Issues in Verification

0.1 Introduction

The diversity of the verification methods, developed for Petri nets and their extensions, may be confusing for the engineer trying to choose the adequate techniques to solve his problem. This chapter aims at clarifying the bases of such a choice by discussing some general issues involved in the design and the application of a verification method :

- the net models that the method enables to verify,
- the kind of properties to be checked,
- the families of methods,
- the interplay of different methods.

On the one hand a net model with a highly expressive power such as a coloured Petri net enables to handle complex systems. On the other hand, this expressive power implies difficulties for the verification process (e.g. increasing complexity, semi-decidability, restrictive kind of properties, etc.). Independently, some high-level models enlarge the range of the results by introducing a parametrization (e.g. abstract data types, variable cardinalities of domains, etc.).

The specification of the properties must address the following question. How to define a good behaviour of a net ? Among the different answers, one can suggest :

- A family of properties expressing the behaviour of the net independently of its interpretation (e.g. liveness, boundedness, etc.),
- A language of properties adapted to the dynamic systems and especially the concurrent ones (e.g. linear time logic, branching time logic, etc.),
- The behaviour equivalence with another net modelling, for instance a more abstract view of the system (e.g. bisimulation),
- The response to a sequence of tests (e.g. failure tests, acceptance tests, etc.)

The methods may be classified according to basic criteria. What kind of nets is supported by the method ? Does the method work at the structural level (i.e. the net description) or at the behavioural level (i.e. the reachability description)? Is the verification process entirely or partially automatic ? What kinds of properties is the method able to check ?

At last, to combine the different verification methods, it is necessary to understand what benefits one method can take from the results of another. Furthermore, it may also have an impact on the specification process: for instance the system can be modeled with a very abstract net and exhaustive results, then refined while keeping at the same time as many previous results as possible.

The rest of the chapter is organized as follows : in section 0.2 we present a classification of net models and especially of coloured nets then in section 0.3 we discuss the kind of properties one can check. In section 0.4 we list and provide details about the criteria of each method and we show how to combine them in section 0.5. We conclude the chapter in section 0.6 by an overview of the methods presented in this part of the book.

0.2 Classification of nets

From the model of P/T-nets, one can derive new models in different ways : restriction, extension, abbreviation, parametrization. In this section, we discuss the impact of these derivations on the verification methods.

0.2.1 Restriction of nets

The most meaningful restrictions from the point of view of verification rely on the conflicts between transitions. For instance, the well-known model of free-choice nets [Bes87] restricts the conflicts between transitions with the same input places. The impact of such a restriction is twofold: new algorithms with reduced complexity may be developed to check properties and an equivalence between structural properties and behavioural properties may be established. A characteristic property which has been defined by Commoner states that any siphon must contain a marked trap (see section 17.4 for more explanations). Thus a free-choice net is live if and only if it fulfills the Commoner property.

0.2.2 Extension of nets

A net model is an extension of the P/T-net model if its expressive power is strictly greater than the original one. The first extensions proposed for P/T-nets aim at giving more flexibility in the design process [CDF91b], [LC94]. The inhibitor arcs model the zero test, the transition priorities model, for instance, the interruption mechanism, the flush arcs model, for instance, the crash of a machine, etc.

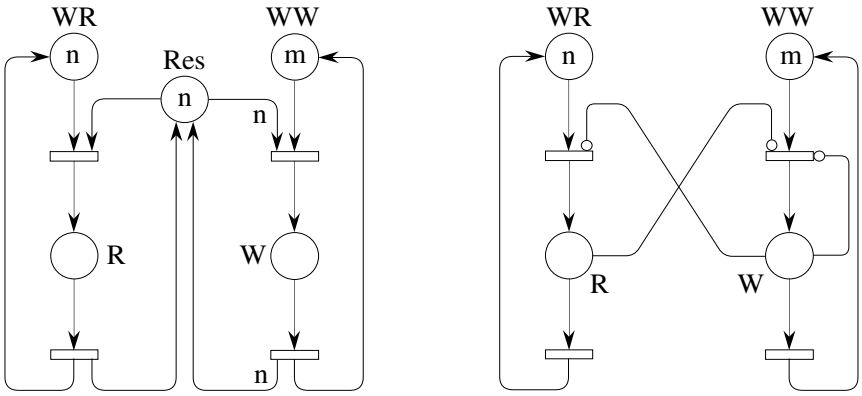

Figure 0.1: *Two Reader-Writer nets*

In most cases, the new model has the same expressive power as the Turing machines thus the reachability problem - is one marking reachable from another one ? - becomes indecidable [Hac75]. However this drawback should not be overestimated:

- at first, many structural methods which produce results will remain unchanged. For instance the computation of the flows is unaffected by the presence of inhibitor arcs since they do not induce movement of tokens,
- the hypothesis of bounded nets required for many state-based methods transforms the extension into an abbreviation. For instance the inhibitor arcs can be modeled by the method of complementary places,
- often, the modification brought to the verification methods is easy to develop and straightforward. For instance the computing of the (extended) conflict sets between transitions handles the inhibitor arcs in an intuitive and natural way.

Nevertheless there is a pernicious effect of the extensions on the design process. Let us look at the two nets of figure 0.1. These two nets model the concurrent accesses to a file by readers and writers. The safety property is evenly ensured in both cases. However the computation of the flows will give us this property directly in the first net while it will give only information about the number of readers and writers in the second net. The key point is that the more the extensions are involved in the design process, the less the classical methods will give significant results. As heuristic principle, one can state "just use extensions if necessary".

0.2.3 Abbreviation of nets

A net model is an abbreviation of a P/T-net model if :

1. there is a common semantics for the two models,
2. for any net there is a semantically equivalent P/T-net (generally of bigger size).

A useful abbreviation of P/T-net is the model of coloured nets introduced by K. Jensen [Jen92]. The main interests of this abbreviation are the information associated with tokens by colour and the ability to factorize activities with the help of firing instances of a transition. The unfolding is quite easy : any node (place or transition) is developed in a set of nodes indexed by the colours of its domain and the arcs are defined according to the applications of colour functions.

Given this unfolding it is not difficult to transform a verification method for P/T-nets into a method for coloured nets :

1. unfold the coloured net,
2. apply the algorithm,
3. interpret the results for the original coloured net.

However such a transformation is unsatisfactory for two main reasons : the complexity of the algorithm depends on the size of the unfolded net and it is sometimes cumbersome to interpret the results. So the main objective of a verification theory for coloured nets is to develop algorithms which do not require the unfolding of the coloured nets.

In order to avoid the unfolding of Petri nets, one is led to examine the syntax and the properties of the colour functions. However the general definition of coloured nets works at the semantic level. The easiest way to give a syntactical definition of a colour function is to represent it as an expression in which the constants denote bags of colours, the variables denote projections of colour domains and operators denote operations on functions. Then syntactical conditions on an expression provide necessary and/or sufficient conditions on the denoted function. Let us take an example : in a net reduction (called pre-agglomeration) a transition is required not to share its input places. In the coloured net, the equivalent condition is defined by :

1. the transition does not share its input places,
2. the colour functions which label the input arcs fulfill a condition called quasi-injectivity.

Rather than explaining what quasi-injectivity is, let us say that there are numerous necessary or sufficient conditions for quasi-injectivity. In the example of figure 0.2, we have decribed two coloured nets with their unfolded net. The first unfolded net does not share the input places and it can be detected directly on the expression of the coloured net as the expression is a tuple of all the variables of the transition. The second unfolded net shares the input places and it can

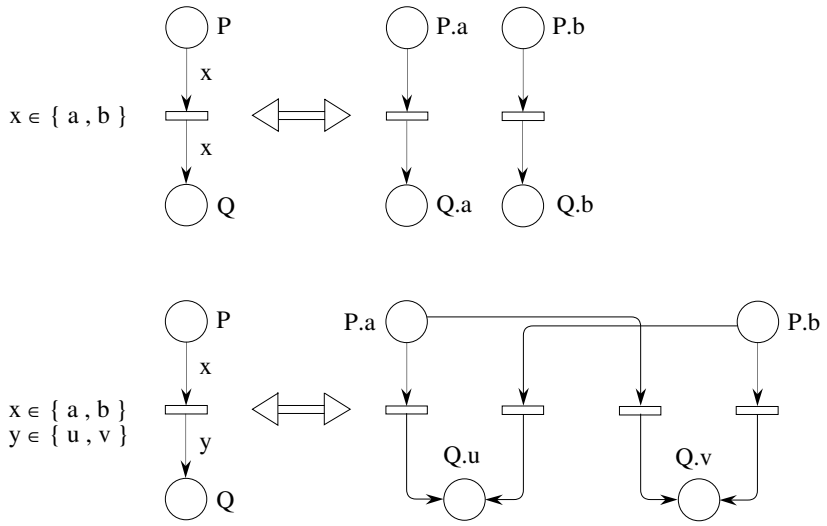

Figure 0.2: *Two coloured nets with their unfoldings*

also be detected directly on the expression of the coloured net as not all the variables of the transition appear in the expression.

Other important properties can be checked on expressions such as their algebraic structure for the flow computation or their symmetrical structure with respect to a colour domain.

0.2.4 Parametrization of nets

A net model is a parametrization of P/T-nets if it denotes a family of P/T-nets. Implicitely an unmarked P/T-net is a parametrized Petri net [CDF91a] and we can already obtain results which do not depend on the initial marking (see section 17.3). Nevertheless the parametrization of nets is very interesting in the field of coloured nets as there are many ways to achieve it, among them:

- Abstract Predicate/Transition nets [Gen88]
- Algebraic Petri nets [Rei91]
- Well-Formed Petri nets [CDFH93]

An Abstract Predicate/Transition net is associated with first-order logic and colour functions are expressions of this logic. Each interpretation of this logic provides a concrete Predicate/Transition net which is a syntaxical denotation of a coloured Petri net. The main theoretical results of this parametrization concern the existence of a normalized specification of such nets which makes it possible to decide whether two nets are semantically equivalent.

An Algebraic Petri net is associated with an abstract data type and colour functions are expressions of this abstract data type. Each algebra which fulfills this abstract data type also gives a coloured Petri net. There are various results on Algebraic Petri nets ; for instance one establishes statements such as "if the net associated with initial (or final) algebra has a property then a net associated with any algebra has the same property". The algebraic Petri nets can be easily integrated in a prototyping software environment, which is another advantage.

Well-Formed Petri nets have been introduced in order to develop efficient verification methods on coloured nets and parametrized coloured nets. The syntax of such nets relies on three basic constructions : the variables, some particular constants (the static subclasses) which denote colours with similar behaviour and one operator, the successor function, which chooses the colour "following" a colour selected by a variable. Despite its restricted syntax, it has been shown that WFNs have the same expressive power as the general coloured nets. The parametrization is introduced by the cardinalities of colour domains. Reductions and flow computations exploit the parametrization whereas the symbolic reachability graph building operates on an unparametrized WFN. Numerous applications of the symbolic graph have been developed to obtain measures of performances (steady-state probabilities, bounds, tensorial decomposition, etc.).

0.3 Properties

The choice of properties for Petri nets raises the same problem as the choice of the Petri nets model. Specifying a large set of properties forbids the development of efficient specialized algorithms whilst a restricted set of properties fails to express the various properties of protocols and/or distributed algorithms.

If one chooses to restrict the properties then these properties must be generic in the following sense: they express the behaviour of the modelized system for a large range of interpretations. Let us see how such an interpretation is possible. We give below a non exhaustive list of properties which, of course,do not cover all the general properties a net may have (see the discussion later on in this section) :

- **quasi-liveness** "Every transition is fired at least one time" expresses a syntactically correct design in the sense that any activity or event must occur at least once in the net behaviour.
- **deadlock-freeness** "There is no dead marking" means that global deadlock never happens in the system.
- **liveness** "Every transition is fired at least one time from any reachable marking" means that the system never looses its capacities at any time.
- **boundedness** "There is a bound on the marking of any place" ensures that the system will reach some stationnary behaviour on the long run. Let us note that multiple stationnary behaviours are possible.

- **home state** "There exists a marking which is reachable from any other marking" denotes the possibility for the system to reinitialize itself.
- **unavoidable state** "There exists a marking which can not be avoided indefinitely" indicates that for the system must necessarily reinitialize itself.

Despite the generality of the previous properties, there will always be some features of behaviour that will not be captured by a fixed set of properties. For instance, "The firing of t1 will eventually be followed by the firing of t2" is a useful fairness property which is not one of the previous properties. Of course, one could include it, but there are a lot of possible variants. Thus it is better to adopt a language of properties adapted to the dynamical systems and especially the concurrent ones. Among such languages, the temporal logic framework (e.g. linear time logic, branching time logic, etc.) has been widely used for Petri nets (see for instance [Bra90]). The reason for this development is twofold : most interesting properties of concurrency are expressed by simple formulas and the model checking associated with these logics can be easily transported on the reachability graph. In fact, by exploiting the structure of the Petri nets, the complexity of model checking can be reduced but we will discuss this topic later in section 0.4.

The framework of temporal logic is interesting if one wants to verify a set of properties which characterizes the desired behaviour of the modellized system. Nevertheless, starting from a global behaviour such as a set of services requires a great deal of work to specify the correct formulae. Moreover the modeller is led to build more and more complex formulae where the semantics of such formulae becomes mysterious. In such cases, it is much simpler to specify the set of services by a Petri net and to compare the behaviour of the net modelling the services with the behaviour of the net modelling the protocol. However it is necessary to define what equivalence between nets is. First, one has to distinguish between internal transitions (implementing the protocol) and external transitions (associated with the service interface). Then the projection of the protocol net language onto external transitions should be equal to the language of the service net (language equivalence). However the language equivalence does not capture the choices offered by a net upon reaching some state. Equivalence including language and choice can be defined by means of one of the numerous bisimulation definitions [BDKP91] which roughly says that whatever a Petri net can do (sequence and reached state) the other one can simulate it. The interest of the bisimulations is twofold :

- An efficient algorithm has been developed once the reachability graph of Petri nets is built.
- For models like process algebras, axiomatisation of equivalence is possible at the structural level.

Moreover Petri nets refine the definition of equivalence by distinguishing true concurrency from interleaving concurrency (see figure 0.3). Lastly (see

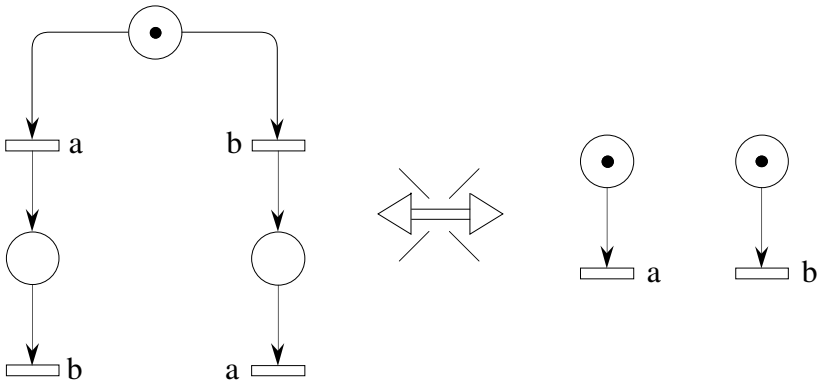

Figure 0.3: *Two nets which do not bisimulate concurrently*

section 0.6) the Petri net model can be translated into a process algebra during the design of a system in order to facilitate rewriting techniques and equational reasoning. Again it should be noted that restrictions are required in order to avoid the undecidability of bisimulations for Petri nets [Jan94].

Another possibility (and the last we examin here) is the response of a Petri net or more generally a transition system to a sequence of tests [Bri88]. A typical test application may be described as follows :

1. It starts with a specification (often a process algebra model),
2. then it generates an intermediate object called a success tree which takes into account the sequence of transitions and the choice offered by the states,
3. this tree is transformed into a transition system called the canonical tester,
4. the synchronous product of the Petri net and the canonical tester is formed,
5. the observation of deadlocks in the product provides information on the failures of the implementation given by the Petri net.

0.4 Classification of methods

There are different ways to discriminate between the methods : automatic verification versus manual verification, property verification versus property computation, specific Petri net methods versus general transition systems methods. Let us develop each of these points.

One objective of formal models is computer-aided verification. At first sight, automatic verification may appear as highly desirable. However there are some inherent limitations to automatic verification that the modeller should be aware of :

- there are numerous undecidable properties,
- even for decidable properties, checking is so complex that it may become impractical,
- automatic verification never takes into account the specificities of the modellized system.

Another advantage of manual verification is the insight it gives to the understanding of the behaviour of the system. However manual verification is prone to errors and a sound (and sometimes complete) axiomatisation of proofs may help to develop correct proofs. The duality of the manual and automatic verification should be emphasised: for instance the reduction or the abstraction of nets may be done automatically whereas refinement of nets requires the participation of the designer. Yet, these are two facets of the same theory.

The automatic method may check properties given by the modeller or simply generate valid properties of the model. Each method has its own drawbacks. Checking of properties is sometimes tricky : an inductive proof may not be obtained whereas it would have been possible to find a stronger property which is inductive. Indeed for a large class of transition systems, a property is true if and only if there is a stronger property which is inductive. On the other hand, the automatic generation of properties is generally limited in its scope : a non linear invariant will never be generated by the computation of the flows.

Different models may be employed when developing a system. Thus even if one models a system with Petri nets during the design phase, it is not obvious that the verification must be Petri net based. For instance the compositional aspect of a model is not easily exploited by Petri net techniques. Translation into a process algebra may be fruitful in this particular case. Nevertheless there are some good reasons to stick to Petri net formalisms :

- the most important techniques for other models of parallelism have been adapted for Petri nets,
- Petri net verification is one of the most flexible because of the various methods,
- some methods have no equivalents in other models (e.g. computation of the flows),
- some other methods have equivalents but their application in Petri nets is easier (partial order methods)

However the most important criterion for verification techniques depends on which aspects of Petri nets are exploited. We will list these aspects before introducing the methods based on it.

- A Petri net is a graph and the token flows must follow the arcs of this graph ; structural deadlocks are clearly based on this feature.

- A Petri net is a linear transformation of the vector of tokens and so linear algebra can take advantage of it (for instance computation of the flows).
- A Petri net underlies event structures with causality and compatibility relations. Partial order methods reduce the complexity of building the reachability graph.
- The colours of a domain often have the same behaviour. The symmetry methods also reduce the reachability graph using equivalence relations.
- The application of logics is widely used in Petri nets: for instance a logic can encode semantics of Petri nets in such a way that one obtains properties by deduction or one can build a graph of formulas where a formula naturally denotes a subset of reachable states.

Graph theory

The examination of the graph structure leads to two different and complementary families of methods which are based either on the local structure or on the global structure. The local structure of a subnet may make it possible to reason about its behaviour independently of the rest of the net. This is the key point of the reductions theory where the agglomeration of transitions corresponds to transforming a non atomic sequence of transitions into an (atomic) transition [Ber87], [Had88], [CMS87]. Even if they just simplify the net by eliminating , say a transition, their impact is considerable. Indeed in the reachability graph they eliminate all the intermediate states between the initial firing and the ending firing of the sequence. Roughly speaking, an agglomeration divides by two the reachability space and thus n agglomerations have a reduction factor of 2^n. Analysing the global structure of the net can be done by restricting the class of Petri nets and developing polynomial algorithms for the standard properties (e.g. liveness). With no restrictions on the Petri nets, similar algorithms provide necessary or sufficient conditions for the standard properties.

Linear algebra

Linear algebra techniques rely on the state change equation which claims that a reachable marking is given by the sum of the initial marking and the product of incidence matrix by the occurence vector of the firing sequence. Thus a weighting of the places which annuls the incidence matrix (i.e. a flow) is left invariant by any firing sequence. Similarly a vector of transition occurences which annuls the incidence matrix (i.e. a rhythm) keeps any marking invariant.

So there are two objectives for linear algebra techniques : computing a generative family of flows (resp. rhythms) and apply then the flows (resp. rhythms) to the analysis of the net. The computation of the flows is more or less easy depending on the constraints on flows. For instance the complexity of the computations of general flows is polynomial whereas unfortunately the computation of positive flows is not polynomial [KJ87]. However positive flows are often more useful than general flows and researchers have produced heuristics to decrease

the average complexity [CS89]. In P/T-nets, algorithms are now well known. The applications of flows and rhythms are numerous : they help to define reductions, they characterize a superset of the reachable set, they gives bounds on maximal firing rates, they make it possible to compute synchronic distances between transitions, etc. Some of them are illustrated in chapter 17.

State-based methods

Before speaking about partial-order and colour analysis methods, we must point out that one common objective of these two methods is to reduce the complexity of the state-based methods. As these latter methods are with simulation assuredly the most widely used ones , it is important to give an insight of what the different ways to cope with the space complexity of the state graph are. There are two ways to do so : to manage the graph construction or to build another graph.

An efficient management of the graph construction has an important advantage. It is independent of the structural model which generates the graph and thus can be applied to Petri nets, process algebra , etc. The two main methods of this kind are the Binary Decision Diagramm and on-the-fly verification.

Binary Decision Diagramm

Originally the BBD technique was defined to compress the representation of boolean expressions [Ake78]. Any boolean expression is represented by a rooted acyclic graph where non terminal nodes are variables of the expression with two successors (depending on the valuation of the variables) and there are two terminal nodes (true and false). In order to evaluate an expression one follows the graph from the root to a terminal node choosing a successor with respect to the chosen affectation. As subexpressions occuring more than once in the expression are factorized, the gain may be very important.

The application of the BDD technique to graph reduction relies on the representation of a node by a bit vector and the representation of the arc relation by an expression composed of variables denoting the bits of the vectors. It can be shown that the formula of modal logics can also be represented in this way and lastly that the building of the graph and the property checking can be reduced to operations on BDDs. In a famous paper, this technique has been employed to encode graphs with 10^{20} states [BCM+90]. A drawback of the method is that it is impossible to predict the compression factor even roughly.

On-The-Fly verification

The on-the-fly technique is based on two ideas : state properties can be checked on each state independently and in a finite state graph there is no infinite path with different states. Thus one does not build the entire graph but instead develops the elementary branches of this graph. The only memory required is what is required for the longest elementary path of the graph [Hol87]. In the worst case there is no gain but on average case the gain is important.

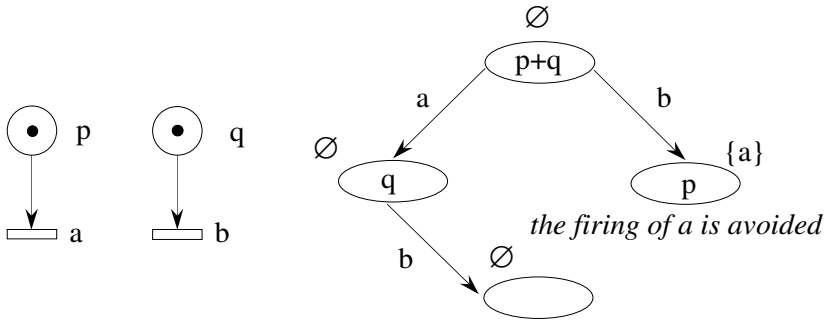

Figure 0.4: *Application of the sleep set method*

Moreover, the technique can be extended to check the properties of temporal logics [JJ89]. There the trick is to dynamically develop the product of the state graph with an automaton (say for instance a Buchi automaton for LTL formula) and check for particular states [CVWY90].

What is quite interesting with this method is its adaptation to the memory space of the machine. Indeed one can add a cache of states which remembers a number of states which are not on the current path, thus reducing the development of the branch if a cache state is encountered. Another fruitful aspect of this method is that it can be combined with other reduction methods (for instance the partial order method discussed below).

Partial order methods

The partial order methods lie on structural criteria to reduce the state graph and are efficiently implemented on Petri nets. The two main methods - sleep set and stubborn set - associate a set of transitions to a state reached during the building and use this set to restrict further developments of the graph. These sets of transitions are based on a basic structural (or possibly marking-dependent) relation between transitions. Two transitions are independent if their firings are not mutually exclusive. The independence property is structural if the precondition sets do not intersect whereas it is marking dependent if the bag sum of the preconditions do not exceed the current marking.

The sleep sets method keeps track in a reached marking of independent transitions fired in other branches of the graph [God90]. The method ensures that if one fires a transition of this (sleep) set, one encouters an already reached marking. Thus the sleep sets method "cuts" arcs on the reachability graph but the number of states is left unchanged. Figure 0.4 illustrates such a process.

Given a marking, a stubborn set of transitions is such that any sequence built with other transitions includes only independent transitions with respect to the stubborn set [Val89]. Note that if the independency relation is marking-dependent then the independency must be fulfilled at the different firing markings. Then it can be shown that restricting the firing of enabled transition being

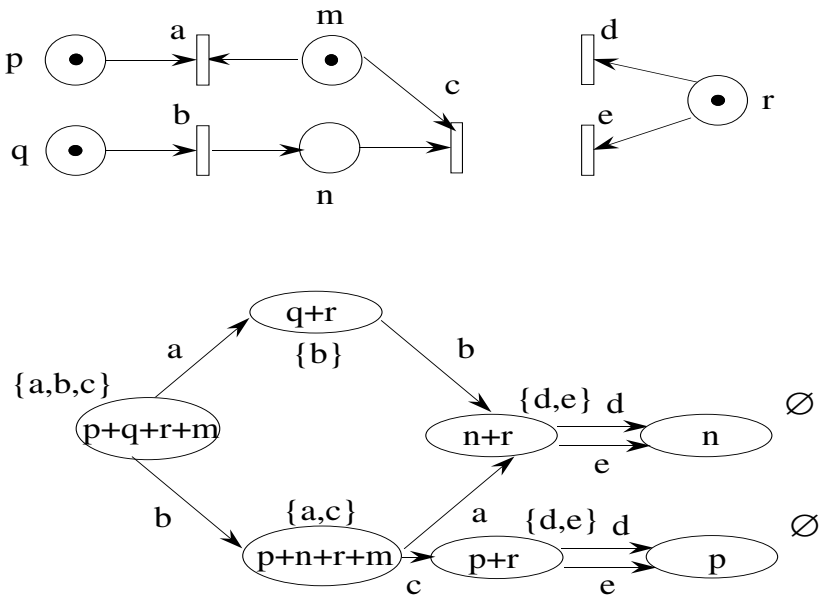

Figure 0.5: *Application of the stubborn set method*

in any stubborn set preserves the possibility of the other firing sequences. The building of the reduced graph is similar to the ordinary one except that:

- once a state is examined, the algorithm computes a stubborn set of transitions including at least one enabled transition (if the marking is not a deadlock),
- the successors of the state are the ones reached by the enabled transitions of the stubborn set.

An interesting consequence is the deadlock equivalence between the reduced graph and the original graph. Figure 0.5 illustrates such a process. Let us note that the initial stubborn set is {a,b,c} since starting from a one must include c and then b. Another possible stubborn set would have been {d,e}. The attentive reader will have noticed that an arc building would have been avoided by combining stubborn sets with sleep sets.

The stubborn set method requires more computations than the sleep set method since there is no incremental computation of the stubborn set and the computation includes disabled transitions. On the other hand, the reduction factor is often more important as here states are pruned. Nevertheless, the combination of the two methods is straightforward, thus improving the reduction factor [GW91]. What is more difficult to obtain is a large equivalence of properties between the reduced graph and the original one. Safety properties may be obtained if the property is taken into account during the building process. The

handling of general liveness properties is not possible and one must restrict to the checking of special liveness properties [Val93].

A third partial-order method is based on unfoldings of Petri nets. An unfolding of a Petri net is an acyclic Petri net where places represent tokens of the markings of the original net and transitions represent firings of the original net. One starts with the places corresponding to the initial marking and one develops the transitions associated with the firings of every initially enabling transition linking input places to the new transition and producing (and linking) output places; then one iterates the process. Of course if the net has an inifinite sequence the unfolding would be infinite and thus this unfolding must be 'cut'. In order to produce finite unfoldings, different cut methods have been proposed [McM92] [ERW96]. The unfolded net is a very compact representation of the reachability set and thus safeness properties can be checked with a low order of space complexity (time complexity may also be reduced but not so significantly). Recently the method has been extended to support linear temporal logic verification [CP96]. The principle is to build a graph of unfolded nets where the relevant transitions for the property are always graph transitions.

Colour structure analysis

The colour structure analysis has many theoretical applications. Here we just mention three theoretical developments (which will be developed in more details in chapter 16 and chapter 17). The first important point that should be emphasized is that a theoretical development may be applied on coloured nets and/or on parametrized coloured nets. As discussed before, parametrization is better for the modeller's point of view but more difficult for the researcher's point of view. Moreover, there are two ways to obtain results for parametrized coloured nets: first develop a theory for unparametrized coloured nets and then adapt the conditions to include the parametrization and restrict the kind of parametrization to develop a particular theory.

The reduction theory for coloured nets is based on the following principle to develop a sound reduction rule :

1. Take a reduction for ordinary Petri net.

2. Add coloured conditions to the structural conditions (i.e. conditions on the colour functions valuating the arcs) ; these coloured conditions are as weakest as possible to ensure the structural conditions on the unfolded net for a set of reductions.

3. Check that there is a possible ordering of the set of reductions in the unfolded net.

4. Define the transformation by a structural transformation similar to the original reduction with complementary coloured transformations ; this transformation must correspond to the successive reductions of the unfolded net.

The parametrization of the method is more or less straightforward as coloured conditions may be ensured by syntactical conditions on expressions (see the discussion above in section 0.2)

The flow computation for coloured nets requires deeper analysis of the colour function structure. It appears that the cornerstone of the flow computation is the algebraic concept of generalized inverses. Colour functions are linear transformations on a set of bags and thus this algebraic concept is sound. Moreover, a elegant algorithm adapted from Gaussian elimination rules can be developed provided that the successive generalized inverses may be computed. The space and time complexity are dramatically reduced and the flows are represented in a compact way which allows for natural interpretation.

Unfortunately, the parametrization of this method is not possible. So researchers have looked for a different direction : colour expressions can be identified to polynomials . The idea is then to apply a Gaussian-like elimination on a ring of polynomials. The whole difficulty lies in the transformation (and the reciprocal transformation) from a colour function to a polynomial one. Some subclasses of Well-Formed nets have been successfully studied (regular nets, ordered nets) with this technique. Another way to obtain parametrized methods is not to require that the flow family be a generative family. Then simple methods can work on very general nets and give useful information anyway (even if not complete).

The symbolic reachability graph of Well-Formed nets exploits the symmetry of colour functions with respect to the firing semantics. This symmetry leads to an equivalence relation between markings and transition firings. Once canonical representation of equivalence marking (and firing transitions) classes is defined, symbolic graph building is similar to ordinary graph building. Some studies show that the comparison between the reduction factor of symmetrical methods and partial-order methods depends on the modelled system. Again these methods may be combined. Another difference between the symmetrical methods and the partial-order methods is that very general properties may be checked on the symbolic reachability graph (indeed any formula of CTL*).

Logics

Logics is the support of reasoning about nets. Often some inductive rules or schemes are defined to derive properties. There are two ways to do so.

The first one is an automatic (but semi-decidable) verification of a property. For instance a safeness property must be true at the initial state and also true afterwards. Then one begins with the initial formula and one derives successively stronger formulas using the firing rule until stability is obtained. This technique may be refined using a graph of formulas where a formula is an intensive representation of states.

The other direction is manual proofs using a proofs scheme. Such an example is given by a verification diagram which is in fact a directed acyclic graph of the

formula used to prove safeness or fairness properties. Even if the proof schemes are very detailed, the verifier needs some skills to obtain his proof.

0.5 Verification Process

The verification step is closely linked to the design process. Ideally, even the (formal) specification of which properties should be satisfied by the system has to be checked. Indeed there are examples where the protocol meets the service requirements but the service is not correctly defined ! Generally, the verification process is interleaved with the different steps of new designs. The reasons for these steps are multiple. Obviously, a new model is required if failures are detected. But if the design is incremental, then once a first step of verification is successful, the model is enriched with more details before new checkings.

The two main mechanisms for incremental design are refinement and composition. Here we focus on the consequences for the verification process. Sound refinment should be local so that the properties are kept. Nevertheless, it is clear to the skilled modeller that the hypothesis of locality must be formalized in order to preserve properties. The composition aims at combining results already obtained on the components. A lot of work has been done in this respect but considering the difficulties encountered by the theoretical research we have to impose some restrictions:

- a process must have restricted choices between synchronisation actions and local actions,
- the behaviour of a synchronization must not - or only slightly - depend on previous synchronisations.

It should be noted that refinement and composition have their counterparts in the area of verification, namely, reduction and decomposition. These two aspects are very similar, however, there are particularities for the latter which are:

- the reduction (resp. decomposition) process should be automatic,
- the choice between reductions (resp. decomposition) is proof-oriented,
- if the congruence of reductions (resp. decomposition) rules is ensured then the order of application is irrelevant and this is a great complexity reduction factor.

Here are some hints on how the different verification methods should be ordered. Starting from a Petri net, applying automatic structural methods has numerous advantages :

- It points out what was implicitely in the modelling. For instance the process decomposition and message traffic are often described as flows of the net.

- It quickly discovers modelling bugs such as an unmarked structural deadlock.
- The established properties can dramatically alleviate a deductive proof.
- Lastly it helps the modeller to choose the next verification methods. As a simple illustration, positive flows covering all the places of the net ensure the success of state-based methods and also give an upper bound on the size of the states space.

Once the structural methods have been fully exploited, the modeller can use the state-based methods. As said before, it may happen that the state space is too big to be generated. However even in this case, the modeller have some alternatives :

- Classically, he can always simulate his net and the consequences are twofold: negative properties are shown and long runs without trouble develop confidence in the model,
- He can do on-the-fly verification which, even though it takes much longer, can check all the important properties of the net. Here, the key point is that the complexity space is related to the longest simple path in the graph.
- He can generate a smaller object equivalent to the state graph with respect to some properties. Partial-order methods and symmetry methods typically produce such objects.

Alternatively deduction methods avoid the space complexity problem. It does not mean that these methods should be used after the other methods have failed. Indeed, if the modeller has a clear idea of how his model fulfills its properties, he often develops a quick deductive proof of its correctness. Examples are numerous in distributed algorithms conception.

Lastly it should be pointed out that the deductive methods can be employed in the design process by means of system synthesis. Indeed, if the specification is a formula and the semantic models of the involved logic are the behaviour of the nets, then the system synthesis can be based on the satisfiability resolution. Using this resolution, one begins to produce a semantic model, then one folds the semantic model in order to obtain a Petri net. The first step is often possible with modal logics (they verify the small model property), but the second step is technically and sometimes theoretically difficult. At the current time, it is an open field of research.

0.6 Overview

Chapter 16 describes state-based methods. It develops some of the techniques we have presented above. First it shows how the computation of state space

may be managed efficiently. Then it introduces more precisely the partial order methods, symmetries and modular methods. There is more development on the use of symmetries including the implementation for Well-formed coloured nets where this building can be completely formalized. This part ends with comparing these methods according to different criteria such as space and time reduction , property equivalence and how these different methods may be combined.

The rest of the chapter takes into account the kind of properties that can be checked with the impact on the graph building. An original technique of parametrized building is developed including the verification of temporal logics. Lastly, the problem of model checking is discussed as a whole.

In chapter 17, the structural methods are developed. Some accurate reduction rules are presented with special emphasis on the implicit place. The implicit place has a particular role as it simplifies the structure of the net and makes it possible to apply other techniques more efficiently. Moreover implicit places have a strong connexion with positive flow computation as shown in the chapter. The linear algebraic techniques are then developed and the equivalence between behavioural properties and linear algebraic results is pointed out. Then siphons and traps are carefully studied as they are the cornerstone of necessary and sufficient conditions for liveness properties.

In the last part of the chapter, some syntactical subclasses are defined showing what behavioural consequences can be established from the syntactical restrictions. The behavioural properties include fairness, liveness, deadlock-freeness and the relation between reachable states and linear invariants.

Chapter 18 presents new techniques which covers an open field of research. The first technique is based on linear logic. The interest of linear logic is twofold: it provides Petri nets with an operational semantics and a proof scheme for linear logic gives the proof of a property in the corresponding Petri net.

The second technique helps to understand how to benefit from multi-formalisms. This technique starts from a specification of the system given in process-algebraic terms. Then it constructs a Petri net model of the system. The Petri net is simulated to show bad behaviours in order to reinforce confidence in the model. At last the Petri net is transformed again in process algebra so that the two process algebra (modelling the specification and the implementation) are equivalent. Emphasis is put on the design cycle rather than on technical aspects.

Bibliography

[Ake78] S.B. Akers. Binary decision diagrams. *IEEE transactions on Computers*, C-27(6):509–516, June 1978.

[BCM+90] J.R. Burch, E.R. Clark, K.L. McMillan, D.L. Dill, and Hwang L.J. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the* 5^{th} *IEEE Symposium on Logic in Computer Science*, 1990.

[BDKP91] E. Best, R. Devillers, A. Kiehn, and L. Pomello. Concurrent bisimulations in petri nets. *Acta Informatica*, 28:231–264, 1991.

[Ber87] G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 359–376. Springer Verlag, Bad Honnef, West Germany, February 1987.

[Bes87] E. Best. Structure theory of Petri nets: the free choice hiatus. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 168–205. Springer Verlag, Bad Honnef, West Germany, February 1987.

[Bra90] J.C. Bradfield. Proving temporal properties of petri nets. In *Proc.* 11^{th} *Intern. Conference on Application and Theory of Petri Nets*, Paris, France, June 1990.

[Bri88] E. Brinksma. A theory for the derivation of tests. In S. Aggrawal and K. Sabani, editors, *Protocol Specification Testing and Verification volume VIII*. Elsevier Science Publishers B.V., North-Holland, 1988.

[CDF91a] G. Chiola, S. Donatelli, and G. Franceschinis. On parametric P/T nets and their modelling power. In *Proc.* 12^{th} *Intern. Conference on Application and Theory of Petri Nets*, Aarhus, Denmark, June 1991.

[CDF91b] G. Chiola, S. Donatelli, and G. Franceschinis. Priorities, inhibitor arcs, and concurrency in P/T nets. In *Proc.* 12^{th} *Intern. Conference on Application and Theory of Petri Nets*, Aarhus, Denmark, June 1991.

[CDFH93] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.

[CMS87] J.M. Colom, J. Martinez, and M. Silva. Packages for validating discrete production systems modeled with Petri nets. In P. Borne and S. Tzafestas, editors, *Applied Modelling and Simulation of Technological Systems*, pages 529–536. Elsevier Science Publ. (North Holland), 1987.

[CP96] J.M. Couvreur and D. Poitrenaud. Model checking based on occurence net graph. In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX, Theory Applications and Tools, FORTE/PSTV'96*, volume 663 of *LNCS*, pages 380–395, Kaiserslautern, Germany, October 1996. Chapman-Hall.

[CS89] J.M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In *Proc. 10^{th} Intern. Conference on Application and Theory of Petri Nets*, pages 52–73, Bonn, Germany, June 1989.

[CVWY90] C. Courcourbetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In *Proceedings of Computer Aided Verification 90*, volume 30 of *DIMACS*. North-Holland, 1990.

[ERW96] J. Esparza, S. Romer, and Vogler W. An improvment of mcmillan's unfolding algorithm. In *Proceedings of the second International Workshop TACAS'96*, volume 1055 of *LNCS*, pages 87–106, Passau, Germany, March 1996. Springer-Verlag.

[Gen88] H.J. Genrich. Equivalence transformations of Pr/T nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Proceedings of Computer Aided Verification 90*, volume 30 of *DIMACS*, pages 321–340. North-Holland, 1990.

[GW91] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proceedings of Computer Aided Verification 91*, volume 575 of *LNCS*. Springer-Verlag, July 1991.

[Hac75] M. Hack. *Decidability Questions for Petri Nets*. PhD thesis, M.I.T., Cambridge, MA, December 1975. also tech. report 161, Lab. for Computer Science, June 1976.

[Had88] S. Haddad. Generalization of reduction theory to coloured nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[Hol87] G.J. Holzmann. Automated protocol validation in argos: Assertion proving and scatter searching. *IEEE transactions on Software Engineering*, 13(6):683–696, June 1987.

[Jan94] P. Jancar. Decidability questions for bisimilarity of petri nets and some related problems. In *Proc. STACS'94*, volume 775 of *LNCS*, pages 581–592, Caen, France, 1994. Springer-Verlag.

[Jen92] K. Jensen. *Coloured Petri nets. Basic concepts, analysis methods and practical use. Volume 1: Basic concepts.* Springer-Verlag, 1992.

[JJ89] C. Jard and T. Jeron. On-line model checking for finite linear temporal logic specifications. In *Proceedings of Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 189–196. Springer-Verlag, 1989.

[KJ87] C. Kruckeberg and M. Jaxy. Mathematical methods for calculating invariants in petri nets. In G. Goos and J. Hartmanis, editors, *Advances in Petri Nets 1987*, volume 266 of *LNCS*, pages 104–131. Springer-Verlag, 1987.

[LC94] C. Lakos and S. Christensen. A general systematic approach to arc extensions for coloured petri nets. In *Proc. 15^{th} Intern. Conference on Applications and Theory of Petri Nets*, volume 815 of *LNCS*, Zaragoza, Spain, 1994. Springer-Verlag.

[McM92] K.L. McMillan. On-the-fly verification with stubborn sets. In *Proceedings of Computer Aided Verification 93*, volume 663 of *LNCS*, pages 164–175, Montreal, Canada, June 1992. Springer-Verlag.

[Rei91] W. Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, 1991.

[Val89] A. Valmari. Stubborn sets for reduced space generation. In *Proc. 10^{th} Intern. Conference on Application and Theory of Petri Nets*, Bonn, Germany, June 1989.

[Val93] A. Valmari. On-the-fly verification with stubborn sets. In *Proceedings of Computer Aided Verification 93*, volume 697 of *LNCS*, pages 397–408. Springer-Verlag, 1993.