 Stochastic Modeling

Understanding the behavior of real systems is always difficult due to the complexity of their organization and to the intricacy of the interactions among their components. Reasoning about the behavior of systems can become more reliable if proper descriptions are available that help in clarifying the relationships among system components. The best way of obtaining such descriptions is that of constructing a model that highlights the important features of the system organization and provides ways of quantifying its properties neglecting all those details that are relevant for the actual implementation, but that are marginal for the objective of the study. So a model describes the systems at an appropriate level of details, one omits the details which are irrelevant for the aspects of interest.

Models can be developed for a variety of reasons that include understanding and learning about the behavior of the system, improving its performance and making decisions about its design or its operation. Models are useful for explaining why and how certain features of the system actually occur. The model helps in developing insights in the operation of the system and understanding the directions of influence that certain input parameters may have on the results.

Nowadays especially stochastic (probabilistic) modeling plays a key role in modern science. Because of the complexity of the reality our life becomes more and more dependent on random events. In order to express them we introduce elements of stochastics (in particular probabilities) into our models. However, it is worth to mention that probabilities are as old as civilization. Egyptian excavations have brought up several perfectly shaped dice. Even the Bible notes the use of lots to make decisions several times. It might therefore be surprising that it was not until the 16th century that a mathematical probability theory was developed.

Pioneering work was done by Leibniz and Pascal. Since that time, probability theory turned out to be a powerful means to model and analyze reality and rapidly entered in all fields science. Today one finds stochastic models in physics, sociology, economics, epidemiology, etc. Also in computer science, there are various applications of stochastic.

Probabilities can be used to model unreliable or unpredictable behavior exhibited by a system. Secondly, stochastic mechanisms play a dominant role in performance evaluation of computer systems and finally, randomized algorithms make clever use of coin flips or other probability mechanisms to obtain algorithmic solutions to otherwise unsolvable problems. Stochastic mechanisms can be used to model unreliable behavior of a system or its environment.

There are some fully investigated stochastic models that deal with probabilistic choice. Discrete time Markov Chains (DTMCs) have been invented by A.A. Markov and further developed by A.N. Kolmogorov, who was also a pioneer in Continuous Time Markov Chains (CTMCs). Semi-Markov chains are much newer [1]. All these models have been applied in a wide variety applications, including genetics, physics, queuing networks, and so on.

1.2 Nondeterminism

Nondeterministic choices can be specified in transition systems by having several transitions leaving from the same state. Nondeterminism is used when we wish to incorporate several potential system behaviors in a model. Hoare [2] phrases it as follows:

There is nothing mysterious about nondeterminism, it arises from the deliberated decision to ignore the factors which influence the selection.

Nondeterminism choices are often divided into *external* and *internal* nondeterministic choices. External nondeterministic choices are choices that can be influenced by the environment. Since interaction with the environment is performed via external actions, external nondeterminism is incorporated by having several transitions with different labels leaving from the same state.

Internal nondeterministic choices are choices that are made by the system itself, independent of the environment. These are modeled by internal actions or by having several transitions with the same labels leaving from the same state. In the literature the word nondeterminism sometimes refers to what we call internal nondeterminism.

As pointed out by [3], [4], nondeterminism is essential for the modeling of the following phenomena.

Scheduling freedom When a system consists of several components running in parallel, we often do not want to make any assumptions on the relative speeds of the components, because we want the application to work no matter what these relative speeds are. Therefore, nondeterminism is essential to define the parallel composition operator, where we model the choice of which system takes the next step as an (internal or external) nondeterministic choice.

Implementation freedom Systems are often used to represent a specification. Good software engineering practice requires the specification to describes *what* the system should do not *how* it should be implemented. Therefore, a specification usually leaves room for several alternative implementations. Since it does not matter for the correct functioning of the system which of the alternatives is implemented, such choices are also represented by (internal or external) nondeterminism.

External environment A system interacts with its environment via its external actions. When modeling a system, we do not wish to stipulate how the environment will behave. Therefore the possible interactions with the environment are modeled by (external) nondeterministic choices.

Incomplete information Sometimes it is not possible to obtain exact information about the system to be modeled. For instance, one might not know the exact duration of or – in probabilistic systems – the exact probability of an event, but only a lower and upper bound. In that case, one can incorporate all possible values by a nondeterministic choices. This is appropriate since we consider a system to be correct if it behaves as desired no matter how the nondeterministic choices are resolved.

1.3 Nondeterministic Stochastic Models

A Markov Decision Process (MDP) [5] is basically a probabilistic system with external nondeterministic choices only. This model is sometimes called the *reactive model* [6]. There are also some high level modeling formalisms that combine probabilistic choice with nondeterminism: Stochastic Activity Nets (SAN) [7], some variants of Stochastic Petri Nets (SPN) [8], and Interactive Markov Chains (IMC) [9].

1.4 Contribution of the Paper and Organization

While nondeterminism is a useful modeling concept, it hampers the application of standard analysis techniques, which often require a stochastic model without any nondeterminism. Therefore, different techniques have been proposed in

the scientific literature to turn a stochastic model with nondeterminism into a stochastic model without. This paper reviews two different techniques, and sets them in relation to each other. The techniques considered are the so-called well-specified check [10], [11], and the weak bisimulation minimisation approach [9]. While the former has been developed in the context of Petri nets, the latter has its roots in process algebra. We discuss both techniques in the context of Petri nets.

This paper is organised as follows: section 2 briefly introduces the Stochastic Petri Net formalism and the standard procedure of mapping an SPN onto what we call a configuration graph. Section 3 defines a simple generalisation of Stochastic Petri Nets with immediate transitions, and discusses techniques for handling nondeterminism in the resulting configuration graph. Section 4 discusses SPN-based approaches from the literature and discusses algorithms to eliminate nondeterminism for them. Finally section 5 concludes the paper.

2 Stochastic Petri Nets

Stochastic Petri Nets have emerged as a popular way of expressing performance and dependability models. They have been successfully used in building performance and reliability models with complex structural interactions. Full description of them one can find in numerous papers, for example, [12], [13], [8]. Due to their popularity, we do not review them again, but only focus on interesting aspects for us: stochastics and nondeterminism. In short, timed Petri Nets in which the transition firing delays are specified by random variables lead to stochastic models.

The use of exponential distributions for the definition of temporal specifications is particularly attractive because timed PN in which all the transition delays are exponentially distributed can be mapped onto continuous-time Markov Chains [8]. In this case the so called memoryless property of the exponential distribution makes unnecessary the distinction between of the delay itself, and the distribution of the remaining delay after a change of state.

Stochastic Petri Nets (SPN) are timed (transition) PN with atomic firing and in which transition firing delays are exponentially distributed random variables: each transition t_i is associated with random firing delay whose probability density function is a negative exponential with rate w_i.

Formally, a SPN model is a 6-tuple:

$$SPN = (P, T, I(.), O(.), W(.), m_0)$$

$P = \{p_1, p_2, ..., p_{|P|}\}$ is the set of places, $T = \{t_1, t_2, ..., t_{|T|}\}$ is the set of transitions, $m_0 = (m_{01}, m_{02}, ..., m_{0|P|})$ is the initial marking, assigning natural numbers to places: $m_0 : P \to \mathbb{N}$. According to the tradition in the literature we will name these numbers as *tokens*. I(.), O(.) - functions defined on T, are represented as directed arcs from places to transitions and vice versa. The function W allows the definition of the stochastic component of a SPN model mapping

transitions into real positive values of the SPN marking, thereby providing us with the rates of the exponential distribution.

A transition t is *enabled* if and only if each input place contains a number of tokens *greater or equal* than given thresholds defined by $I(t)$. We have a situation of *conflict* when, being several transitions enabled in the same marking, we have to choose which one to fire and, by so doing, we affect the enabling conditions of the other transition.

A SPN expresses timed dynamics; this implies that each timed transition possesses a timer. The timer is set to a value that is sampled from the negative exponential pdf associated with the transition, when the transition becomes enabled for the first time (after firing). During all time intervals in which the transition is enabled, the timer is decremented. Transitions fire when their timer reading goes down to zero.

With this interpretation, each timed transition can be used to model the execution of some activity in a distributed environment; all enabled activities execute in parallel (unless otherwise specified by the PN structure) until they complete. At completion time, activities induce a change of the system state.

As a matter of fact we do not have any nondeterminism in SPNs and as a result we can map them on CTMC. Let's do this step by step. First of all we consider a transformation from SPN to what we call a *configuration graph (CG)*, a structure that is easy to deal with and which will reappear later in slightly more complicated forms. For distinction purposes, this first *CG* version will be referred to as CG^{SPN}.

Definition 1. *A* CG^{SPN} *is a triple* $(S, \multimap, F)$ *where*

- S *is a nonempty set of states, and*
- $\multimap \subset S \times R^+ \times S$ *is a set of timed transitions,*
- $F \in S$ *is the initial state.*

Here timed transitions are labeled with positive real values, corresponding to exponentially distributed random variables of SPN transitions. Set S *consists of a single type of states:*

- ◯ – *stable states.*

The states of a configuration graph represent reachable markings of a Petri net. The following algorithm maps a SPN on CG^{SPN}.

Exploring the SPN: start with initial marking m'
Explored={}
Unexplored={m'}
while {*Unexplored* nonempty} {
  take a marking m from *Unexplored*, move it from *Unexplored* to *Explored*;
  for each transition t enabled in marking m {
    fire t, this leads to marking n;
    add $m \xrightarrow{W(t)} n$ to configuration graph;

```
        if (n not in Unexplored, and n not in Explored) {
          add n to Unexplored;
        }
    }
}
```

The CG^{SPN} does not contain any nondeterminism because ***no special mechanism is necessary for the resolution of timed conflicts***: the temporal information provides a metric that allows the conflict resolution. How can we see it? The conflict resolution depends on the delays associated with transitions and is obtained through the so-called *race* policy: when several timed transitions are enabled in a given marking m, the transition with the shortest associated delay fires first (thus disabling the possible conflicting transitions). With this definition we get the impossibility for two timers to expire at the same time: since the probability that a sample extracted from a negative exponential pdf takes a specific value x equals zero, the probability of two timers expiring at the same time is null. Indeed, given the value sampled by the one of the timers, the probability that the other one samples the same value is zero.

So, according to what is stated, the CG^{SPN} in fact is already a CTMC. It does not exhibit nondeterministic behavior, since each possible behavior at any point in time has a certain, unique probability.

3 Generalized Stochastic Petri Nets

Extensions to SPNs, including GSPNs [8] and SANs [7], allow for both timed and immediate transitions. We treat both of them here more in an abstract form first, which we call generalized SPNs (gSPNs). Several reasons suggest the introduction of the possibility of using immediate transitions into PN models together with timed transitions. The firing of a transition may describe either the completion of a time-consuming activity, or the verification of a logical condition. It is thus natural to use timed transitions in the former case, and immediate transitions in the latter.

Formally, a gSPN model is an 8-tuple

$$gSPN = (P, T, \Pi(.), I(.), O(.), W(.), m_0)$$

$P, T, I(.), O(.), W(.)$ and m_0 have the same meanings. $\Pi(.)$ maps transitions into non-negative natural numbers representing their priority level. Timed transitions are associated with priority zero, whereas all other priority levels are reserved for immediate transitions. Later on we assume that all immediate transitions have equal priority, except section 4, where we shortly show the case of unequal priority over immediate transitions as an alternative way to eliminate nondeterminism.

Immediate transitions against timed ones. Immediate transitions are fired with priority over timed transitions. Is there the need for an explicit priority of

immediate over timed transitions? Let's take the following example which will help to understand this point. Consider a conflict between an immediate and a timed transition and assume for a moment that priority does not exist. The race policy, which means that the enabled transition whose firing delay is minimum will fire, makes the immediate transition always win, except for the case in which a zero delay is sampled from the negative exponential pdf. The probability of selecting the value zero is null theoretically. But practically some problems may arise when the conflict set is enabled infinitely often in a finite time interval. For example, in the case of Fig. 1, where timed transition is drawn as hollow bar, immediate one is drawn as solid line, the timed and the immediate transitions are always enabled, because the firing of the immediate transition does not alter the PN marking. The situation is changed only when the timed transition fires which can happen. After an infinite number of firings of the immediate transition (one may argue that among infinitely many samples drawn, one of them must be 0 with probability 1). To avoid these (sometimes strange) limiting behaviors, the priority of immediate over timed transitions is introduced in the gSPN definition. This makes the timed transition in Fig. 1 never enabled.

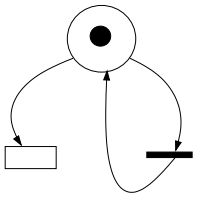

Fig. 1. A conflict set comprising a timed and an immediate transitions

Immediate transitions against immediate ones. In the case of immediate transitions, the gSPN dynamics consumes no time: everything takes place instantaneously. This means that if only one immediate transition is enabled, it fires, and the following marking is produced.

A fundamental problem that arises from immediate transitions is that of *what to do when multiple immediate transitions are enabled at the same time.* The net may behave differently depending on which transition is chosen to fire first. At the first sight one can have surmise that if several immediate transitions are enabled, a metric is necessary to identify which transition will produce the marking modification. But since this metric is not provided in the gSPN definition, we find ourselves in a nondeterministic situation.

The presence of nondeterminism may hamper a probabilistic analysis of many stochastic systems. It appears as a result that it is necessary to take care of it. However, the question arises how to associate a stochastic process (a Markov chain) which does not exhibit nondeterministic behavior to a stochastic model that contains nondeterminism. We will consider this question in section 4.1 in the context of the CG^{gSPN} presented below.

Definition 2. *A CG^{gSPN} is a tuple $(S, \longrightarrow, \dashrightarrow, F)$ where*

- *S is a nonempty set of states, partitioned in two disjoint subsets,*
  - *$\acute{S}$ – stable states ($\bigcirc$), and*
  - *D – decision states ($\triangle$).*
- *$\longrightarrow \subset D \times S$ is a set of immediate transitions,*
- *$\dashrightarrow \subset \acute{S} \times R^+ \times S$ is a set of timed transitions, and*
- *$F \in S$ is the initial state.*

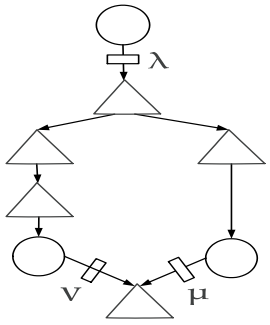

Fig. 2. Example of CG^{gSPN}

Note that according to the above definition, in stable (decision) states only timed (immediate) transitions are available. Consequently, every decision state can be considered as vanishing one. Before we discuss further let's present shortly some notation for the reader's convenience. We use $ET(m)$ to denote the set of enabled transitions in marking m and $IET(m)$ to denote the set of enabled immediate transitions in marking m.

Formally, a gSPN gives rise to a CG^{gSPN} according to the following algorithm:

Exploring the gSPN: start with initial marking m'
Explored={}
Unexplored={m'}
while {*Unexplored* nonempty} {
  take a marking m from *Unexplored*, move it from *Unexplored* to *Explored*.
  if$|IET(m)| \neq \emptyset$ {
    $ET(m) = IET(m)$
  }
  for each transition $t \in ET(m)$ {
    fire t, this leads to marking n;
    if (t is a timed transition) {
      add $m \xrightarrow{W(t)} n$ to configuration graph
    }

```
            else add m ⟶ n to configuration graph;
            if (n not in Unexplored, and n not in Explored) {
              add n to Unexplored
            }
        }
}
```

3.1 Techniques For Eliminating Nondeterminism

In the above, we have discussed how to generate a CG from a gSPN. Contrary to the situation in the SPN context, where the CG^{SPN} was equivalent to a CTMC, the CG^{gSPN} will capture the nondeterminism present in a gSPN behavior. Recall that the presence of nondeterminism is problematic, if it comes to standard stochastic analysis techniques. There are not so many techniques for getting rid of the nondeterminism. To the best of our knowledge actually only two concepts have appeared in the literature. They are known as weak bisimilarity [9] and well-specified check (WSC) [10], [11].

In fact, one looses the essence of nondeterminism when eliminating it. But, in a nutshell, the trick of both algorithms is that we remove only 'equivalent' traces, thereby remaining within the framework of the phenomenon nondeterminism. Of course we are not guaranteed to have 'equivalence' in all our graphs, so we are restricted in our possibilities, that's why we claim that weak bisimilarity and WSC may eliminate nondeterminism, but they do not necessarily eliminate all nondeterminism. For instance, the example on the Fig. 2 provides some insight into the influence of nondeterminism. As we can see from a stochastic point of view, the behavior is not completely determined: among four decision states there is one topmost which has nondeterministic choice and thus it is not clear how to associate a CTMC to this example. We will return to the consideration of the question whether we are able to transform given CG^{gSPN} into Markov chain after presenting our techniques.

Weak Bisimulation. We recall the definition of weak bisimilarity from [9].

Definition 3. *An equivalence relation $\mathcal{E}$ on S is a weak bisimulation iff $P\mathcal{E}Q$ implies*

1. *$P \xrightarrow{\forall} P^{'}$ implies $Q \xrightarrow{\forall} Q^{'}$ for some $Q^{'}$ with $P^{'}\mathcal{E}Q^{'}$,*
2. *If $P^{'}$ and $Q^{'}$ are stable states, then must be held $\gamma(P^{'}, C) = \gamma(Q^{'}, C)$ for all equivalence classes C of $\mathcal{E}$,*

$\xrightarrow{\forall}$ means any sequential number, more or equal 0, of immediate transitions $\longrightarrow$ and γ is real-valued function: $S \times 2^S \mapsto \mathbb{R}^+$, that calculates the cumulative rate to reach a set of states C from a single state R:

$$\gamma(R, C) = \sum\{\mid \lambda \mid R \xrightarrow{\lambda} R^{'} \text{and } \ R^{'} \in C \mid\}.$$

Two processes P and Q are weakly bisimilar if they are contained in some weak bisimulation.

On the left side of the Fig. 3 one of the examples of CG^{gSPN} is depicted

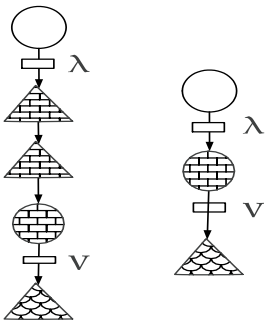

Fig. 3. Example of CG^{gSPN} without nondeterminism and the application of weak bisimilarity

which is divided into equivalence classes according to the weak bisimulation, where bisimilar states are shaded with the same pattern. Thus, every pattern constitutes an equivalence class of weak bisimilarity. Now, replacing the states with the same pattern by just a single state, and only representing transition that cross the border of equivalence classes one can reduce any given CG^{gSPN} to its quotient under weak bisimilarity [9].

In the sequel, this process of constructing the weak bisimilarity quotient of a CG^{gSPN}, is called *an application of weak bisimilarity* to the graph. There is no difficulty in understanding that if the initial CG^{gSPN} is given without any nondeterministic choices the application of weak bisimilarity has to lead to a CG^{gSPN} without any immediate transition and hence we a get Markov chain. But this property cannot be guaranteed in the general case and the above example is one of these cases (Fig. 4). Due to nondeterminism, the stochastic

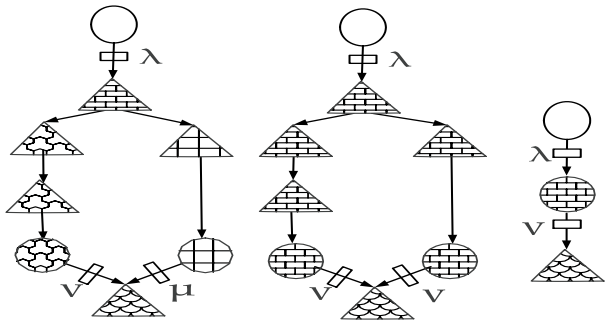

Fig. 4. Examples of CG^{gSPN} with nondeterminism and the application of weak bisimilarity

process to be associated with this CG^{gSPN} is underspecified. This evaluation implicitly assumes that the rates ν and μ are distinct values. If, on the other hand, we have $\nu = \mu$ the situation is different, since the states reached by means of immediate steps turn out to be bisimilar. Thus the nondeterministic choice is not a decisive one, since any outcome of this choice leads into the same class of behaviors. Hence, weak bisimilarity removes immediate transitions and nondeterminism. The resulting Markov chain is depicted on the Fig. 4. Thus finally we can conclude that whenever nondeterminism is faced, the only thing that we can hope is that the subsequent behaviors are equivalent, because then weak bisimilarity will eliminate the nondeterministic state. Otherwise, the specification is underspecified from a stochastic point of view.

Well-Specified Check. Another approach, called well-specified check, is an attempt at managing immediate events in SANs which can be considered as stochastic generalizations of Petri Nets. One restriction here should be mentioned that no cycles are allowed among decision states. This is an on-the-fly algorithm, i.e. it can be incorporated into the generation of the configuration graph. We do so below, in a form tailored to gSPN:

Exploring the gSPN: start with initial stable marking m'
Explored={}
Unexplored={m'}
while {*Unexplored* nonempty} {
  take a marking m from *Unexplored*, move it from *Unexplored* to *Explored*.
  for each transition t enabled in marking m {
    fire t, this leads to marking n;
    if (t is a timed transition) {
      if ($WSC(n)$ contains just one element k) {
        add $m \xrightarrow{W(t)} k$ to the graph
        }
      else {
        add $m \xrightarrow{W(t)} n$ to the graph;
        for each k in $WSC(n)$ {
          add $n \longrightarrow k$ to the graph.
        }
      }
    }
    if (k not in *Unexplored*, and k not in *Explored*) {
      add k to *Unexplored*
    }
  }
}

where $WSC(n)$ is defined in the following way:

WSC(state n) {

```
Result= {};
if n is stable {
     Result= {n}
}
else {
   for each immediate transition t enabled in marking n {
        fire t, this leads to marking h;
        add WSC(h) to Result;
   }
}
return Result; }
```

By construction the states added to *Unexplored* are stable. Also the initial marking m' is stable. Therefore, the first **if** clause in the main algorithm is always fulfilled.

Informally, as long as the way of going to a next stable state through sequences of immediate transitions is independent of the order in which they complete, the CG^{gSPN} is well-specified. For instance, the example depicted on the Fig. 2 is classified as non well-specified, because using different branches of decision node one gets to different stable states. Consequently it is impossible to eliminate the nondeterminism by the well-specified check. Recall that it can be removed by applying weak bisimulation.

Originally this algorithm was offered only for checking whether the given graph is well specified or not, whereas here we develop it to produce the configuration graph even if it is not well specified.

The next formal statement expresses the relation of WSC and weak bisimulation signals to us that the latter is stronger than the former.

Proposition 1. *For any configuration graph P, $WSC(P)$ and P are weakly bisimilar.*

Proof. We construct a weak bisimulation as follows. Let S' be the state space of $WSC(P)$ and S the one of P. Note that $|S'| \leq |S|$, and that there is an obvious injection $\phi : S' \mapsto S$ (each element of S' corresponds to one in S). We partition the states of $S \cup S'$ into the following classes $C_{s'}$, indexed by elements of $s' \in S'$.

$$C_{s'} = \{\ s\ \mid\ (s \xrightarrow{\forall} \phi(s') \text{ and } \forall s'' \in S' : s \xrightarrow{\forall} \phi(s'')) \text{ implies } s' = s''\ \} \cup \{s'\}$$

It is easy to check that $\bigcup_{s \in S'}(C_s \times C_s)$ is a weak bisimulation satisfying Definition 3.

This proposition provides us with the result that weak bisimulation is stronger than WSC in the sense of eliminating nondeterminism. It arises from the fact that if WSC can remove nondeterminism then it can be done by weak bisimulation, while the converse it is not true. The graph in the middle on the Fig. 4 shows it: the application of weak bisimulation to this graph returns a Markov chain, while at the same time after applying algorithm WSC the topmost decision state still will represent a nondeterministic choice.

The considered algorithm involves enumerating and storing all possible paths (from stable states to the next stable ones), which can be prohibitive for models with large subgraphs of immediate transitions, or even for models with smaller subgraphs that are frequently used. But the advantage of this approach is a possibility to do it on-the-fly while in the case of weak bisimulation we have to know all graph in advance.

4 Extensions of gSPN

In the above we touched upon the issue of nondeterminism that arises from immediate transitions when they are enabled at the same time. Let us now consider whether the nondeterminism can be avoided a priori, by some form of enhancement to gSPN. With respect to gSPN historically, there have been several approaches to solving this problem.

Probabilities and weights. An early solution was to assign probabilities to any immediate transitions conflicting in a nondeterministic choice. Since it is not always easy or practical to foresee all the possible combinations of enabled immediate transitions, this solution could be tedious to the user for larger, complex models.

Later refinements to gSPNs assigned weights to immediate transitions. The probability of choosing an immediate transition is then the ratio of the weight of the immediate transition to the sum of the weights of enabled immediate transitions. While this is a reasonable solution, it still requires that the user have some implicit knowledge of what sets of immediate transitions may be enabled at one time. Of course this can be difficult for larger models. Also, it sometimes leads to over-specification, when the user is required to specify weights that are irrelevant. Furthermore, the modeler may make an error in building the model, in which case the result of over-specification is legal but may result in unanticipated and incorrect behavior. Thus, over-specification can be undesirable.

Confusion. Nondeterminism is expressed through confusion which in its turn can be expressed through conflict. Things are more complex when we consider concurrent systems where the fact that two transitions are enabled in a given marking does not necessary mean that we have to choose which one to fire even if they share some input places. Formally, we have a situation of conflict when the set of transitions enabled in a given marking is not a step. A *step* S is a multi-set of transitions that are enabled to concurrently fire in the same marking. An intriguing situation arises when different interleaved firings of the members of a step may either yield conflict situations or not. This phenomenon is known as *confusion* and is depicted by the conflicts of Fig. 5 where we can recognize that t_i and t_k are a step such that, when t_k fires no effects are felt by transition t_i, while the same is not true in the other case. Confusion is an important notion because it highlights the fact that, in terms of event ordering, the system behavior is not completely defined: this underspecification could be due either to a deliberate modeling choice (the chosen abstraction level does not include any information

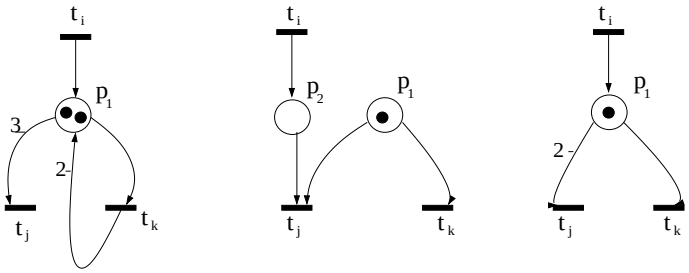

Fig. 5. Examples of confusion produced by transitions t_i and t_k [8]

on the ordering of events, hence the model analysis must investigate the effect of pursuing any possible ordering) or to a modeling error.

Let's consider the following example (Fig. 6) where we have the presence of confused subnets of immediate transitions within a GSPN and assigned weights to immediate transitions.

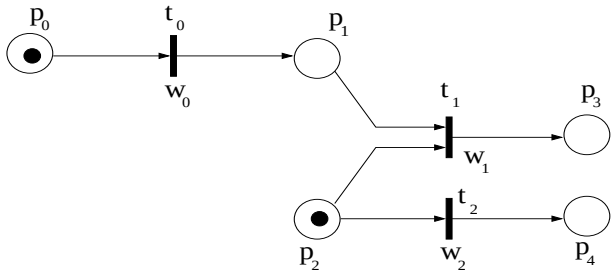

Fig. 6. Example of a simple confused GSPN model [8]

Given the initial marking $m_0 = (p_0 + p_2)$, markings $m_1 = p_3$ and $m_2 = (p_1 + p_4)$ are reached with total probabilities

$$P\{m_0, m_1\} = \frac{w_0}{w_0 + w_2} \cdot \frac{w_1}{w_2 + w_1}$$

$$P\{m_0, m_2\} = \frac{w_2}{w_0 + w_2} \cdot \frac{w_0}{w_0 + w_2} \cdot \frac{w_2}{w_1 + w_2}.$$

From these expressions we can see that, although the picture suggests transitions t_0 and t_2 as concurrent, and transitions t_1 and t_2 as members of a conflict set, the first choice of firing either t_0 or t_2 is actually crucial for the possibility of reaching the desired markings. In order to see it let's assume for example that we do not have assigned a weight w_0. Then there is a nondeterministic choice between transition t_0 and t_2 at the beginning. Firing transition t_2 first implies that only t_0 can be fired afterwards, while firing transition t_0 first results in a

probabilistic choice which is solved by assigned weights w_1 and w_2. So as one can see, the initial situation between t_0 and t_2 is critical and this is the reason why the weight w_0 is introduced. In this case the value assigned by the analyst to w_0 becomes fundamental for the quantitative evaluation of the model.

Extended conflict set. A different approach is to use an *extended conflict set* (ECS) [14] (in short, it can be explained as partitioning the set of immediate transitions into equivalence classes such that transitions of the same partition may be in conflict among each other in possible markings of the net, while transitions of different ECSs behave in truly concurrent manner). It appears with the surmise that actually, the selection of the transition to be fired is relevant only in those cases in which a conflict must be resolved: if the enabled transitions are truly concurrent, they can be fired in any order. For this reason, GSPNs associate *weights* with immediate transitions belonging to the same conflict set [8]. Thus weights on transitions are used only to choose among transitions within the ECS. This is an attempt to reduce the complexity of assigning weights to immediate transitions. The order in which immediate transitions of different ECSs fire may not matter; this means that the firing of a transition in one ECS may not affect the enabling or firing rule of a transition in a different ECS.

Priority. And now let's consider an example showing us how we can treat nondeterminism using *priority*. The introduction of a priority structure may force a deterministic ordering of conflict resolutions that removes confusion.

For instance, let us consider again example depicted in Fig. 6, showing us how the introduction of a priority structure can avoid confusion [8]. Assume first that all three transitions t_0, t_1 and t_2 have the same priority level. A confusion situation arises due to the fact that both sequences $\sigma = t_2, t_0$ and $\sigma^{'} = t_0, t_2$ are fireable in marking $m = P_0 + P_2$, and that they involve different conflict resolutions. By making the priority level of transition t_1 higher than that of t_0 and t_2 we remove the confusion situation since any conflict between t_1 and t_2 is always solved in favor of t_1; as a consequence the sequence $\sigma^{'} = t_0, t_2$ is not fireable in m and confusion is avoided.

4.1 Stochastic Activity Networks

Another extension of SPN is Stochastic Activity Networks (SAN)[15], [7]. A SAN is composed of places and activities (similar to transitions). Places hold tokens, as they do in Petri Nets. Activities in a SAN may be either timed or immediate. As a matter of fact a SAN is some kind of generalization of the gSPN formalism introduced earlier, only in SAN each immediate activity has a (positive) number of cases, which have probabilities assigned to them. When an activity completes, a case is chosen.

Formally, a SAN model is an 9-tuple

$$SAN = (P, A, \Pi(.), I(.), O(.), W(.), \gamma, p(.), m_0)$$

$P, \Pi(.), I(.), O(.), W(.)$ and m_0 have the same meanings. A is a finite set of activities, $\gamma : A \to \mathbb{N}$. Furthermore, p is the case distribution assignment, an assign-

ment of functions to activities such that for any activity a, $p(a) : \{1, ..., \gamma(a)\} \rightarrow [0, 1]$, $p(.)$ is a probability distribution called the case distribution of a.

Definition 4. *A* CG^{SAN} *is a tuple* $(S, \longrightarrow, \multimap, \twoheadrightarrow, F)$ *where*

- S *is a nonempty set of states, partitioned in three disjoint subsets,*
  - $\acute{S}$ *– stable states (◯),*
  - D *– decision states (△), and*
  - C *– case states (□).*
- $\longrightarrow \subset D \times S$ *is a set of immediate transitions,*
- $\multimap \subset \acute{S} \times R^+ \times S$ *is a set of timed transitions,*
- $\twoheadrightarrow \subset C \times [0, 1] \times S$ *is a set of probabilistic transitions, where for* $\forall c \in C$*, the sum of probabilities occurring* $\sum_i p_i$ *is* 1*, and*
- $F \in S$ *is the initial state.*

Similar to the remark below definition 2 note that according to the above definition, in stable (case, respectively decision) states only timed (probabilistic, respectively immediate) transitions are available. Since SANs have cases, there is no need to use weights on immediate activities, as is done in GSPN, to specify choices between different behaviors in the SAN. However, if multiple immediate activities are enabled at the same time, there may be some ambiguity as to which activity completes first. So, similar to the definition of well-specified CG^{gSPN} we introduce another one for CG^{SAN}: *if probability of going to a next stable marking is independent of the order in which immediate transitions complete, we say that the* CG^{SAN} *is well-specified.* A CG^{SAN} must be well-specified if its behavior is to be analyzed probabilistically.

Algorithms have been developed to determine whether a SAN is well-specified [16], [10] and [11]. Similar to the case of the CG^{gSPN} well-specified check, these algorithms do not always eliminate nondeterminism. We consider a similar algorithm but in the spirit of one which was offered in the preceding section. As well as in the subsection 3.1 one restriction here is imposed that no cycles are allowed among case and decision states. The WSC algorithm now needs to deal with probability distributions on sucessor states, which are represented as multisets of tuples, each consisting of states (resp. markings) and probabilities

$$\{\!|(s_1, p_1), ..., (s_n, p_n)|\!\}$$

satisfying that the sum of probabilities occurring $(\sum_1^n p_i)$ is 1. Multiple distributions are stored as sets of such multisets.

Exploring the SAN: start with initial stable marking m'
Explored={}
Unexplored={m'}
while {*Unexplored* nonempty} {
    take a marking m from *Unexplored*, move it from *Unexplored* to *Explored*.
    for each transition t enabled in marking m {
        fire t, this leads to marking n;
        if (t is a timed transition) {

```
            if (WSC(n) contains just one multiset k) {
              for each (s_i, p_i) in k {
                  add m --W(t)*p_i--▯--> s_i
                  if (s_i not in Unexplored, and s_i not in Explored) {
                    add s_i to Unexplored
                  }
              }
            }
            else {
              add m --W(t)--▯--> n to the graph;
              for each multiset k in WSC(n) {
                  add n ⟶ k to the graph.
              }
              for each (s_i, p_i) in k {
                  add k --p_i--> s_i
                  if (s_i not in Unexplored, and s_i not in Explored) {
                    add s_i to Unexplored
                  }
              }
            }
          }
      }
}
```

where $WSC(n)$ is defined in the way described below. We use $p{*}\{\!|(s_1,p_1),\ldots,(s_n,p_n)|\!\}$ to denote $\{\!|(s_1,p{*}p_1),\ldots,(s_n,p{*}p_n)|\!\}$. If $A=\{a_1,\ldots,a_m\}$ and $B=\{b_1,b_k\}$ are sets of such multisets, then $\mathit{shuffle}(A,B)=\{a_1\oplus b_1,\ldots,a_i\oplus b_j,\ldots,a_m\oplus b_k\}$, where $\oplus$ denotes multiset union.

```
WSC(state n) {
        Result= {{|}};
        if n is a stable state {
            Result={{|(n, 1)|}}
        }
        else {
          if n is a decision state{
            for each immediate successor marking h of n {
                add the elements of WSC(h) to Result;
                }
          }
          else {        (n is a case state)
            for each probabilistic transition n --p--> h {
                Result = shuffle(Result, p*WSC(h))
            }
```

```
        }
    }
    return normalize(Result);
}
```

Here, $normalize(\{\!|(s_1, p_1), \ldots, (s_n, p_n)|\!\})$ is a function that *normalizes* a multiset into a set $\{(s_1, p'_1), \ldots, (s_m, p'_m)\}$, such that each state s occurs at most once in a pair (s, p) in this set, and p is the cumulative probability for the multiset elements of the form $(s, .)$. For instance, $normalize(\{\!|(s, 0.3), (s, 0.3), (s'.0.3), (s', 0.1)|\!\}) = \{(s, 0.6), (s', 0.4)\}$. Function *normalize* is applied elementwise to a set of multisets.

As well as in the section 3.1 the states added to *Unexplored* are stable. Also the initial marking m' is stable. Therefore, the first **if** clause in the main algorithm is always fulfilled.

Let's consider the application WSC(state n) on the concrete example depicted on the Fig. 7.

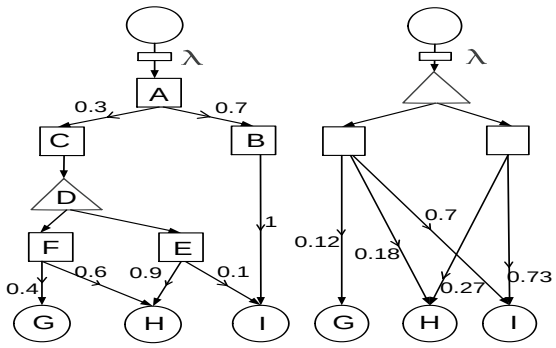

Fig. 7. Examples of CG^{SAN} and the application of WSC algorithm correspondingly

To allow references to states in this CG^{SAN}, we label them with latin letters from A to I. If we apply the main algorithm to the topmost state of CG^{SAN} in this figure, we obtain the result depicted on the right side of Fig. 7. In the sequel, we explain its construction step by step. To compute $WSC(A)$, we recursively descend the graph of case and decision nodes, until reaching the stable states G, H, and I. We have $WSC(G) = \{\{\!|(G, 1)|\!\}\}$, $WSC(H) = \{\{\!|(H, 1)|\!\}\}$, from which we construct $WSC(F)$ as follows. In the construction of $WSC(F)$, *Result* initially is $\{\{\!|\}\!\}$. Since F is a case state, we scan its probabilistic transition . We start with, say $F \xrightarrow{0.4} G$. We get

$$Result = shuffle\{\ \{\{\!|\}\!\}, 0.4 * \{\{\!|(G, 1)|\!\}\} \tag{1}$$

which results in { {|(G,0.4) |}} . Adding transition $F \xrightarrow{0.6} H$ leads to $WSC(F) = \{\{\!|(G, 0.4), (H, 0.6)|\!\}\}$.

Analogously $WSC(E)$ is given by $WSC(E) = \{\{\!|(I, 0.1), (H, 0.9)|\!\}\}$, since $WSC(I) = \{\{\!|(I, 1)|\!\}\}$. According to the algorithm $WSC(D)$ is constructed by adding the elements of $WSC(E)$ and $WSC(F)$ to *Result*, which was empty initially, resulting in

$$WSC(D) = \{\{\!|(G, 0.4), (H, 0.6)|\!\}, \{\!|(I, 0.1), (H, 0.9)|\!\}\}.$$

By virtue of having only one probabilistic transition $WSC(B)$ is expressed easily as $WSC(B) = \{\{\!|(I, 1)|\!\}\}$, and similarly $WSC(C) = WSC(D)$. Now, $WSC(A)$ is constructed. In this construction *Result* is initially empty, and, since A is a case state, we scan its probabilistic transitions. We consider $A \xrightarrow{0.3} C$ first, and obtain

$$\begin{aligned} Result = shuffle(\{\{\!|\}\!\}, 0.3 * WSC(D)) = \\ = shuffle(\{\{\!|\}\!\}, \{\{\!|(I, 0.03), (H, 0.27)|\!\}, \{\!|(G, 0.12), (H, 0.18)|\!\}\}), \end{aligned}$$

which reduces to (using the definition of *shuffle*)

$$Result = \{\{\!|(I, 0.03), (H, 0.27)|\!\}, \{\!|(G, 0.12), (H, 0.18)|\!\}\}.$$

After considering the second probabilistic transition $A \xrightarrow{0.7} B$ we get

$$\begin{aligned} Result = shuffle(Result, 0.7 * WSC(B)) = \\ = shuffle(Result, \{\{\!|(I, 0.7)|\!\}\}). \end{aligned}$$

After expanding $shuffle$, this becomes

$$Result = \{\{\!|(I, 0.03), (H, 0.27), (I, 0.7)|\!\}, \{\!|(G, 0.12), (H, 0.18), (I, 0.7)|\!\}\}.$$

So, finally after applying function *normalize* we obtain

$$WSC(A) = \{\{\!|(I, 0.73), (H, 0.27)|\!\}, \{\!|(G, 0.12), (H, 0.18), (I, 0.7)|\!\}\}.$$

In this example $WSC(A)$ returns a set of distributions, and this is used in the main algorithm to construct the example depicted on the right side of Fig. 7.

In a nutshell, $WSC(n)$ goes through all the case and decision states, and drives all distributions into a flat set of distributions. For the merging of probabilities the operation of multiplication $p * WSC(h))$ is used as well as function *normalize*, where we collect our probabilities for the tuples which have the same states. Operation *shuffle* helps us to preserve nondeterminism.

If the resulting set of distributions is just a singleton, we can say that this is a well-specified situation, because no nondeterminism persists. If this is the case for the entire graph, the resulting CG is a Markov chain. Here we bridge to the original definition of the algorithm [16].

This algorithms require the searching of all paths from a stable state to the set of next stable states. This makes the algorithm computationally complex, requiring an exponential worst time in the number of transitions in the configuration graph [11].

5 Discussion

In this paper we have discussed the notion nondeterminism in general and particularly in stochastic modeling. From a performance analysis perspective, the issue of nondeterminism may seem as two sides of the same coin. The presence of nondeterminism is useful to model different important concepts, including implementation freedom, scheduling freedom, and influence of an external environment. But from another point of view it prevents to get reasonable stochastic analysis of the model under study.

Furthermore, we have demonstrated that stochastic modeling formalism can be reduced to the corresponding configuration graphs which are more convenient for the investigation. On the example of configuration graph of SPN we have showed that according to the race policy actually there is no nondeterminism. In this the case configuration graph turns to be out a CTMC, ready for applying probabilistic analysis. On the contrary, nondeterminism arises when we consider already generalized stochastic modeling introducing immediate transitions.

In the section 'Generalized SPNs' algorithms for mapping stochastic models onto configuration graph are introduced. Besides, here two approaches - well-specified check and weak bisimulation - for eliminating nondeterminism are reviewed and related. We concluded that weak bisimulation is stronger than the well-specified check. But at the same time we would like to emphasize that the well-specified algorithm is able to eliminate nondeterminism on-the-fly, while for weak bisimulation we have to know the entire graph in advance.

In the last section we have focused on other extensions of SPN. Here we have discussed weights, priorities and case nodes as a pragmatic ways to get rid of the nondeterminism. But as it has been shown there is still the problem of confusion and well-specifiedness. We have discussed how the WSC algorithm extends to case probabilities. A similar extension for the weak bisimulation setting has not been developed yet.

Our investigations have been driven by the desire to obtain a CTMC for a large class of models involving nondeterminism. Of course, another option is to accept nondeterminism in the analysis, and to transform the model to a Continuos Time Markov Decision Processes (CTMDP) where the role of uncertain choice is played by *policy*.

Acknowledgment The author is grateful to his supervisor Holger Hermanns who directed this research, provided valuable ideas and carefully read all writings.

References

1. R.A. Howard. *Dynamic Probabilistic Systems.* John Wiley & Sons, 1971.
2. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
3. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.

4. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
5. M. Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.
6. M. Stoelinga. *Verification of Probabilistic, Real-time and Parametric Systems*. PhD thesis, Nijmegen Univeristy, 2002.
7. W.H. Sanders and J.F. Meyer. Stochastic Activity Networks. Formal Definitions and Concepts. In E. Brinksma, H. Hermanns and J.-P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis: First EEF Summer School on Trends in Computer Science*, Berg en Dal, The Netherlands, July 3-7, 2000, volume 2090 of *Lecture Notes in Computer Science*, Springer, 2001.
8. G. Balbo. Introduction to Stochastic Petri Nets. In E. Brinksma, H. Hermanns and J.-P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis: First EEF Summer School on Trends in Computer Science*, Berg en Dal, The Netherlands, July 3-7, 2000, volume 2090 of *Lecture Notes in Computer Science*, Springer, 2001.
9. H. Hermanns. Interactive Markov Chains: and the Quest for Quantified Quality. Vol. 2428 of *Lecture Notes in Computer Science*, Springer, 2002.
10. G. Ciardo, and R. Zijal. Well-defined stochastic Petri Nets. *In Proc. 4th International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'96)*, pages 278-284, San Jose, California, Feb. 1996.
11. D.D. Deavours, and W.H. Sanders. An Efficient Well-Specified Check. *In Proceedings of PNPM '99: the 8th International Workshop on Petri Nets and Performance Models*, Zarogoza, Spain, September 8-10, 1999.
12. M.K. Molloy. *On the integration of delay and throughput measures in distributed processing models*. PhD thesis, UCLA, Los Angeles, CA, 1981.
13. G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri Net. *IEEE Trans. Softw. Eng.,* 20(7):506-515, July 1994.
14. G. Chiola, M.A. Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Trans. Softw. Eng.,* 19:89-107, Feb. 1993.
15. J.F. Meyer, A. Movaghar, and W.H. Sanders. Stochastic Activity Networks: Structure, Behavior, and application. In *Proc. International Workshop on Timed Petri Nets*, pages 106-115, Torino, Italy, July 1985.
16. M.A. Qureshi, W.H. Sanders, A.P.A. van Moorsel, and R. German. Algorithms for the generation of state-level representations of stochastic activity networks with general reward structures. In *Sixth International Workshop on Petri Nets and Performance Models*, pages 180-190, Durham, NC, Oct. 1995.