Department of Computer Science

Institute of Information Technology
and
Electrical Engineering

University of Veszprém

DIPLOMA THESIS

DEADLOCK ANALYSIS IN HIERARCHICAL PETRI NETS

Németh Erzsébet

Veszprém

Supervisor: Dr. Hangos Katalin

2002

Nyilatkozat

Alulírott Németh Erzsébet, diplomázó hallgató, kijelentem, hogy a diplomadolgozatot a Veszprémi Egyetem Számítástudomány Alkalmazása tanszékén készítettem mérnök-informatikus diploma (master of engineering in information technology) megszerzése érdekében.

Kijelentem, hogy a diplomadolgozatban foglaltak saját munkám eredményei, és csak a megadott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a diplomadolgozatban foglalt eredményeket a Veszprémi Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2002. május 13.

Németh Erzsébet

Köszönetnyilvánítás

Legelőször témavezetőmnek, Dr. Hangos Katalinnak mondok hálás köszönetet a gondos irányításért, és a sok támogatásért, amellyel munkámat, fejlődésemet segítette.

Köszönettel tartozom szüleimnek, hogy tanulmányaimat sok esztendőn át minden erejükkel segítették.

Hálás vagyok mindazoknak, akik bármilyen más módon hozzájárultak e diplomadolgozat megszületéséhez.

Summary

The subject of this diploma work is the analysis of behavioural properties in hierarchical Petri nets. As a result of the analysis we can conclude on the presence or absence of these properties. I have investigated a composite property: the existence of dead-locks in bounded Petri nets.

As the analysis of behavioural properties is a computationally hard problem, we usually structure a huge Petri net with many elements using hierarchical decomposition. This way we hope to conclude about the analysis results for the overall net from the results for the sub-nets and from the way they are composed together. The structuring obeys strict syntactical rules. It is an important technical assumption in my diploma thesis that I have assumed a two-level hierarchy for the decomposition.

In order to solve the problem in an abstract way, a formal description of the Petri net together with a formal description of the composition net is needed. The last net plays an important role in evaluating the analysis results of the Petri sub-nets.

The proposed deadlock detection algorithm is based upon the hierarchical structure and on the deadlock analysis of the sub-nets using Boolean algebra. Two basic cases have been identified in the design of the algorithms: the case of safe (1-bounded) Petri nets and the k-bounded Petri net case.

I have used Boolean variables for describing the Petri nets. The first case is considered as a basic case. In the second case I had to modify the basic case so that Boolean variables could be used. Therefore I have applied binary encoding in order to be able to use Boolean variables.

I have implemented both versions of the deadlock detection algorithm using Matlab [9].

I have analyzed the resulted algorithms and their applicability conditions.

Keywords: *hierarchical Petri nets, Boolean algebra, behavioral properties, boundedness, deadlock analysis.*

Tartalmi összefoglaló

Diplomamunkám témája hierarchikus Petri hálók viselkedési tulajdonságainak vizsgálata. A vizsgálat eredményeképpen viselkedési tulajdonságok hiányára vagy meglétére következtethetünk. Egy összetett tulajdonságot vizsgáltam: a holtpont létezését korlátos Petri hálók esetén.

Mivel nagy méretű, sok elemet tartalmazó Petri hálók analízise algoritmikusan bonyolult feladat, ezért a hálót struktúráljuk hierarchikus dekompozicióval, remélve, hogy a kisebb méretű hálók elemzési eredményeiből és az összetétel tulajdonságaiból következtethetünk az egész háló tulajdonságaira. A strukturálást szigorú szabályok betartásával végezzük. Lényeges technikai megszorítás, hogy diplomamunkámban a legegyszerűbb kétszintes hierarchiát feltételeztem.

A feladat absztrakt megoldásához szükséges a vizsgált Petri háló, valamint a hierarchikus kapcsolatot leíró háló formális leírása is, mely jelentős szerepet játszik az alhálókról rendelkezésre álló információk kiértékelésekor.

A kifejlesztett holtpont detektáló eljárás a hierarchikus szerkezeten és az alhálók Boole-algebrai eszközökkel történő holtpont analízisén alapul. Az algoritmus tervezése során két esetet különböztettem meg: az 1-korlátos (biztonságos) és a k-korlátos Petri hálókat.

Mindkét esetben Boolean változókat használtam a Petri háló leírására. Az első eset tekinthető az alapesetnek. A második esetben az alapesetet módosítani kell oly módon, hogy alkalmazhatók legyen a Boolean változók. Ezért bináris kódolást alkalmaztam, hogy áttérhessek a Boolean változók használatára.

A hierarchikus Petri háló holtpontját detektáláló algoritmus a két különböző esetre vonatkozó változatait Matlab [9] programozási nyelven implementáltam.

A kapott algoritmust elemeztem és vizsgáltam annak alkalmazhatóságát.

Kulcsszavak: *hierarchikus Petri háló, Boolean algebra, viselkedési tulajdonságok, korlátosság, holtpont vizsgálat.*

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem statement

Discrete event dynamic systems (DESs) are such time-varying systems that their variables have discrete values. The behaviours of such systems is usually determined by discrete environmental effects, for example discrete change of the values of some external signals or disturbances and/or operator's interferences. This system class includes industrial operating procedures, for example manufacturing processes in machine industry , procedures of control and security systems built up from discrete circuit elements, and so on.

A wide class of discrete event dynamic systems can be well described by low level Petri nets. This modelling technique supports the dynamic analysis, but in general case the analysis is a computationally hard problem. As the result of the analysis we conclude on the lack or presence of structural and/or behavioural properties (for example deadlock-freeness, boundedness, liveness).

The models of huge DESs containing many variables are becoming increasingly complex, which render the analysis more difficult. In order to avoid computational explosion we use model structuring methods, such as hierarchical decomposition. From mathematical point of view, hierarchical Petri nets are collection of embedded Petri nets where the embedding is driven by strict rules.

In this diploma work I analyze how one can efficiently deduce some behavioural properties (deadlock-freeness and boundedness) of the composite overall net from the behavioural properties of the component Petri nets and from the Petri net which describes the connections in the case of two-level hierarchical Petri nets. I complete the theoretical analysis with simulation tests on a freeware Petri net simulation software.

1.2 Preliminaries[2]

An important class of dynamic systems is the class of *discrete event systems* (DES). Discrete event systems are described by so called language models which are usually represented by *discrete automata*.

When we speak about DES? When the state space of a system is naturally described by a discrete set like $\{0, 1, 2, ...\}$, and state transitions are only observed at discrete points in time. We associate these state transitions with "events" and talk about a "discrete event system". DESs satisfy the following two properties:

1. Their state space is a *discrete* set.

2. The state transition mechanism is *event-driven*.

In event-driven systems, it is only the occurrence of asynchronously generated discrete events that forces instantaneous state transitions. In between event occurrences the state remains unaffected. Conventional differential equations are not suitable for describing such "discontinuous" behavior.

We would like to use *discrete event modelling formalisms* that would allow us to represent languages in a manner that highlights structural information about the system behavior and that is convient to for synthesis and analysis. Discrete event modelling formalisms can be untimed, timed, or stochastic, according to the level of abstraction of interest.

Two main discrete event modelling formalisms are automata and Petri nets. These formalisms have in common the fact that they represent languages by using a state transition structure, that is, by specifying what the possible events are in each state of the system. The formalisms differ by how they represent state information. They are also amenable to various composition operations, which allows building the discrete event model of a system from discrete event models of the system components. This makes automata and Petri nets convenient for model building.

The main elements of a DES are:

- a discrete state space, denoted by X,
- a discrete event set, denoted by E.

An automaton is a device that is capable of representing a language according to well-defined rules. The simplest way to present the notion of automaton is to consider its directed graph representation, or state transition diagram. We use the following example for this purpose.

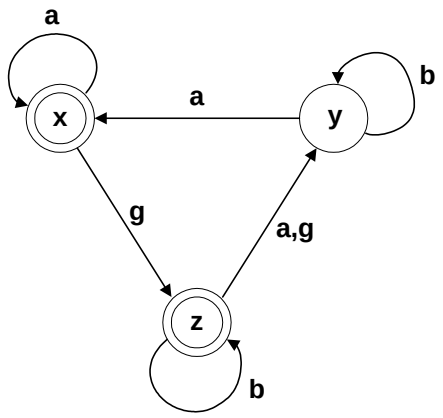


Figure 1: State transition diagram

Consider the directed graph in Fig. 1, where nodes represent *states* and labelled arcs represent *transitions* between these states. This graph provides a complete description of an automaton. The set of nodes is the state set of the automation, $X = \{x, y, z\}$. The set of labels for the transitions is the event set (alphabet) of the automaton, $E = \{a, b, g\}$. The arcs in the graph provide a graphical representation of the *transition function* of the automaton, which is denoted as $f : X \times E \to X$:

$$f(x, a) = x \qquad f(x, g) = z$$
$$f(y, a) = x \qquad f(y, b) = y$$
$$f(z, b) = z \qquad f(z, a) = f(z, g) = y$$

The notation $f(x, a) = x$ means that if the automaton is in state y, then upon the "occurrence" of event a, the automaton will make an instantaneous transition to state x.

Two more ingredients are necessary to completely define an automaton: an initial state, denoted by x_0, and a subset X_m of X that represents the states of X that are *marked*. Marked states are also referred to as "accepting" states or "final" states. The initial state is identified by an arrow pointing into it and states belonging to X_m are identified by double circles.

A *deterministic automaton*, denoted by $\mathcal{A}$, is a five-tuple

$$\mathcal{A} = (X, E, f, x_0, X_m)$$

where:

X is the set of states,
E is the finite set of events associated with the transition in $\mathcal{A}$,
$f : X \times E \to X$ is the transition function: $f(x, e) = y$ means that there is a transition labelled by event e from state x to state y; in general, f is a *partial* function on its domain
x_0 is the initial state,
$X_m \subset X$ is the set of marked states.

If X is a finite set, we call $\mathcal{A}$ a deterministic finite-state automaton (DFA).

The alternative to automata for untimed models of DES is provided by Petri nets. These models were first developed by C. A. Petri in the early 1960's. Petri nets are related to automata in the sense that they also explicitly represent the transition function of DES.

It can be proved that an automata can always be represented as a Petri net; on other hand, not all Petri nets can be represented as finite-state automata. Consequently, Petri nets describe a larger class of systems than finite automata.

Petri nets are subject of this diploma thesis, therefore their description will follow separately in the next chapter.

2 Definitions and basic properties[6, 7]

In this chapter the most important definitions are briefly summarized from the viewpoint of this diploma thesis.

2.1 The basic components of Petri nets

A Petri net consists of *places* and *transitions*, and describes the relations between them. As the names of these elements show, places refer to static parts of the modelled system while transitions refer to changes or events occurring in the system.

The mathematical representation of Petri nets consists of the sets of transitions and places, the functions describing the relations between them and a function that describes the dynamic state of the net. The graphical representation of Petri nets is a bipartite directed graph where places are drawn as circles and transitions are drawn as bars or boxes. Logical relations between transitions and places, i.e. between events and their preconditions and consequences are represented by directed arcs.

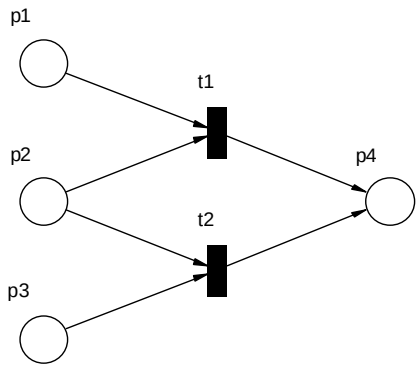


Figure 2: A simple Petri net

Transitions can be seen as the steps or substeps of operating procedures while places imply the preconditions and consequences of these steps in a controlled discrete event system.

In a complex system a consequence of an event is a precondition of other events.

We generally use the term *"condition"* instead of both precondition and consequence. In a Petri net model we use the expressions *input place* and *output place* rather than precondition and consequence if it is necessary to emphasize the relation between a place and a transition.

The validity (or occurrence) of a condition in the modelled system can be represented by the presence or absence of *tokens* in the appropriate place in the net or by nonnegative numbers (in the mathematical representation). If the condition is not valid in the real system, then there is no token in its place or the value associated to it is equivalent to zero. On the other hand, if the condition is valid, then there is a token in its place or its value is equivalent to one (or "true"). In certain cases, there can be more than one token in a given place.

In the so-called *low-level Petri nets* there is no distinction made between the tokens. This means that the items represented by the tokens in a given place are either the same or the differences between them are not relevant from the point of view of the modelling goal.

The modelling of the investigated system using places, transitions and arcs means the description of the static structure of the system. This is useful in itself because it helps the understanding of the system structure and it is also used in the analysis. At the same time, the Petri net we obtain as a result makes it possible to carry out behavioural investigations, when we want to know what will happen in the system starting from an initial state.

The *firing of transitions* in the net follows the behaviour of the real system: an even can occur if all of its preconditions are fulfilled. In the Petri net, a *transition* is called *enabled* if all of input places are valid. If a transition is enabled then it can fire. If an event occurs in the real system then the transition referring to this event must be fired in the net.

During the firing of a transition the appropriate number of tokens is removed from the input places and added to the output places. The logical relations between

places and transitions define the number of tokens to be removed or added.

2.2 The formal definition of Petri nets

A Petri net is a 4-tuple

$$N = (P, T, F, W) \tag{1}$$

where:

$P = \{p_1, ...p_m\}$ is a finite set of places,
$T = \{t_1, ...t_n\}$ is a finite set of transitions,
$F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs,
$W : F \to \{1, 2, 3, ...\}$ is a weight function,
$M_0 : P \to \mathbb{N}$ is the initial marking,
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

The input function is defined as

$$I : T \to P^{\infty}, \quad I(t_i) = \{p_j \in P | (p_j, t_i) \in F \quad and \quad j = 1, ...n_i\}, \tag{2}$$

and the output function is defined as

$$O : T \to P^{\infty}, \quad O(t_i) = \{p_j \in P | (t_i, p_j) \in F \quad and \quad j = 1, ...m_i\}. \tag{3}$$

The input function and the output function partition the set F into subsets F_I and F_O, i.e. $F_I \cup F_O = F$ and $F_I \cap F_O = \emptyset$.

We use the following symbols for a *pre-set* and *post-set* (where F is the set of all arcs):

${}^{\bullet}t = \{p | (p, t) \in F\}$ is the set of input places of t ,
$t^{\bullet} = \{p | (t, p) \in F\}$ is the set of output places of t ,
${}^{\bullet}p = \{t | (t, p) \in F\}$ is the set of input transitions of p ,
$p^{\bullet} = \{t | (p, t) \in F\}$ is the set of output transitions of p .

2.3 Markings

An arbitrary distribution of tokens on the places is called *marking*. The initial distribution of tokens on the places is called the *initial marking* and it is denoted by M_0.

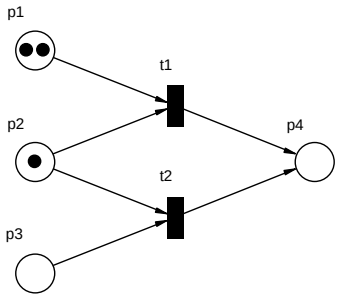


Figure 3: A Petri net with markings

The marking function gives the distribution of tokens in a given net state:

$$M : P \rightarrow N$$

A Petri net with a given initial marking is denoted by

$$PN = (P, M_0)$$

2.4 The firing of transitions

The firing rules of transitions are the following:

1. A transition t_j is said to be enabled if there is least $w(p_i, t_j)$ token on each input place p_i of t_j :

   $$M(p_i) \geq w(p_i, t_j) \quad for\ \forall p_i \in P$$

   where $w(p_i, t_j)$ is the arc weight.

2. An enabled transition may or may not fire depending on whether or not the event modelled by the transition actually takes place in the real system.

3. At firing of transition t_j the value of the marking function of a place is decreased by the weight of the arc connecting the given places place to transition

t_j, and is increased by the weight of the arc from transition t_j to the given place:

$$M''(p_i) = M'(p_i) - w(p_i, t_j) + w(t_j, p_i) \quad for\ \forall p_i \in P$$

As it can be seen, we generalized the increases and decreases for all places in the net. These operations have no effect on the marking value if there is no logical relation between the given places and transition so this generalization enables a simpler treatment of markings.

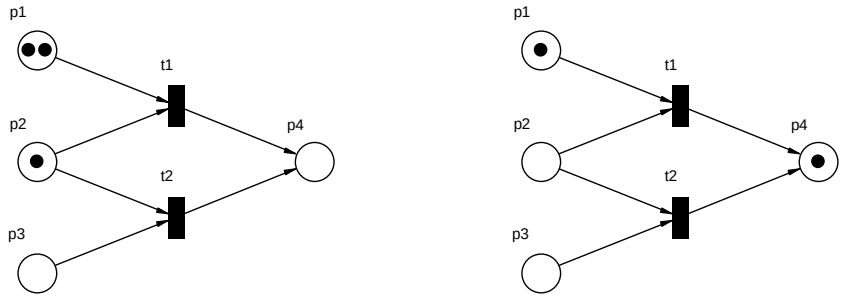


Figure 4: A Petri net before and after firing a transition

2.5 Self-loop

If a place is an input and output place of the same transition then this place-transition pair is called a *self-loop* (see Fig. 5). If the weights are the same in both cases then the firing of the transition does not change the marking value on that place. It can be proved that if the place belonging to the loop cannot be found in the input set of any other transition and the transition has no other input place then once the transition becomes enabled, it remains enabled throughout the execution of the net. A *Petri net* without self-loop is said to be *pure*.

2.6 Capacity of places

Up to this point we assumed that places can have an infinite number of tokens, i.e. there is no constraint on the marking value of a place. A net of this type is called an infinite capacity net. However, real discrete event systems are different.

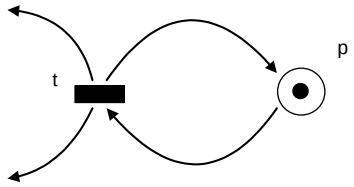


Figure 5: A self-loop

For a *finite capacity net* we have to add a new function, K, which is the capacity function to the formal definition:

$$K : P \rightarrow \mathbb{N}^+,$$

and we have to modify the rule for transitions to be enabled.

A transition t_j is said to be enabled if there is at least $w(p_i, t_j)$ token in each input place p_i of t_j :

$$M'(p_i) \geq w(p_i, t_j) \quad for\ \forall p_i \in P$$

and after the firing transition of t_j the markings in its output places does not exceed their upper limits:

$$M''(p_i) \leq K(p_i) \quad for\ \forall p_i \in P$$

where

$$M''(p_i) = M'(p_i) - w(p_i, t_j) + w(t_j, p_i)\ .$$

This modified rule is called a *strict transition rule* whereas the former rule for an infinite capacity net is the *(weak) transition rule*.

2.7 The analysis of Petri nets

A Petri net is more than a description of system structure: nets are intended to be executed. Such execution can provide information about system dynamics that could never be derived by looking at a static model and considering its implications. There is just too much information involved in the functioning of a complex system for the unaided human mind to cope with.

Petri nets can be used for modelling a large variety of systems especially those containing concurrent events.

Modelling a system and the execution of its Petri net model give a lot of information about the basic structure and processes taking place in it but the analysis also ensures provable consequences.

2.7.1 Analysis problems for Petri nets

Petri net properties can be divided into two major classes: behavioural (or marking dependent) properties and structural properties which are independent of the initial marking, i.e. initial state. Several properties can be identified and analyzed for Petri nets.

The most important Petri net properties include:

- Behavioural properties: which depend on the initial marking (these are most most interesting properties)
  - reachability
  - boundedness
  - schedulability
  - liveness
  - conservation
- Structural properties: which do not depend on the initial marking (often too restrictive)
  - consistency
  - structural boundedness
  - place and transition invariants

2.7.1.1 Safeness and boundedness For a Petri net one of the most important question is boundedness. Boundedness and its special case safeness are related to

the limited capacity of places.

A place in a Petri net is *bounded* if the number of tokens in that place never exceeds a given value. If this maximum value is equal to 1 then the place is called *safe*.

The interpretation of safeness and boundedness depends on the system to be modelled. The presence of more than one token in a place may mean many different things. For example, there can be two pumps in the system, which are represented by two tokens in a place.

The examination of boundedness and safeness can be done for a group of places or for all places in the net. If all places are safe then the net can be called a *safe Petri net*. If an upper limit k that holds for all places can be determined in the net then the net is a *k-safe Petri net*.

2.7.1.2 Conservation The conservation property is related to the changes in the sum of tokens in Petri net during execution. A Petri net is *strictly conservative* if the number of tokens is the same in all markings starting from an initial marking.

Strict conservation is a very strong property. It can be useful in the case of modelling resource allocation systems where tokens may represent the resources. It is a very natural requirement for these systems that tokens are neither created nor destroyed there.

It is possible to assign conservation weight to places and check the sum based on the linear combination of tokens computed using the place weights. In this case the *weighted* sum for all reachable markings should be constant for a *conservative Petri net*.

The investigation of conservation can be done for a subset of places, too.

2.7.1.3 Liveness Liveness addresses the question whether it is always possible to activate a specific transition or the system can reach a state where this transition

is "dead". As a generalization of this problem we can investigate whether the system can reach a state where there is no enabled transition at all. This state is called a deadlock. A system can get into a deadlock when the operating procedure is over but it could also happen that the process stops before the final state. A deadlock is very dangerous in the latter case because it refers to a state where the operator has no possibility to intervene; that is, the system is out of control.

Liveness implies the lack of deadlocks, but not viceversa.

A Petri net is *live* if and only if from any reachable state and for any transition it is possible to reach a state from which the transition is firable.

2.7.1.4 Reachability and coverability During execution different markings can be reached in a Petri net starting from a given initial marking M_0. These markings are either desirable or undesirable from the viewpoint of the operation of the modelled system. The reachability problem addresses the question whether there is a marking M'' in the *reachability set* of M_0 such that $M'' \geq M'$ i.e. M'' covers a predefined marking M''. (A marking M'' covers marking M' if $\forall i : M''_i \geq M'_i$, i.e. each component of marking M'' is greater than or equal to the components of marking M'.)

The investigation of reachability and coverability can be done for a restricted set of places, too.

2.7.1.5 Structural properties The most important structural question in the analysis of Petri nets is the determination of *place and transition invariants*. A place invariant is a set of places, in which the sum of tokens remains constant, independently of which transition fires. Tokens of this set of places are neither generated nor consumed, only "moved" between places.

A transition invariant is a set of transitions. When these transitions fire starting from an initial state then the system returns to the same initial state. Transition invariants correspond to the cyclic behaviours of the modelled system.

2.7.2 Analysis techniques

There are two major classes of Petri net analysis techniques: the construction and analysis of the reachability tree and using matrix equations.

The aim of constructing a reachability tree is to answer the initial state dependent questions. It involves the determination of all possible markings that belong to a given initial state. On the another hand matrix equations are used for determining structural properties.

Both techniques can be implemented on a computer. The use of computers is very important during the analysis because apart from some simpler cases the analysis of the above mentioned properties is very difficult without software support.

2.7.2.1 Reachability tree The reachability tree technique involves the enumeration of all reachable markings from a given initial marking. The method of constructing a reachability tree is the following. Starting from the given initial state (marking) as the root of the tree we determine the enabled transitions. Then we fire the enabled transitions to generate new markings. These new markings that will be added as new nodes to the tree. These new nodes are connected to their parent node by directed arcs, which have the colour (weight) of the fired transition. We repeat this process for every new node until there is no enabled transition. The terminal nodes of tree are the "idea" markings where there are no enabled transitions. An example is seen in Fig. 6.

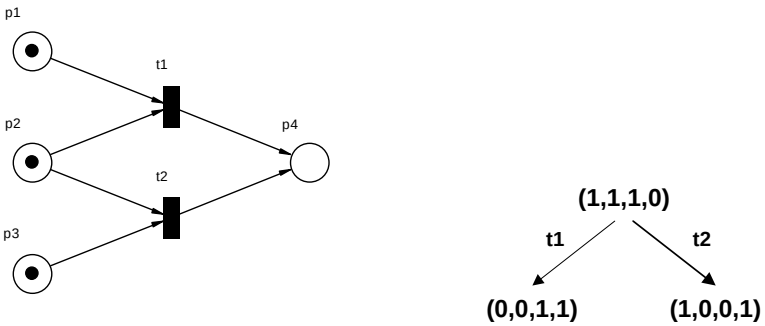


Figure 6: A Petri net and its reachability tree

It is easy to see that even simple bounded Petri nets can have an infinite reachablility tree. To avoid infinite trees we do not perform the investigation of an enabled transition if the new marking is either equal to an earlier one in the tree or it covers another marking which is found on the path leading from the root to this new node.

In the first case, equality, we can mark the new node as a duplicate node. There is then no need to check the enabled transitions and the new markings resulting from the firing of these transitions because it has already been done for the first appearance of this node in the tree. An example see in Fig. 7.

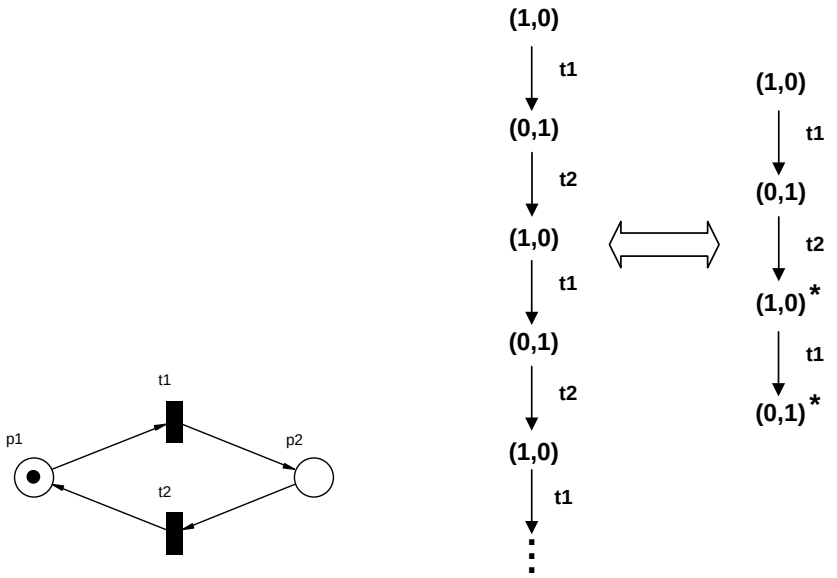


Figure 7: A Petri net and its infinite reachability tree and its reachability tree with duplicate nodes

The second case (see an example in Fig. 8), when the new marking covers an earlier one lying on the path from the root, refers to the cyclic behaviour of the net. This means that there is a loop of transitions that can be performed an arbitrarily number of times. It is unnecessary to indicate all nodes belonging to each appearance of this loop but somehow we have to refer to them.

The introduction of symbol ω can solve the loop indication problem. The symbol ω represents an arbitrarily large number of tokens. For any constant a the following

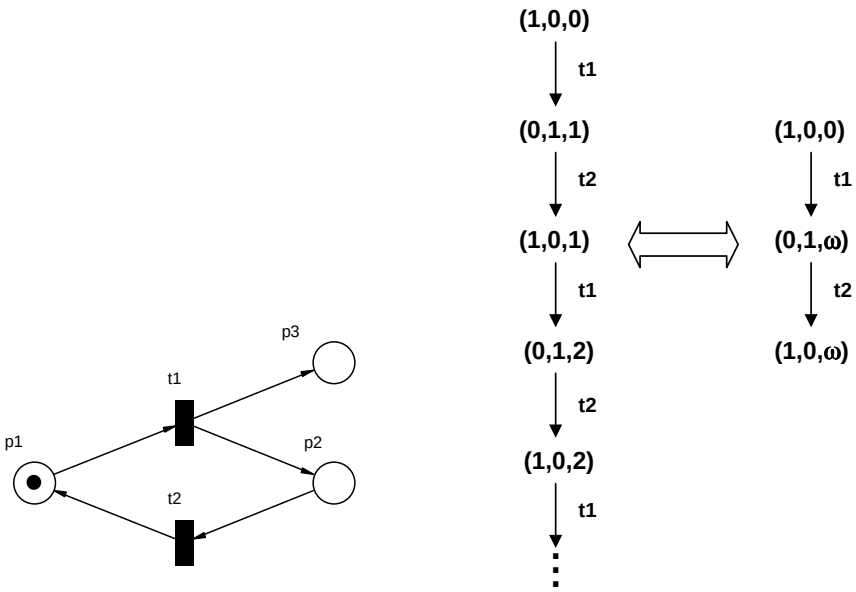


Figure 8: A Petri net and its infinite reachability tree and its reachability tree with using of symbol ω

is true:

$$\omega \pm a = \omega, \qquad a < \omega, \qquad \omega \leq \omega.$$

Having constructed the reachability tree of a Petri net, most of the analysis of its properties can be performed by searching in the tree as follows:

- A Petri net is *bounded* if and only if the symbol ω does not appear in any of the markings of the tree.
- A Petri net is *safe* if and only if only zero and one values (0's and 1's) appear in the markings of the tree.
- A transition is *dead* if and only if it does not appear in the tree (as edge weight).
- The reachability and coverability problems can be solved by searching for the predefined marking(s) in the tree.

The main disadvantage of the analysis with the reachability tree is its exhaustive characteristic. Despite the introduced modifications in the construction, the reachability tree can be very large and this can cause the time and space needed for

the construction and search to grow exponentially, therefore this analysis is usually computationally hard.

2.7.2.2 Analysis with matrix equations An analysis using the reachability graph gives information about the behaviour of the net starting from a given initial state. It would be a great advantage if it could somehow be generalized and we could find a method which solved the analysis problem in a shorter time and in a simpler way than the generation of trees. The invariance analysis can partly give an answer to this request. Using the matrix based description of the Petri net model we can analyze the structural properties of the system.

The representation of Petri net by matrices Let us represent the Petri net model of the investigated system by an incidence matrix. The first index of an element in the incidence matrices refers to the corresponding place, while the second index refers to the corresponding transition. The number of rows is equal to the number of places and the number of columns is equal to the number of transitions. An entry in the matrix is equal to the difference between the weights of the outgoing and and incoming arcs of a transition-place pair:

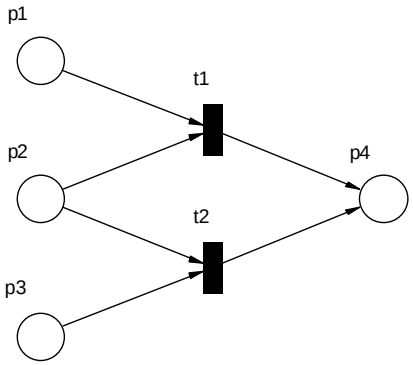
$$h_{ij} = h_{ij}^{+} - h_{ij}^{-}$$

where:

h_{ij} an entry of incidence matrix $H = [h_{ij}]$;
$h_{ij}^{+} = w(p_i, t_j)$ is the weight of the arc from place i to transition j;
$h_{ij}^{-} = w(t_j, p_i)$ is the weight of the arc from transition j to place i.

From the point of view of engineering meaning, incidence matrices can be interpreted as follows. An element of an incidence matrix gives the relation between a place and a transition. If an element h_{ij} is not equal to zero then transition t_j and place p_i are connected. If h_{ij} is a positive number then place p_i is a precondition of transition t_j while if h_{ij} is a negative number then this place is a consequence of it.

A zero entry can have different meaning. It can mean that there is no connection between the given transition and the given place but we get the same entry if the



$$H = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ -1 & -1 \end{bmatrix}$$

Figure 9: A Petri net and its incidence matrix

given place is both an input place and an output place of transition having the same weight. To avoid this information loss we assume that the investigated Petri net is pure, i.e. it does not contain any self-loops. If it contains one then we can eliminate it by adding a dummy transition-place pair to this self-loop. The column vector of an incidence matrix gives then all the preconditions and consequences of a given place and the transitions of the net.

2.8 Hierarchical Petri nets[1]

One of the main advantages of modelling with Petri nets is the capability to describe hierarchical systems. This means that the system to be modelled can be described on different levels. In the case of a complex system the models of subprocesses can be made first and checked separately then they can be built into the model of the whole system. Both transitions and places can be considered as composite elements; i.e. subnets can be built into them. This process can be repeated in arbitrary depth[6].

In the case of large systems containing a large number of similar subprocesses another advantage of this method is that the subnets of these elements have only to be made in one instance in advance, then they can be used as modular elements during the modelling process which simplifies the modelling task [6].

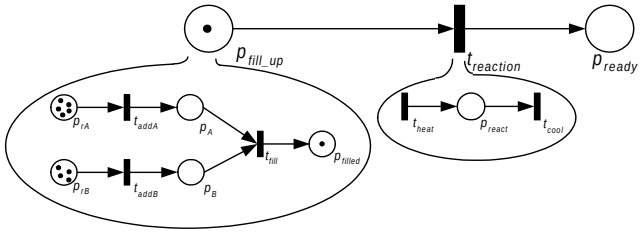


Figure 10: Decomposition of Petri net elements

Let the low level Petri net (denoted by H) be defined in the following way:

$$H = (P, T, I, O), \tag{4}$$

where:

P	finite nonempty set of places,
T	finite nonempty set of transitions,
I	$I : T \to P^\infty$ the input function (see Eq. (2)),
O	$O : T \to P^\infty$ the output function (see Eq. (3)).

The difference between definition (1) and definition (4) is the following: here we have a decomposed the set of edges by the input and output functions and the weight function W is also modified to have:

$$W : F \to 1 \ .$$

The Petri net of the plant-level modelled system, i.e. the net without any substituting *subnet*, will be called *supernet* in this report. Embedded Petri nets describing operating procedures on the unit-level are called subnets. The overall Petri net containing the plant-level procedures with all of its substituting subnets is termed *complete net* in the following.

2.8.1 Definition of subnets

2.8.1.1 Subnet substituting a transition Net $H_i = (P_i, T_i, I_i, O_i)$ is called the subnet *substituting transition* $t_i \in T$. The interpretation of the sets of the subnet and the relation of these sets to the set of the supernet is the following:

1. The set of places (P) in the supernet can be partitioned into subsets Q and $\overline{Q}$ such that

$$P = Q \cup \overline{Q} \ , \quad Q \cap \overline{Q} = \emptyset \ .$$

The element of the subsets Q and $\overline{Q}$ are the following:

$$\forall q \in Q : \#(q, O(t_i)) \neq 0 \ \ OR \ \ \#(q, I(t_i)) \neq 0 \ ,$$

i.e. the elements of Q are either inputs or outputs of transition t_i, and

$$\forall \overline{q} \in \overline{Q} : \#(\overline{q}, O(t_i)) = 0 \ \ AND \ \ \#(\overline{q}, I(t_i)) = 0 \ ,$$

i.e. the element of $\overline{Q}$ are not in connection with transition t_i.

2. Let the set P_i be the set of places of the subnet t_i:

$$Q \cap \overline{Q} \cap P_i = \emptyset \ .$$

The set of Q can be partitioned in the following way:

$$Q = Q_1 \cup Q_2$$

$$\forall q_1 \in Q_1 : \#(q_1, I(t_i)) \neq 0, \forall q_2 \in Q_2 : \#(q_2, I(t_i)) \neq 0$$

i.e. the elements in Q_1 are the input places to t_i and Q_2 consists of the output places.

3. Let T_i denote the set of transitions in the subnet:

$$T \cap T_i = \emptyset, \ |T_i| > 1 \ .$$

There is one and only one transition in the set T_i which inputs refer to the inputs in t_i:

$$\exists t_e \in T_i : \forall q_1 \in Q_1 : \#(q_1, I_i(t_e)) = \#(q_1, I(t_i))$$

and there is one and only one transition in the set T_i which outputs refer to the outputs in t_i:

$$\exists t_v \in T_i : \forall q_2 \in Q_2 : \#(q_2, O_i(t_v)) = \#(q_2, O(t_i)) \ .$$

4. The element of the set P_i are the inner places of the subset, so they must not be neither inputs of t_e nor outputs of t_v:

$$\forall r \in P_i : \#(r, I_i(t_e)) = 0, \#(r, O_i(t_v)) = 0 \ .$$

Every inner place has to be an input and an output of at least one transition:

$$\forall r \in P_i : \exists t_1, t_2 \in T_i : \#(r, I_i(t_1)) \neq 0, \#(r, O_i(t_2)) \neq 0 \ .$$

2.8.1.2 Subnet substituting a place Net $H_j = (P_j, T_j, I_j, O_j)$ is called the subnet *substituting place* $p_j \in P$. The interpretation of the sets of the subnet and the relation of these sets to the set of the supernet is the following:

1. The set of transitions (T) in the supernet can be partitioned into subsets S and $\overline{S}$ such that

$$T = S \cup \overline{S} \ , \quad S \cap \overline{S} = \emptyset \ .$$

The element of the subsets S and $\overline{S}$ are the following:

$$\forall s \in S : \#(p_j, O(s)) \neq 0 \ \ OR \ \ \#(p_j, I(s)) \neq 0 \ ,$$

i.e. the elements of S are either inputs or outputs of place p_j, and

$$\forall \overline{s} \in \overline{S} : \#(p_j, O(\overline{s})) = 0 \ \ AND \ \ \#(p_j, I(\overline{s})) = 0 \ ,$$

i.e. the element of $\overline{S}$ are not in connection with place p_j.

2. Let the set T_j be the set of transitions of the subnet p_j:

$$S \cap \overline{S} \cap T_j = \emptyset \ .$$

The set of S can be partitioned in the following way:

$$S = S_1 \cup S_2$$

$$\forall s_1 \in S_1 : \#(p_j, I(s_1)) \neq 0, \forall s_2 \in S_2 : \#(p_j, I(s_2)) \neq 0$$

i.e. the elements in S_1 are the input transitions to p_j and S_2 consists of the output transitions.

3. Let P_j denote the set of places in the subnet:

$$P \cap P_j = \emptyset,\ |P_j| > 1\ .$$

There is one and only one place in the set P_j which inputs refer to the inputs in p_j:

$$\exists p_e \in P_j : \forall s_1 \in S_1 : \#(p_e, I_j(s_1)) = \#(p_j, I(s_1))$$

and there is one and only one place in the set P_j which outputs refer to the outputs in p_j:

$$\exists p_v \in P_j : \forall s_2 \in S_2 : \#(p_v, O_j(s_2)) = \#(p_j, O(s_2))\ .$$

4. The element of the set T_j are the inner transitions of the subset, so they must not be neither inputs of p_e nor outputs of p_v:

$$\forall t \in T_j : \#(p_e, I_j(t)) = 0, \#(p_v, O_j(t)) = 0\ .$$

Every inner transition has to be an input and an output of at least one place:

$$\forall t \in T_j : \exists p_1, p_2 \in P_j : \#(p_1, I_j(t)) \neq 0, \#(p_2, O_j(t)) \neq 0\ .$$

2.8.2 Definition of the complete net

Let $H = (P, T, I, O)$ be a low level Petri net, $H_i = (P_i, T_i, I_i, O_i)$ be a subnet substituting $t_i \in T$ and $H_j = (P_j, T_j, I_j, O_j)$ be a subnet substituting $p_j \in P$ as defined above. Then net $H_t = (P_t, T_t, I_t, O_t)$ is defined to be the complete net built up from H, H_i and H_j, where

$$P_t = (P \setminus \{p_j\}) \cup P_i \cup P_j\ ,$$

$$T_t = (T \setminus \{t_i\}) \cup T_i \cup T_j\ ,$$

$$I_t = (I \cup I_i \cup I_j)\ \setminus\ \{(t_i, K) | K \in P_t^\infty\}\ \setminus\ \{(L, p_j) | L \in P_t\}$$
$$\cup\ \{(t_e, q_1) | q_1 \in Q_1\}\ \cup\ \{(s_1, p_e) | s_1 \in S_1\}$$

and

$$O_t = (O \cup O_i \cup O_j)\ \setminus\ \{(t_i, K) | K \in P_t^\infty\}\ \setminus\ \{(L, p_j) | L \in P_t\}$$
$$\cup\ \{(t_v, q_2) | q_2 \in Q_2\}\ \cup\ \{(s_2, p_v) | s_2 \in S_2\}\ .$$

2.8.3 Example for a hierarchical Petri net

Let we see an example to understand the previous definitions. A supernet H is seen in Fig. 11.

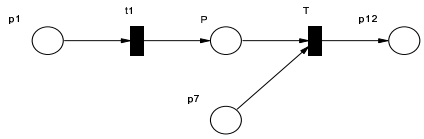


Figure 11: The supernet

By the previous definitions we give subnet $H = (P, T, I, O)$, where:

$$\begin{aligned} P &= \{p_1, P, p_7, p_{12}\} \\ T &= \{t_1, T\} \\ I &= \{(p_1, t_1), (P, T), (p_7, T)\} \\ O &= \{(P, t_1), (p_{12}, T)\} \end{aligned}$$

There are two subnet, one is substituting place P and other one is substituting transition T. So the complete net is seen in Fig. 12.

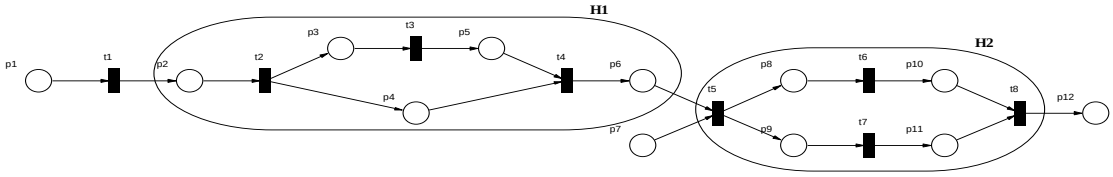


Figure 12: The complete net

The subnet H_1 substituting place P is writing as $H_1 = (P_1, T_1, I_1, O_1)$, where:

$$\begin{aligned} P_1 &= \{p_2, p_3, p_3, p_4, p_6\} \\ T_1 &= \{t_2, t_3, t_4\} \\ I_1 &= \{(p_2, t_2), (p_3, t_3), (p_4, t_4), (p_5, t_4)\} \\ O_1 &= \{(p_3, t_2), (p_4, t_2), (p_5, t_3), (p_6, t_4)\} \end{aligned}$$

The subnet H_2 substituting transition T is writing as $H_2 = (P_2, T_2, I_2, O_2)$, where:

$$\begin{aligned} P_2 &= \{p_8, p_9, p_{10}, p_{11}\} \\ T_2 &= \{t_5, t_6, t_7, t_8\} \\ I_2 &= \{(p_8, t_6), (p_8, t_7), (p_{10}, t_8), (p_{11}, t_8)\} \\ O_2 &= \{(p_8, t_5), (p_9, t_5), (p_{10}, t_6), (p_{11}, t_7)\} \end{aligned}$$

Therefor the complete net $H = (P, T, I, O)$ computes as we wrote in Section 2.8.2, where:

$$\begin{aligned} p_e &= p_2 & p_v &= p_6 \\ t_e &= t_5 & t_v &= t_8 \end{aligned}$$

3 Modelling safe Petri nets with Boolean algebras[3]

3.1 Notations

Let $N = (P, T, F, M_0)$ be a safe Petri net. The set of reachable markings from M is denoted by $[M\rangle$. If a transition t is enabled at a marking M, we denoted by $M[t\rangle$. A marking in $[M_0\rangle$ can be represented by a set of places $M = \{p_1, ..., p_M\}$, $p_i \in P$, where $p_i \in M$ denotes the fact that there is a token in p_i. Therefore, any set of markings in $[M_0\rangle$ can be represented by a set $\mathcal{M}$ of subsets of P. Let M_P be the set of all markings of a safe Petri net with $|P|$ places ($|M_P| = 2^{|P|}$). The system

$$(2^{M_P}, \cup, \cap, \emptyset, M_P)$$

is the Boolean algebra of sets of markings. This system is isomorphic with the Boolean algebra of n-variable logic functions, where $n = |P|$.

We will indistinctively use p_i to denote a place in P, or a variable in the Boolean algebra of n-variable logic functions. Therefore, there is a one-to-one correspondence between markings of M_P and vertices of B^n. A marking $M \in M_P$ is represented by means of an *encoding function* that provides a binary mapping from M_P into B^n, that is, $\mathcal{E} : M_P \rightarrow B^n$, where the image of markings $M \in M_P$ is encoded into an element $(p_1, ..., p_n) \in B^n$, such that:

$$p_i = \begin{cases} 1 & if\ p_i \in M, \\ 0 & if\ p_i \notin M. \end{cases}$$

3.1.1 Simple example

Let us look at a simple Petri net seen in Fig. 13. We use this Petri net throughout this section.

As an example, in Fig. 13 both the vertex $(1, 0, 0, 0, 0, 0, 0) \in B^7$ and the cube $p_1\overline{p_2}p_3p_4p_5p_6p_7$ represent the markings in which p_1 is marked and p_2, p_3, p_4, p_5, p_6, p_7 are not marked.

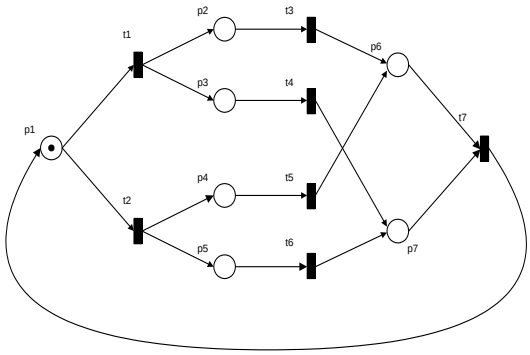


Figure 13: Simple Petri net

3.2 Characteristic function

Let $V \subseteq B^n$ be the set of vertices in the Boolean algebra of n-variable logic functions. The *characteristic function* χ_V of the set V is defined as the logic function that evaluates to 1 for those vertices of B^n that are in V, i.e.

$$v \in B^n \ : \ v \in V \Leftrightarrow \chi_V(v) = 1 \ .$$

Extending the previously introduced *encoding function* $\mathcal{E}$, each set of markings $\mathcal{M} \in 2^{M_P}$ has a corresponding *image* $V \in B^n$ according to $\mathcal{E}$ defined by:

$$V = \{v \in B^n \ : \ \exists M \in \mathcal{M}, v = \mathcal{E}(M)\} \ .$$

Then the *characteristic function* of the set of $\mathcal{M}$ is a function $\chi_{\mathcal{M}} : B^n \rightarrow B$, that evaluates to 1 for those vertices that correspond to markings belonging to $\mathcal{M}$; that is, $\chi_{\mathcal{M}} = \chi_V$. From now on, and for sake of simplicity, we will use $\mathcal{M}$ and $\chi_{\mathcal{M}}$ to denote the characteristic function of the set of markings $\mathcal{M}$.

Characteristic functions can also be used to represent *binary relations* between sets of markings; that is, subsets of a Cartesian product between both sets. Given two sets $\mathcal{M}$ and $\mathcal{M}'$, to represent the binary relation $R \subseteq \mathcal{M} \times \mathcal{M}'$ it is necessary to use different sets of variables to identify the elements of each set. Taking sets of Boolean variables $\{p_1, ..., p_n\}$ for $\mathcal{M}$ and $\{q_1, ..., q_n\}$ for $\mathcal{M}'$, the characteristic function is defined by:

$$\chi_R(p_1, ..., p_n, q_1, ..., q_n) = 1 \Leftrightarrow$$

$$\exists(M, M') \in R : [\mathcal{E}(M) = (p_1, ..., p_n) \wedge \mathcal{E}(M') = (q_1, ..., q_n)] \ .$$

Given the binary relation R between sets $\mathcal{M}$ and $\mathcal{M}'$, the elements of $\mathcal{M}$ that are in relation with some element of $\mathcal{M}'$, is defined by the set $R(\mathcal{M})$; such that:

$$R(\mathcal{M}) = \{M \in \mathcal{M} : \exists M' \in \mathcal{M}', (M, M') \in R\} \ ,$$

and its characteristic function of $\chi_R(\mathcal{M})$ is computed as:

$$\chi_R(\mathcal{M})(p_1, ..., p_n) = \exists_{q_1,...,q_n} \chi_R(p_1, ..., p_n, q_1, ..., q_n) \ .$$

3.3 Transition firing

The structure of Petri net defines a set of firing rules that determine the behavior of the net, called *transition firing rules*. The *transition function* for a transition t of the net, is a function:

$$\delta : M_P \times T \rightarrow M_P \ , \tag{5}$$

that transforms a marking $M \in M_p$ into new marking(s) $M' \in M_P$ by firing the transition $t \in T$ $(M' = \delta(M, t))$. The corresponding markings in M' are generated assuming that t is enabled in M; otherwise, the empty marking $\delta(M, t) = \emptyset$ is generated.

This concept is equivalent to the one-step reachability in Petri nets; M' is reachable from M in one step if there is a transition $t \in T$ such that $M' = \delta(M, t)$. According to this objective, the transition function $\delta = (\delta_1, ..., \delta_{|P|})$ for a transition $t \in T$ defines how the contents of each place is transformed as a result of firing a transition, and it is defined:

$$\forall i \in 1, ..., |P| \quad \delta_i(p_1, ...p_n, t) = E_t \cdot \begin{cases} 1 & if\ p_i \in t^\bullet \\ 0 & if\ p_i \in^\bullet t\ and\ p_i \notin t^\bullet \\ p_i & otherwise \end{cases}$$

By firing transition t, the function returns 1 if p is in its post-set, 0 if p is in its pre-set (but not a self-loop), and otherwise remains with the same value. Additionally, E_t is the characteristic function of the set of markings in which transition t is enabled, defined as:

$$E_t = \prod_{p_i \in^\bullet t} p_i \ . \tag{6}$$

Extending the concept to k-steps reachability, a marking M_k is *reachable* in k steps from the initial marking M_0 if there is a sequence of markings $M_1, M_2, ..., M_{k-1}$ and a sequence of transitions $t_1, t_2, ..., t_k$ such that, $\delta(M_0, t_1) = M_1$,...,$\delta(M_{k-1}, t_k) = M_k$.

In order to manipulate the firing of transitions in sets of markings rather than using a marking-per-marking basis, the *transition function* of a transition can be redefined as a function

$$\delta : 2^{M_P} \times T \rightarrow 2^{M_P} \ , \tag{7}$$

that for a given transition $t \in T$ transforms a set of markings $\mathcal{M}$ into a new set of markings $\mathcal{M}'$, i.e.

$$\mathcal{M}' = \delta(\mathcal{M}, t) = \{M' \in M_P \ : \ \exists M \in \mathcal{M}, M[t\rangle M'\} \ .$$

Transition functions for net transitions can be further generalized to be the transition function of the whole Petri net:

$$\Delta : 2^{M_P} \rightarrow 2^{M_P} \ , \tag{8}$$

where all transitions are simultaneously fired in the same function. Δ transforms a set of markings $\mathcal{M}$ into the set of markings $\mathcal{M}'$ that can be reached from $\mathcal{M}$ in one step (one transition firing). Equation (8) can be obtained by computing:

$$\Delta(\mathcal{M}) = \bigcup_{\forall t_i \in T} \delta(\mathcal{M}, t_i) \ .$$

Note that (8) calculates the image of several markings simultaneously. Δ performs the *constrained image computation* of the net.

There are two different techniques to implement the *constrained image computation* for transitions by

- *topological image computation*,
- *transition relation image computation*,

In the remainder of this section we will study the topological image computation and the transition relation image computation. We refer the reader to [8] for the other techniques.

3.3.1 Simple example (Continued)

Following the example in Fig. 13 that contains the Boolean variables $p_1, ...p_7$, the transition function for the transitions in the Petri net are the functions:

$$\begin{array}{rcl|ccccccc|} & & & \delta_1, & \delta_2, & \delta_3, & \delta_4, & \delta_5, & \delta_6, & \delta_7, & \\ \delta(M,t_1) & = & (& 0, & p_1, & p_1, & p_1p_4, & p_1p_5, & p_1p_6, & p_1p_7, &), \\ \delta(M,t_2) & = & (& 0, & p_1p_2, & p_1p_3, & p_1, & p_1, & p_1p_6, & p_1p_7, &), \\ \delta(M,t_3) & = & (& p_1p_2, & 0, & p_2p_3, & p_2p_4, & p_2p_5, & p_2, & p_2p_7, &), \\ \delta(M,t_4) & = & (& p_1p_3, & p_2p_3, & 0, & p_3p_4, & p_3p_5, & p_3p_6, & p_3, &), \\ \delta(M,t_5) & = & (& p_1p_4, & p_2p_4, & p_3p_4, & 0, & p_4p_5, & p_4, & p_4p_7, &), \\ \delta(M,t_6) & = & (& p_1p_5, & p_2p_5, & p_3p_5, & p_4p_5, & 0, & p_5p_6, & p_5, &), \\ \delta(M,t_7) & = & (& p_6p_7, & p_2p_6p_7, & p_3p_6p_7, & p_4p_6p_7, & p_5p_6p_7, & 0, & 0, &). \end{array}$$

Therefore, firing transition t_1 from markings $p_1\overline{p_2}p_3\overline{p_4}p_5\overline{p_6}p_7$ and $\overline{p_1}p_2p_3\overline{p_4}p_5\overline{p_6}p_7$ results into:

$$\overline{p_1}p_2p_3\overline{p_4}p_5\overline{p_6}p_7 = \delta(p_1\overline{p_2}p_3\overline{p_4}p_5\overline{p_6}p_7, t_1) \ ,$$

$$\overline{p_1}p_2p_3\overline{p_4}p_5\overline{p_6}p_7 = \delta(\overline{p_1}p_2p_3\overline{p_4}p_5\overline{p_6}p_7, t_1) \ .$$

The transition function for the Petri net is the function:

$$\begin{array}{rll} \Delta(M) = (& p_1(p_2+p_3+p_4+p_5)+p_6p_7, & \\ & p_2(p_1+p_3+p_4+p_5+p_6p_7)+p_1, & \\ & p_3(p_1+p_2+p_4+p_5+p_6p_7)+p_1, & \\ & p_4(p_1+p_2+p_3+p_5+p_6p_7)+p_1, & \\ & p_5(p_1+p_2+p_3+p_4+p_6p_7)+p_1, & \\ & p_6(p_1+p_3+p_5)+p_2+p_4, & \\ & p_7(p_1+p_2+p_4)+p_3+p_5 &). \end{array}$$

3.3.2 Topological image computation

Constrained image computation for transitions can efficiently be implemented by using the topological information of the Petri net and the characteristic function of sets of markings. First of all, we will present the characteristic function of some

important sets related to a transition $t \in T$:

$$\begin{aligned} E_t &= \prod_{p_i \in {}^\bullet t} p_i \qquad (t \; enabled), \\ NPM_t &= \prod_{p_i \in {}^\bullet t} \overline{p_i} \qquad (no \; predecessor \; of \; t \; is \; marked), \\ ASM_t &= \prod_{p_i \in t^\bullet} p_i \qquad (all \; successors \; of \; t \; are \; marked), \\ NSM_t &= \prod_{p_i \in t^\bullet} \overline{p_i} \qquad (no \; successor \; of \; t \; is \; marked). \end{aligned}$$

Given these characteristic functions, the *constrained image computation* for transitions is reduced to calculate:

$$\delta(M,t) = (M_{E_t} \cdot NPM_t)_{NSM_t} \cdot ASM_t \; . \tag{9}$$

3.3.2.1 Simple example (Continued) We will show by an example how this formula "simulates" firing a transition t. In the example of Fig. 13, given the set of markings $\mathcal{M} = \{\{p_2 \; p_3\} \{p_2 \; p_7\} \{p_4 \; p_7\}\}$, with its characteristic function:

$$\mathcal{M} = \overline{p_1}p_2p_3\overline{p_4}p_5\overline{p_6}\overline{p_7} + \overline{p_1}p_2\overline{p_3}p_4p_5\overline{p_6}p_7 + \overline{p_1p_2p_3}p_4\overline{p_5p_6}p_7 \; ,$$

we will calculate $\mathcal{M}' = \delta(\mathcal{M}, t_3)$. First, $\mathcal{M}_{E_{t_3}}$ (the cofactor of $\mathcal{M}$ with respect to $E_{t_3} = p_2$) selects those markings in which t_3 is enabled and removes its predecessor places from the characteristic function:

$$\mathcal{M}_{E_{t_3}} = \overline{p_1}p_3\overline{p_4}p_5\overline{p_6}\overline{p_7} + \overline{p_1p_3}p_4p_5\overline{p_6}p_7 \; ,$$

Then the product with $NPM_{t_3} = \overline{p_2}$ simulates the elimination of the tokens in the predecessor places:

$$\mathcal{M}_{E_{t_3}} \cdot NPM_{t_3} = \overline{p_1p_2}p_3\overline{p_4}p_5\overline{p_6}\overline{p_7} + \overline{p_1p_2p_3}p_4p_5\overline{p_6}p_7 \; .$$

Next, taking the cofactor with respect to $NSM_{t_3} = \overline{p_6}$ removes all successor places from the characteristic function:

$$\left(\mathcal{M}_{E_{t_3}} \cdot NPM_{t_3}\right)_{NSM_{t_3}} = \overline{p_1p_2}p_3\overline{p_4}p_5\overline{p_7} + \overline{p_1p_2p_3}p_4p_5\overline{p_7} \; .$$

Finally, the product with $ASM_{t_3} = p_6$ adds a token in all successor places of t_3:

$$\left(\mathcal{M}_{E_{t_3}} \cdot NPM_{t_3}\right)_{NSM_{t_3}} \cdot ASM_{t_3} = \overline{p_1p_2}p_3\overline{p_4}p_5p_6\overline{p_7} + \overline{p_1p_2p_3}p_4p_5p_6p_7 \; ,$$

generating the characteristic function of the set of markings $\mathcal{M}' = \{\{p_3 \; p_6\} \{p_6 \; p_7\}\}$.

3.3.3 Transitional relation image computation

The *transition function* relates sets of markings $\mathcal{M}' = \delta(\mathcal{M}, t)$ such that the markings in $\mathcal{M}'$ are reachable from $\mathcal{M}$ after firing transition t. The relation defined by δ can also be represented by a *characteristic function* in which there are two different sets of variables, $\{p_1, ..., p_n\}$ for $\mathcal{M}$ and $\{q_1, ..., q_n\}$ for $\mathcal{M}'$ respectively ($n = |\mathcal{P}|$). According to the definition of function δ, its characteristic function is described by the binary relation[1]:

$$\mathcal{R}_t(q_1, ..., q_n, p_1, ..., p_n) = \prod_{i=1}^{|P|} (q_i \equiv \delta_i(p_1, ..., p_n, t)) \ .$$

Finding the set of markings $\mathcal{M}'$ that can be reached after firing transition t from any marking in the set $\mathcal{M}$ (the *constrained image computation* for transitions) is reduced to compute:

$$\mathcal{M}' = \exists_{p_1,...,p_n} \mathcal{R}_t(q_1, ..., q_n, p_1, ..., p_n) \cdot \mathcal{M} \ . \tag{10}$$

The transition relation of the whole net can be computed following an *interleaving* model, in which only one transition is fired at the same time, i.e.

$$\mathcal{R}_t(q_1, ..., q_n, p_1, ..., p_n) = \bigcup_{t \in T} \left[\prod_{i=1}^{|P|} (q_i \equiv \delta_i \, (p_1, ..., p_n, t)) \right] \ .$$

The main computation problem in image computation with the transition relation appears when taking the conjunction $\prod_{i=1}^{|P|}$.

3.3.3.1 Simple example (Continued) As an example, we compute the characteristic function for the transition relation of t_1 in Fig. 13. Note that in this example the following set of Boolean variables are required: $\{p_1, ..., p_7\}$ and $\{q_1, ..., q_7\}$. Then

$$\begin{aligned}
\mathcal{R}_t(q_1, ..., q_7, p_1, ..., p_7) &= \overline{q_1} \cdot (q_2 \equiv p_1) \cdot (q_3 \equiv p_1) \cdot (q_4 \equiv p_1 p_4) \cdot \\
&\quad (q_5 \equiv p_1 p_5) \cdot (q_6 \equiv p_1 p_6) \cdot (q_7 \equiv p_1 p_7) \\
&= \overline{q_1} \cdot (q_2 p_1 + \overline{q_2 p_1}) \cdot (q_3 p_1 + \overline{q_3 p_1}) \cdot \\
&\quad (q_4 p_1 p_4 + \overline{q_4}(\overline{p_1} + \overline{p_4})) \cdot (q_5 p_1 p_5 + \overline{q_5}(\overline{p_1} + \overline{p_5})) \cdot \\
&\quad (q_6 p_1 p_6 + \overline{q_6}(\overline{p_1} + \overline{p_6})) \cdot (q_7 p_1 p_7 + \overline{q_7}(\overline{p_1} + \overline{p_7})) \ .
\end{aligned}$$

[1]Note that the operation $a \equiv b$ stands for a equivalent to b and it is defined as $ab + \overline{a}\overline{b}$

3.4 Reachability set computation

The algorithm presented in Fig. 14 below traverses the Petri net and calculates the reachability set from the selected initial marking. The union and difference of sets of markings are performed by manipulating their corresponding characteristic functions.

```
traverse_Petri_net (P,T,W,M0)
{
  /* Let δ be the transition function of the transitons in T */
  Reached := From := {M0};
  repeat
     To := ∅;
     foreach t∈T do To := To ∪ δ(From,t);
     New := To - Reached;
     From := New;
     Reached := Reached ∪ New;
  until (New = ∅);
  return Reached; /* The set of all reached marking from M0 */
}
```

Figure 14: Algorithm for symbolic Petri net traversal using the δ function

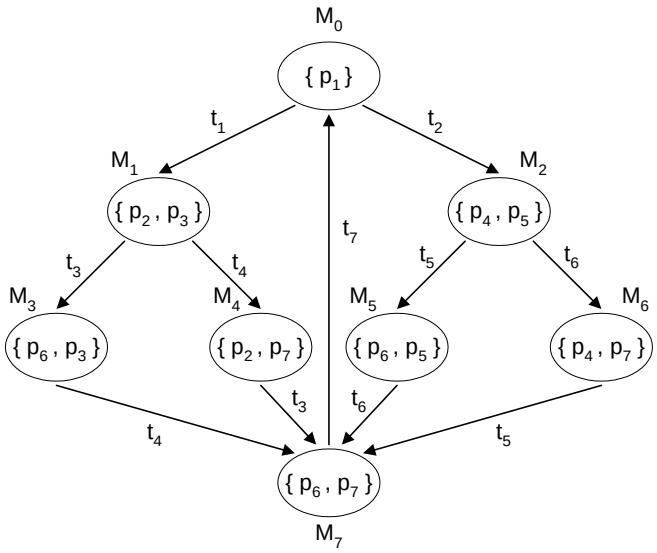


Figure 15: Symbolic reachability tree of the example Petri net (Fig. 13)

3.4.0.2 Simple example (Continued) As an example, take the initial marking $\{p_1\}$ of the example depicted in Fig. 13. After the first iteration of the *repeat*

loop, the algorithm yields (by firing transitions t_1 and t_2) to $To = \{p_2p_3 + p_4p_5\}$, $New = \{p_2p_3 + p_4p_5\} - \{p_1\} = \{p_2p_3 + p_4p_5\}$, $Reached = \{p_1 + p_2p_3 + p_4p_5\}$.

The final set of reachable markings is shown in Fig. 15, where the nodes represent markings and the edges the firing transitions.

3.5 Verification of Petri net properties

In this section we show how safe Petri net properties can be verified by Boolean manipulation on the set of reachable markings. From the wide range of properties that can be verified with this approach we have chosen two of them as examples: safeness and liveness.

3.5.1 Safeness verification

Using Eq. (9) (the cofactor with NSM_t), markings with more than one token in any place are removed from the set of reachable markings. However, in the case when the transition relation is used to implement the image computation procedure, non-existing markings may be introduced.

Detecting unsafeness can be done by identifying a marking M in which a transition t is enabled, and some successor place $p \in t^\bullet$ not included in a self-loop ($p \notin^\bullet t$), is already marked. In that situation, after firing transition t, place p will contain two tokens. Formally, a Petri net is *not safe* if:

$$\exists (M \in [M_0\rangle, t \in T, p \in P) \ : \ [M[t\rangle \wedge p \in t^\bullet \wedge p \notin^\bullet t \wedge M(p) = 1] \ . \qquad (11)$$

Given the set of reachable markings $[M_0\rangle$, the algorithm decipted in Fig. 16 detects whether a Petri net is safe or not, by checking one equation for each transition. On the other side, in case the transition relation is used for the reachability computation, safeness must be guaranteed every time a transition t is going to be fired from a set of markings. Given the set of markings *From* in the algorithm of Fig. 14, the safeness of the Petri net can be assured at each iteration of the algorithm by checking that the following formula holds:

$$\forall t \in T \ : \ \left[From \cdot E_t \cdot \sum_{(p \in t^\bullet) \wedge (p \notin^\bullet t)} p = \emptyset \right] \ . \qquad (12)$$

```
is_safe (P,T,W,M_0,[M_0⟩)
{
  foreach t∈T do
     Succ_p := ∅;
     Enabled := [M_0⟩ * E_t;
     foreach (p_i ∈ t• ∧ p_i ∉ •t) do
        Succ_p := Succ_p + p_i;
     if (Enabled * Succ_p ≠∅) then return false;
  return true;
}
```

Figure 16: Algorithm for safeness checking of ordinary Petri nets

3.5.2 Liveness verification

A Petri net is said to have a *deadlock* if there is a marking where no transition can be fired. A transition is said to be *dead* if it can never be fired in any firing sequence from the initial marking M_0. A transition that can be fired at least once in some firing sequence from M_0 is said to be *potentially fireable*.

The set of markings where a *deadlock* occurs is calculated as follows:

$$Deadlock \equiv \left[[M_0\rangle \cdot \prod_{t \in T} \overline{E_t} \neq \emptyset \right] \ . \tag{13}$$

The set of markings where a transition t is potentially fireable is calculated as:

$$Fireable(t) = [M_0\rangle \cdot E_t \ . \tag{14}$$

Then, if $Fireable(t) = \emptyset$, then transition t is dead, otherwise it is live.

4 Extension to weighted and bounded Petri nets[3, 4]

This section presents the modifications needed to extend the Boolean manipulation techniques to k-bounded Petri nets.

4.1 Place encoding

A k-bounded place $p \in P$ can be represented with a set of Boolean variables, $v^1, ..., v^q$ to encode the up-to-k possible number of tokens. The number of required variables depends on the type of encoding. Two different encoding strategies will be proposed: *one-hot encoding* and *binary encoding*. If a *one-hot encoding* is used, k variables are needed. For example, in a 3-bounded Petri net the number of tokens in a place p could be represented by three variables. With a *binary encoding* $\lceil log_2(k+1) \rceil$ variables would be required (see Table 1), where $\lceil x \rceil$ (round x) is the nearest integer of x towards infinity.

#tokens	one-hot encoding	binary encoding
0	$\overline{v^3 v^2 v^1}$	$\overline{v^2 v^1}$
1	$\overline{v^3 v^2} v^1$	$\overline{v^2} v^1$
2	$\overline{v^3} v^2 \overline{v^1}$	$v^2 \overline{v^1}$
3	$v^3 \overline{v^2 v^1}$	$v^2 v^1$

Table 1: Encoding of k-bounded places (k=3)

The one-hot encoding can be implemented using a *transition function* simpler than the binary encoding, however, the number of variables, which is a critical parameter during the computation mechanisms, is larger than for the one-hot encoding.

4.2 Simple example

Let us look at a simple Petri net can seen in Fig. 17. We use this Petri net throughout this section.

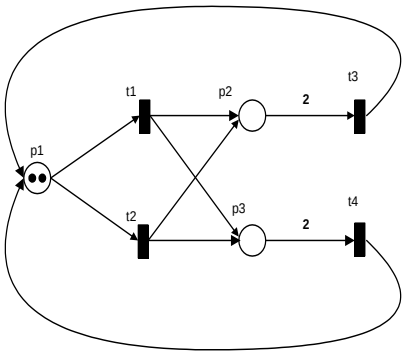


Figure 17: Bounded Petri net with an upper bound of two tokens

4.3 Transition firing

This section introduces the *transition functions* and *transition relations* required to implement weighted Petri nets using both *binary* and *one-hot* encodings. According to this objective, the transition function described in Eq. (5) for a transition t of the net, should be redefined into:

$$\delta_i(p_1, ..., p_n, t) = E_t \cdot \begin{cases} M(p_i) - W(p_i, t) & if\ p_i \in^\bullet t \setminus t^\bullet, \\ M(p_i) + W(t, p_i) & if\ p_i \in t^\bullet \setminus^\bullet t, \\ M(p_i) - W(p_i, t) + W(t, p_i) & if\ p_i \in t^\bullet \cap^\bullet t, \\ M(p_i) & otherwise. \end{cases}$$

Where E_t is the characteristic function of the set of k-bounded markings in which transition t is enabled. For a weighted Petri net:

$$E_t = \prod_{p \in^\bullet t} M(p) \geqq W(p, t)\ . \tag{15}$$

4.3.1 Binary encoding

Assuming that the number of tokens in place p is represented by a number of Boolean variables $p^1, ..., p^{K_p}$, and that the weight $W(p, t)$ is represented by the same number of binary encoded Boolean constants $w^1, ..., w^{K_p}$. Then the relation $M(p) \geqq W(p, t)$

can be described by the equation[2]:

$$
\begin{aligned}
M(p) \geqq W(p,t) \;=\; & (p^{K_p} > w^{K_p}) + \\
& (p^{K_p} \equiv w^{K_p}) \wedge (p^{K_p-1} > w^{K_p-1}) + \\
& (p^{K_p} \equiv w^{K_p}) \wedge (p^{K_p-1} \equiv w^{K_p-1}) \wedge (p^{K_p-2} > w^{K_p-2}) + \\
& ..... \\
& (p^{K_p} \equiv w^{K_p}) \wedge (p^{K_p-1} \equiv w^{K_p-1}) \wedge ... \wedge (p^2 \equiv w^2) \wedge (p^1 > w^1) + \\
& (p^{K_p} \equiv w^{K_p}) \wedge (p^{K_p-1} \equiv w^{K_p-1}) \wedge ... \wedge (p^2 \equiv w^2) \wedge (p^1 \equiv w^1) \\
= \; & (p^{K_p} \cdot \overline{w^{K_p}}) + \\
& (\overline{p^{K_p} \oplus w^{K_p}}) \cdot (p^{K_p-1} \cdot \overline{w^{K_p-1}}) + \\
& (\overline{p^{K_p} \oplus w^{K_p}}) \cdot (\overline{p^{K_p-1} \oplus w^{K_p-1}}) \cdot (p^{K_p-2} \cdot \overline{w^{K_p-2}}) + \\
& ..... \\
& (\overline{p^{K_p} \oplus w^{K_p}}) \cdot (\overline{p^{K_p-1} \oplus w^{K_p-1}}) \cdot ... \cdot (\overline{p^2 \oplus w^2}) \cdot (p^1 \cdot \overline{w^1}) + \\
& (\overline{p^{K_p} \oplus w^{K_p}}) \cdot (\overline{p^{K_p-1} \oplus w^{K_p-1}}) \cdot ... \cdot (\overline{p^2 \oplus w^2}) \cdot (\overline{p^1 \oplus w^1}) \;,
\end{aligned}
$$

where $\oplus$ denotes the exclusive "or" operation.

Hence, assuming that K_p is the number of Boolean variables to represent a place p, the markings in which a transition t is enabled to fire are defined by the characteristic function:

$$E_t = \prod_{p \in {}^\bullet t} \left[\sum_{i=1}^{K_p} \left[(p^i \cdot \overline{w^i}) \cdot \prod_{j=i+1}^{K_p} (\overline{p^j \oplus w^j}) \right] + \prod_{i=1}^{K_p} (\overline{p^i \oplus w^i}) \right] \;. \tag{16}$$

Once we have decided that transition t is enabled to fire in a given marking, we have to effectively implement the transition firing by:

1. eliminating the corresponding tokens from any place in the pre-set of transition t, and

2. adding the corresponding tokens to any place in the post-set of transition t.

Assume that the number of tokens in place p is represented by a number of (binary encoded) Boolean variables $p^1, ..., p^{K_p}$, and that the number of tokens that must

[2]Recall that $(a \geqq b)$ stands for $(a + \overline{b})$, $(a \equiv b)$ for $(\overline{a \oplus b})$, and $(a > b)$ for $(a \cdot \overline{b})$

be subtracted is represented by the same number of Boolean constants $w^1, ..., w^{K_p}$ (either subtracting $W(p,t)$ or $w(t,p) - W(p,t)$ if $W(p,t) > W(t,p)$). In those cases, the relation $\delta(M,t)$ is equivalent to the subtraction of two natural numbers, that can be described by the set of equations:

$$
\begin{array}{lll}
\left(\delta^1(M,t), ..., \delta^{K_p}(M,t)\right) & = & \\
E_t \cdot \big(\; p^1 \oplus w^1, & and & B_1 = \overline{p^1} \cdot w^1 \\
\quad p^2 \oplus w^2 \oplus B_1, & and & B_2 = \overline{p^2} \cdot w^2 + B_1 \cdot (\overline{p^2 \oplus w^2}) \\
\quad ... & & ... \\
\quad p^{K_p} \oplus w^{K_p} \oplus B_{K_p - 1} \; \big) & and & B_{K_p} = \overline{p^{K_p}} \cdot w^{K_p} + B_{K_p - 1} \cdot (\overline{p^{K_p} \oplus w^{K_p}}) \; .
\end{array}
$$

On the other hand, the number of tokens that must be added is represented by the same number of boolean constants $w^1, ..., w^{K_p}$ (either adding $W(t,p)$ or $w(t,p) - W(p,t)$ if $W(t,p) > W(p,t)$). In those cases, the relation $\delta(M,t)$ is equivalent to the addition of two natural numbers, that can be described by the set of equations:

$$
\begin{array}{lll}
\left(\delta^1(M,t), ..., \delta^{K_p}(M,t)\right) & = & \\
E_t \cdot \big(\; p^1 \oplus w^1, & and & C_1 = p^1 \cdot w^1 \\
\quad p^2 \oplus w^2 \oplus C_1, & and & C_2 = p^2 \cdot w^2 + B_1 \cdot (p^2 \oplus w^2) \\
\quad ... & & ... \\
\quad p^{K_p} \oplus w^{K_p} \oplus C_{K_p - 1} \; \big) & and & B_{K_p} = p^{K_p} \cdot w^{K_p} + C_{K_p - 1} \cdot (p^{K_p} \oplus w^{K_p}) \; .
\end{array}
$$

Therefore, in order to implement an image computation based on transition relations for weighted Petri nets, we only need to redefine the characteristic function of the function δ (see Eq.(10)) into:

$$
\mathcal{R}_t(q_1, ..., q_n, p_1, ..., p_n) = \prod_{i=1}^{|P|} \prod_{j=1}^{K_P i} \left(q_i^j \equiv \delta_i^j \left(p_1, ..., p_n, t\right)\right) \; .
$$

The function δ_i^j for the j-variable encoding the tokens in a place p_i is described by:

$$
\delta_i^j(p_1, ..., p_n, t) = E_t \cdot \begin{cases}
p_i^j \oplus w^j \oplus B_{j-1} & if \; p_i \in^\bullet t \setminus t^\bullet, \\
p_i^j \oplus w^j \oplus C_{j-1} & if \; p_i \in t^\bullet \setminus p_i \in^\bullet t, \\
p_i^j \oplus w^j \oplus B_{j-1} & if \; p_i \in t^\bullet \cap p_i \in^\bullet t \wedge W(p_i,t) > W(t,p_i), \\
p_i^j \oplus w^j \oplus C_{j-1} & if \; p_i \in t^\bullet \cap p_i \in^\bullet t \wedge W(t,p_i) > W(p_i,t), \\
p_i^j & otherwise;
\end{cases}
$$

where the carry (C) and borrow (B) functions are defined as:

$$C_j = \begin{cases} 0 & if\ j = 0 \\ p_i^j \cdot w^j + C_{j-1} \cdot (p_i^j \oplus w^j) & if\ j \geqq 1 \end{cases}$$

$$B_j = \begin{cases} 0 & if\ j = 0 \\ \overline{p_i^j} \cdot w^j + B_{j-1} \cdot (\overline{p_i^j \oplus w^j}) & if\ j \geqq 1 \end{cases}$$

and finally, taking the appropriate constant value w^j in each case:

$$w^j = \begin{cases} W^j(p_i, t) & if\ p_i \in^\bullet t \setminus t^\bullet, \\ W^j(t, p_i) & if\ p_i \in t^\bullet \setminus p_i \in^\bullet t, \\ (W(p_i, t) - W(t, p_i))^j & if\ p_i \in t^\bullet \cap p_i \in^\bullet t \wedge W(p_i, t) > W(t, p_i), \\ (W(t, p_i) - W(p_i, t))^j & if\ p_i \in t^\bullet \cap p_i \in^\bullet t \wedge W(t, p_i) > W(p_i, t). \end{cases}$$

4.3.1.1 Simple example (Continued) Given the example in Fig. 17, the characteristic functions with binary encoding for each transition will be:

$$\begin{aligned} E_{t_1} &= (p_1^1 \cdot \overline{1}) \cdot (\overline{p_1^2 \oplus 0}) + (p_1^2 + \overline{0}) + (\overline{p_1^1 \oplus 1}) \cdot (\overline{p_1^2 \oplus 0}) = p_1^1 + p_1^2\ , \\ E_{t_2} &= (p_1^1 \cdot \overline{1}) \cdot (\overline{p_1^2 \oplus 0}) + (p_1^2 + \overline{0}) + (\overline{p_1^1 \oplus 1}) \cdot (\overline{p_1^2 \oplus 0}) = p_1^1 + p_1^2\ , \\ E_{t_3} &= (p_2^1 \cdot \overline{0}) \cdot (\overline{p_2^2 \oplus 1}) + (p_2^2 + \overline{1}) + (\overline{p_2^1 \oplus 0}) \cdot (\overline{p_2^2 \oplus 1}) = p_2^2\ , \\ E_{t_4} &= (p_3^1 \cdot \overline{0}) \cdot (\overline{p_3^2 \oplus 1}) + (p_3^2 + \overline{1}) + (\overline{p_3^1 \oplus 0}) \cdot (\overline{p_3^2 \oplus 1}) = p_3^2\ . \end{aligned}$$

The transition functions for each transition will be:

$$\begin{aligned} \delta_1^1(p_1, ..., p_3, t_1) &= E_{t_1} \cdot (p_1^1 \oplus 1 \oplus 0) = E_{t_1} \cdot (\overline{p_1^1})\ , \\ \delta_1^2(p_1, ..., p_3, t_1) &= E_{t_1} \cdot (p_1^2 \oplus 0 \oplus (\overline{p_1^1} \cdot 1)) = E_{t_1} \cdot (\overline{p_1^2 \oplus \overline{p_1^1}})\ , \\ \delta_2^1(p_1, ..., p_3, t_1) &= E_{t_1} \cdot (p_2^1 \oplus 1 \oplus 0) = E_{t_1} \cdot (\overline{p_2^1})\ , \\ \delta_2^2(p_1, ..., p_3, t_1) &= E_{t_1} \cdot (p_2^2 \oplus 0 \oplus (\overline{p_2^1} \cdot 1)) = E_{t_1} \cdot (\overline{p_2^2 \oplus \overline{p_2^1}})\ , \\ \delta_3^1(p_1, ..., p_3, t_1) &= E_{t_1} \cdot (p_3^1 \oplus 1 \oplus 0) = E_{t_1} \cdot (\overline{p_3^1})\ , \\ \delta_3^2(p_1, ..., p_3, t_1) &= E_{t_1} \cdot (p_3^2 \oplus 0 \oplus (\overline{p_3^1} \cdot 1)) = E_{t_1} \cdot (\overline{p_3^2 \oplus \overline{p_3^1}})\ . \end{aligned}$$

4.3.2 One-hot encoding

Assume that the number of tokens in place p is represented by a number of (one-hot encoded) Boolean variables $p^1, ..., p^{K_p}$, and that the weight $W(p,t)$ is represented by an integer constant. The relation $M(p) \geqq W(p,t)$ holds if any of the variables $p^w, ..., p^{K_p}$ is active (where $w = W(p,t)$). Hence, assuming that K_p is the number of Boolean variables required to represent a place p, the markings in which a transition t is enabled to fire are defined by the characteristic function:

$$E_t = \prod_{p \in {}^\bullet t} \left[\sum_{j=W(p,t)}^{K_p} p^j \right] .$$

The required token increase and decrease that corresponds to transition firing simply implies shifting left or right the active bit in each word $p^1, ..., p^{K_p}$ according to the number of tokens that must be subtracted or added respectively. For example, the relation $\delta(M,t) = E_t \cdot (M(p) - W(p,t))$ is implemented by shifting left $w = W(p,t)$ times the active variable, i.e. the set of equations:

$$\left(\delta^1(M,t), ..., \delta^{K_p}(M,t)\right) = E_t \cdot \left(p^{w+1}, p^{w+2}, ..., p^{K_p}, 0, .., 0\right) .$$

On the other hand, the relation $\delta(M,t) = E_t \cdot (M(p) + W(t,p))$ is implemented by shifting right $w = W(t,p)$ times the active variable, i.e. the set of equations:

$$\left(\delta^1(M,t), ..., \delta^{K_p}(M,t)\right) = E_t \cdot \left(0, ..., 0, \overline{p^1} \cdots \overline{p^{K_p}}, p^1, ..., p^{K_p - w - 1}, p^{K_p - w}\right) .$$

Note that the selected one-hot variables encode $M(p) = 0$ as $\overline{p^1} \cdot \overline{p^2} \cdots \overline{p^{K_p}}$. Therefore, increasing the number of tokens in p from 0 to $W(t,p)$ (i.e. activating variable p^w) requires detecting this special case.

The function δ_i^j for the j-variable encoding the tokens in a place p_i is described by:

$$\delta_i^j(p_1,...,p_n,t) = E_t \cdot \begin{cases} p_i^{j+w} & if\ (p_i \in^\bullet t \setminus t^\bullet \vee (p_i \in t^\bullet \cap p_i \in^\bullet t \\ & \quad \wedge W(p_i,t) > W(t,p_i))) \wedge (j \leqq K_p - w), \\ 0 & if\ (p_i \in^\bullet t \setminus t^\bullet \vee (p_i \in t^\bullet \cap p_i \in^\bullet t \\ & \quad \wedge W(p_i,t) > W(t,p_i))) \wedge (j > K_p - w), \\ p_i^{j-w} & if\ (p_i \in t^\bullet \setminus p_i \in^\bullet t \vee (p_i \in t^\bullet \cap p_i \in^\bullet t \\ & \quad \wedge W(t,p_i) > W(p_i,t))) \wedge (j > w), \\ \overline{p_i^1} \cdots \overline{p_i^{K_p}} & if\ (p_i \in t^\bullet \setminus p_i \in^\bullet t \vee (p_i \in t^\bullet \cap p_i \in^\bullet t \\ & \quad \wedge W(t,p_i) > W(p_i,t))) \wedge (j = w), \\ 0 & if\ (p_i \in t^\bullet \setminus p_i \in^\bullet t \vee (p_i \in t^\bullet \cap p_i \in^\bullet t \\ & \quad \wedge W(t,p_i) > W(p_i,t))) \wedge (j < w), \\ p_i^j & otherwise; \end{cases}$$

and again, taking the appropriate constant value w in each case:

$$w = \begin{cases} W(p_i,t) & if\ p_i \in^\bullet t \setminus t^\bullet, \\ W(t,p_i) & if\ p_i \in t^\bullet \setminus p_i \in^\bullet t, \\ W(p_i,t) - W(t,p_i) & if\ p_i \in t^\bullet \cap p_i \in^\bullet t \wedge W(p_i,t) > W(t,p_i), \\ W(t,p_i) - W(p_i,t) & if\ p_i \in t^\bullet \cap p_i \in^\bullet t \wedge W(t,p_i) > W(p_i,t). \end{cases}$$

4.3.2.1 Simple example (Continued) Given the example in Fig. 17, the characteristic functions with one-hot encoding for each transition will be:

$$E_{t_1} = p_1^1 + p_1^2\ ,$$
$$E_{t_2} = p_1^1 + p_1^2\ ,$$
$$E_{t_3} = p_2^2\ ,$$
$$E_{t_4} = p_3^2\ .$$

4.4 Boundedness verification

The overall transition system works under the assumption that no place can contain more than a given number of tokens - say k. This upper bound can be either self-imposed by the designer or limited by the number of Boolean variables allocated to represent the place; that is, the number of Boolean variables K_p should be least equal to $\lceil log_2(k+1) \rceil$ when using a binary encoding, or k when using a one-hot

encoding.

While using a binary encoding of places, a violation of the required boundedness condition for place p can be interpreted as an *overflow* in the operations to compute the actual number of tokens that should be placed in p after firing a transition $t \in^{\bullet} p$. Hence, each time a transition t is enabled to fire the k-boundedness of the Petri net can be verified by using the specific carry function in each place in the post-set of t; that is,

$$overflow(t) \equiv \left[From \cdot E_t \cdot \sum_{p \in t^{\bullet}} C_{K_p} \right] \neq \emptyset \ . \tag{17}$$

A second option is to verify if there exists any place that exceeds a particular tokens limit k at a given place p. Taking a binary encoding of the places, the upper bound k described as a set of constants $k^1, ..., k^{K_p}$, and since $(a > b) \equiv (a \cdot \overline{b})$, the relation $M(p) > k$ can described by the equation:

$$\begin{aligned} M(p) > k \ \equiv \ & (p^{K_p} \cdot \overline{k^{K_p}}) + \\ & (\overline{p^{K_p} \oplus k^{K_p}}) \cdot (p^{K_p - 1} \cdot \overline{k^{K_p - 1}}) + \\ & (\overline{p^{K_p} \oplus k^{K_p}}) \cdot (\overline{p^{K_p - 1} \oplus k^{K_p - 1}}) \cdot (p^{K_p - 2} \cdot \overline{k^{K_p - 2}}) + \\ & ..... \end{aligned}$$

Hence, the markings that contain more tokens than the upper limit k at place p are characterized by the equation:

$$B_k(p) = \sum_{i=1}^{K_p} \left[(p^i \cdot \overline{k^i}) \cdot \prod_{j=i+1}^{K_p} (\overline{p^j \oplus k^j}) \right] \ .$$

A similar characterization can be derived for a *one-hot* encoding of variables. The relation $M(p) > k$ holds if any of variables $p^{k+1}, ..., p^{K_p}$ is active. Hence, the markings that contain more tokens than the upper limit k at place p are characterized by the equation:

$$B_k(p) = \sum_{i=1}^{K_p} p^i \ .$$

5 Algorithms

Combining the Petri net analysis methods based on Boolean representation with the concept of hierarchical Petri nets, it is possible to develop efficient Petri net analysis algorithms. The purpose of this section to present out "decision of deadlock" algorithm for hierarchical Petri nets. Firstly, the question of firing transitions will be discussed. Then we will apply it to analyze the subnets, and we will use these results for deciding if there is any deadlock in the overall system.

5.1 The idea

Since the size of Petri nets usually very large and Petri net deadlock analysis is an $\mathcal{NP}$ problem the number of computational steps may grow exponentially with the size of the net. The idea is to reduce a Petri net into smaller parts (we called subnets) and analyze these subnets in about polynomial time. Therefor we can merge there results and finally make decision on the presence of a deadlock.

5.2 Partition problem

A basic problem in decomposing Petri nets is how we construct subnets. A subnet can be substituted either for a transition or a place. The substituting depends on the structure of subnet. If the subnet is beginning with a transition and ending in a transition, the subnet is substituting for a transition. In the other case, if the subnet is beginning with a place and ending in a place, the subnet is substituting for a place. The requirements of a substituting subnet are the following:

- there are one input connection place/transition and one output connection place/transition which connect the subnet to the supernet, and there isn't further connection between a subnet and the supernet and another subnets,
- a subnet has one input place/transition and one output place/transition.

5.3 Decision algorithm

For that to make the decision on the presence of deadlock of a Petri net with a given marking, it is enough if we can find at least one transition which is enabled and

doesn't cause overflow. If we can find a transition which fulfills the afore-mentioned requirements, the complete net has no deadlock with the given marking, otherwise the complete net has a deadlock with the given marking.

We must then analyze the supernet and one after the other all the subnets and merge the obtained information with the information of supernet. This process is repeated until we have found an appropriate transition or we have examined all subnets and haven't found an appropriate transition.

5.3.1 Analyzing a subnet

How we can analyze a subnet? During the analysis the algorithm computes the value of the characteristic function E_t of every transition t. If a transition is enabled, it must be checked whether it causes any overflow.

For analyzing we use the incidence matrix of a subnet and the given marking vector. The incidence matrix as it was defined in section 2.7.2.2 has the following constraints:

- if the subnet is substituting for a transition, in the incidence matrix of the sunbet the first transition being the input transition, and the last transitionin being the output transition.
- if the subnet is substituting for a place, in the incidence matrix of the sunbet the first place being the input place, and the last place being the output place.

These constraints simplify the computation.

5.3.2 Merge the information

After analyzing a subnet we only give some information which is relevant for deadlock analysis about it. This is obtained by searching an appropriate transition which is enabled in the subnet. If we can find one, the algorithm would terminate and give answer "live". Otherwise we can check the subnet and the supernet input and output connections. In this case we must to distinguish if the subnet is substituting for a transition or the subnet is substituting for a place.

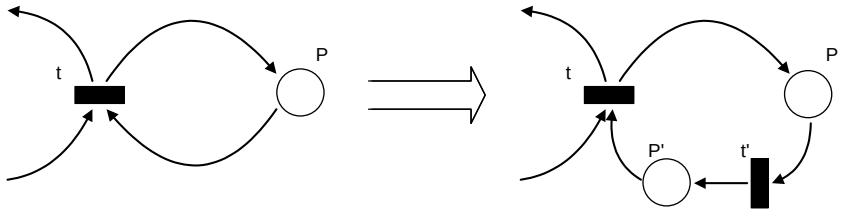


Figure 18: Self-loop elimination

The incidence matrix of the supernet has the following constraints:

- places which are denoted subnets being in the beginning in the rows of the incidence matrix,
- similarly, transitions which are denoted subnets being in the beginning in the columns of the incidence matrix.

The other requirement the analyzed Petri net doesn't contain self-loop. Because the incidence matrix can not represent the self-loop. So if the Petri net contains self-loop we must to set free it. We give a new transition and a new place into the Petri net as you can see in Fig. 18.

5.4 Developing the algorithms

We have developed two cases of the above mentioned algorithms. The first case is when the Petri net is safe, the second case is when the Petri net is k-bounded.
In the k-bounded case we use binary encoding for coding places and transitions. Together with the binary encoding we used the appropriate extended equations for k-bounded Petri nets (see the equations of the section 4).

6 Simulation results

This section contains simulation result which we used to verify and validate our deadlock testing algorithms for hierarchical Petri nets.

6.1 Description of the overall Petri nets

The implemented deadlock detection algorithm was tested on a previous example Petri nets which can be seen in Fig. 12 the safe case and Fig. **??** the k-bounded case.

6.2 The hierarchical models used

6.2.1 The safe case

We have decomposed the overall Petri net by two subnets in the Petri net as shown in Fig. 12. So we have following nets in our hierarchical Petri net structure:

- the supernet H (see in Fig. 11)
- the first subnet H_1 which is substituting for place P
- the second subnet H_2 which is substituting for transition T

First we construct the incidence matrices of the supernet and the subnets. For the construction of the incidence matrices *Visual Object Net++ v2.0a* [5] was used and we transformed them the necessary forms. The following incidence matrices describe the supernet H and the two subnets H_1 and H_2:

$$H = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \quad H_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \quad H_2 = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

6.2.2 The k-bounded case

We have decomposed the overall Petri net by three subnets in the Petri net. So we have following nets in our hierarchical Petri net structure:

- the supernet H (see in Fig. 19)
- the first subnet H_1 which is substituting for transition t_1
- the second subnet H_2 which is substituting for transition t_2
- the third subnet H_3 which is substituting for place p_3

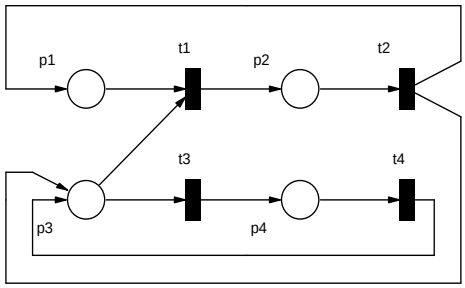


Figure 19: The supernet of the k-bounded Petri net

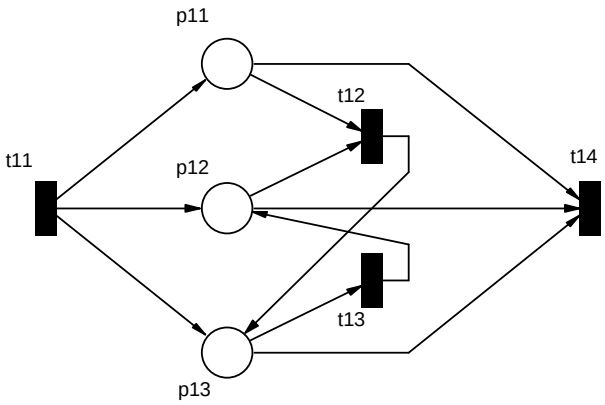


Figure 20: The subnet substituting for transition t_1

First we construct the incidence matrices of the supernet and the subnets. For the construction of the incidence matrix and we transformed them the necessary forms. The following incidence matrices describe the supernet H and the three

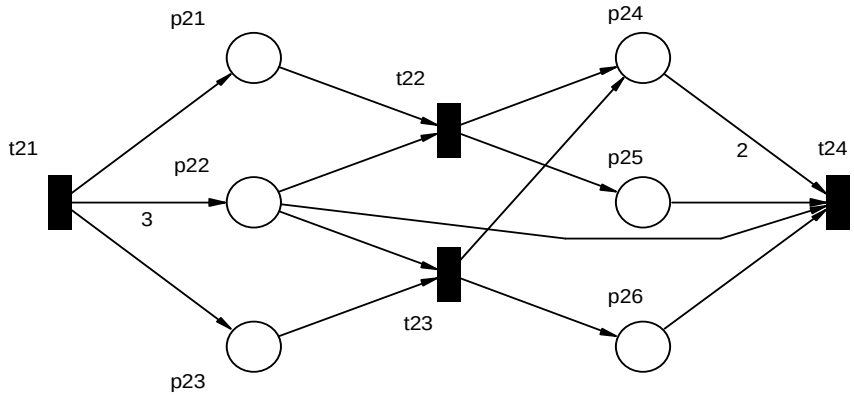


Figure 21: The subnet substituting for transition t_2

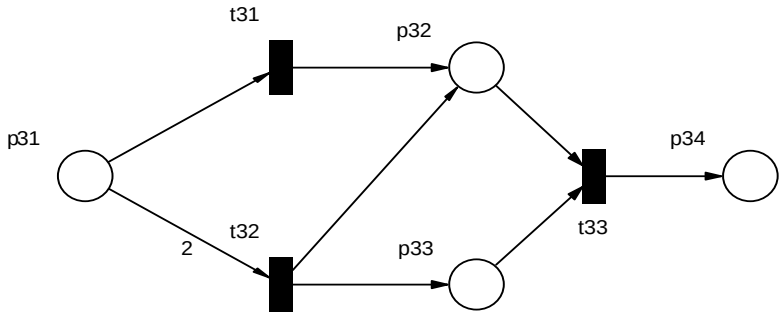


Figure 22: The subnet substituting for place p_3

subnets H_1, H_2 and H_3:

$$H = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 1 & -1 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad H_1 = \begin{bmatrix} -1 & 1 & 0 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -3 & 1 & 1 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 2 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad H_3 = \begin{bmatrix} 1 & 2 & 0 \\ -1 & -1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix}$$

In this case the bound (k) is 3. Therefor, we need 2 variables for binary encoding a value.

6.3 Implementation of the deadlock detection algorithm

The algorithms were implemented in Matlab's programming language[9]. The complete source code of the algorithms can be found in the Appendix (Section 8.1 – 8.5). Have we only give the input-output description of the main part of the source code.

Notation Since Matlab programming language doesn't use Boolean variables, so we should have correction this gap. The 1 denotes the value of the Boolean "True", and 0 denotes the value of the Boolean "False".

6.3.1 Detection algorithm

Safe Petri nets The commented source code of the method can be found in Appendix 8.1.
File name: SAFE_DETECTION.M
Inputs

`main_A`: the incidence matrix of the supernet
`main_M`: the marking vector of the supernet
`subnet_A`: the incidence matrix of all subnets
`subnet_M`: the marking vector of a subnets
`place_subnet_number`: the number of transitions in the supernet
`transition_subnet_number`: the number of transitions in a supernet

Output

the value of the return and written message: write "deadlock" and return 1 if the Petri net with a given marking is deadlock, otherwise write "live" and return 0

Operation

The procedure is the implementation of the algorithm described in Section 5.3. This method is specialized in safe Petri nets.

k-bounded Petri nets The commented source code of the method can be found in Appendix 8.3.

File name: BOUNDED_DETECTION.M

Inputs

`main_A`: the incidence matrix of the supernet

`main_M`: the marking vector of the supernet

`subnet_A`: the incidence matrix of all subnets

`subnet_M`: the marking vectors of all subnets

`place_subnet_number`: the number of transitions in the supernet

`transition_subnet_number`: the number of transitions in a supernet

`Bound`: the value of the bound

Output

the value of the return and written message: write "deadlock" and return 1 if the Petri net with a given marking is deadlock, otherwise write "live" and return 0

Operation

The procedure is the implementation of the algorithm described in Section 5.3. This method is specialized in k-bounded Petri nets.

6.3.2 Analyze a subnet

Safe Petri nets The commented source code of the method can be found in Appendix 8.2.

File name: SAFE_ANALYZE_SUBNET.M

Inputs

`A`: the incidence matrix of a subnet

`M`: the marking vector of a subnet

Output

`result`: combination of the characteristic functions of subnet (`Et`) and the overflow vector of a subnet (`Ov`)

Operation

The procedure is the implementation of the algorithm described in Section 5.3.1. This method is specialized in safe Petri nets. The characteristic function calculation uses the Eq. (6). The overflow checking uses the Eq. (12).

k-bounded Petri nets The commented source code of the method can be found in Appendix 8.4.

File name: BOUNDED_ANALYZE_SUBNET.M

Inputs

`code_A`: the encoded incidence matrix of a subnet

`code_M`: the encoded marking vector of a subnet

`Bound`: the value of the bound

Output

`result`: combination of the characteristic functions of subnet (`Et`) and the overflow vector of a subnet (`Ov`)

Operation

The procedure is the implementation of the algorithm described in Section 5.3.1. This method is specialized in k-bounded Petri nets. The characteristic function calculation uses the Eq. (15) which specialized in k-bounded case (Eq. (16)). The overflow checking uses the Eq. (12).

6.3.3 Encoding function for bounded analyze

The commented source code of the method can be found in Appendix 8.5.

File name: CODE.M

Inputs

`In`: the matrix what we want to encode

`Bound`: the value of the bound

Output

`result`: encoded `K`

Operation

The function is encode the input matrix with using binary encode.

6.4 Evaluation of the algorithms

As we see in the source code of the algorithms, the k-bounded case is more difficult than the safe case. As a result of the encoding we obtained much more variables that we could handle. So the conditions in the safe case are simple, but these simplest conditions are becoming complex and they aren't comprehended.

If we increase the bound, the number of the variables is exponentially increasing and the time of the evaluation of the conditions is increasing too. But if we restrict the size of any subnet, we can bound the execution time. Or if we restrict the maximum value of the bound k, we can also bound the execution time.

These algorithms are facilitated by the usage and the utilization of large Petri nets. Their usefulness comes into the surface as we get much more information by using them which we can't see because of the large size.

7 Conclusions and possible future work

Algorithms for deadlock analysis in hierarchical Petri nets have been proposed in this diploma work. The method is based on Boolean variable description of Petri nets and their operation and on their hierarchical decomposition. We have achieved partial results for hierarchical Petri nets, where we can detect deadlocks in a given Petri with a given initial marking.

Two variants of the deadlock detection method have been developed and implemented according to whether the Petri net is safe or k-bounded. Accordingly, I have constructed algorithms for two cases: safe and k-bounded Petri nets.

The idea of hierarchically decomposing a complex Petri net made the analysis much faster and more efficient. This is, because we usually don't use the whole net at the same time, just one subnet. Theis way the execution time is decreasing.

At the same time, however, the binary encoding leading to Boolean variables increases exponentially the number of variables in the k-bounded case, therefore it increases the execution time.

The algorithm for the safe ($k = 1$) case is used for analyzing a complex test problem. If we determine all the possible markings and execute this algorithm for every marking we can generate the set of all dead markings and the set of all live markings.

In the future there are several topics to be investigated further:

1. Generalize this algorithm for higher complexity (more than 2 levels) hierarchical Petri nets.

2. How we can efficiently transform a k-bounded Petri net into a safe one? What is the difference between the execution time of the algorithms.

3. How we can transform a colored Petri net into a safe one.? How will change the execution time for this case?

8 Appendix – Source code of the algorithms

8.1 The deadlock detection algorithm - safe Petri nets

```
%File:  SAFE_DETECTION.M
%Description:  Algorithm for decides a given hierarchical Petri net is deadlock
%               or live with a given initial marking
%Inputs:  main_A,main_M,subnet_A,subnet_M,place_subnet_number,
%          transition_subnet_number
%Output:  write ''deadlock'' and return 1 if the Petri net with a given marking
%          is deadlock, otherwise write ''live'' and return 0
%Author:  Erzsébet Németh
%Last modified:  05 03 2002
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
%analyze of the supernet
result=safe_analyze_subnet(main_A(place_subnet_number+1:size(main_A,1),
                transition_subnet_number+1:size(main_A,2)),
                main_M(place_subnet_number+1:size(main_A,1)));
main_Et=result(1:size(main_A,2)-transition_subnet_number);
main_Ov=result(size(main_A,2)-transition_subnet_number+1:size(result,2));
for a=1 :  transition_subnet_number
   main_Et=[0 main_Et];
   main_Ov=[-1 main_Ov];
end
main_m=size(main_A,2); % the number of transitions in the supernet
main_n=size(main_A,1); % the number of places in the supernet
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% for all subnets which are substituting for a place
for which_replaced=1 :  space_subnet_number
   % select the corresponding subnet and its marking vector
   sub_A=subnet_A[which_replaced];
   sub_M=subnet_M[which_replaced];
   % analyze the subnet
   result=safe_analyze_subnet(sub_A,sub_M);
   sub_Et=result(1:size(sub_A,2));
   sub_Ov=result(size(sub_A,2)+1:size(result,2));
   sub_m=size(sub_A,2); % the number of transitions in the supernet
   sub_n=size(sub_A,1); % the number of places in the supernet
%-------------------------------------------------------------------------------
   %search adequate transition
   i=1;
   while i <= sub_m
      if sub_Et(i) == 1 & sub_Ov(i) == 0
         'live'
```

```
        return 0
      end
      i=i+1;
   end
   % if there isn't a token in the input place
   if sub_M(1) == 0
      for k=1 :  main_m
         if main_A(which_replaced,k) == -1
            % transition in the supernet which consequence is the input place
            %  doesn't cause overflow
            main_Ov(k)=0;
         end
      end
   end
end
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% for all subnets which are substituting for a transition
for which_replaced=1 :  transition_subnet_number
   % select the corresponding subnet and its marking vector
   sub_A=subnet_A[place_transition_number+which_replaced];
   sub_M=subnet_M[place_transition_number+which_replaced];
   % analyze the subnet
   result=safe_analyze_subnet(sub_A,sub_M);
   sub_Et=result(1:size(sub_A,2));
   sub_Ov=result(size(sub_A,2)+1:size(result,2));
   sub_m=size(sub_A,2); % the number of transitions in the supernet
   sub_n=size(sub_A,1); % the number of places in the supernet
%-------------------------------------------------------------------------------
   %search adequate transition
   i=1;
   while i <= sub_m
      if sub_Et(i) == 1 & sub_Ov(i) == 0
         'live'
         return 0
      end
      i=i+1;
   end
   main_Et(which_replaced)=1;
   preset=0;
   for j=1 :  main_n
      if main_A(j,which_replaced)==1
         preset=1;
         main_Et(which_replaced)=main_Et(which_replaced) &
                    main_A(j,which_replaced)*main_M(j);
      end
```

```
   end
   if preset == 0 % if pre-set of t_(which_replaced) is empty set
      main_Et(which_replaced)=0;
   end
   contain=0;
   % recompute the value of sub_Ov(1)
   sub_Ov(1)=0;
   for j=1 :  sub_n
      if sub_A(j,1) == -1 % consequence of which_replaced
         contain=1;
         sub_Ov(1)=sub_Ov(1) | sub_M(j) ;
      end
   end
   if contain == 0
      sub_Ov(1)=0;
   end
   if main_Et(which_replaced) == 1 & sub_Ov(1) == 0
      'live'
      return 0
   end
   % analyze the output transition
   contain=0;
   % recompute the value of main_Ov(which_replaced)
   main_Ov(which_replaced)=0;
   for i=1 :  main_n
      if main_A(i,which_replaced) == -1 % consequence of which_replaced
         contain=1;
         main_Ov(which_replaced)=main_Ov(which_replaced) | main_M(i);
      end
   end
   if contain == 0
      main_Ov(which_replaced)=0;
   end
   if sub_Et(sub_m) == 1 & main_Ov(which_replaced)  = 1
      'live'
      return 0
   end
end
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% reanalyze the supernet
% is there an enabled transition which doesn't cause overflow?
i=1;
while i <= main_m
   if main_Et(i) == 1 & main_Ov(i) == 0
      'live'
```

```
     return 0
   end
   i=i+1;
end
'deadlock'
return 1
```

8.2 The subnet analyze algorithm - safe Petri nets

```
%File:  SAFE_ANALYZE_SUBNET.M
%Description:  Algorithm for compute the characteristic function for each
% transition and check the overflow
%Inputs:  A,M
%Output:  result
%Author:  Erzsébet Németh
%Last modified:  05 03 2002
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% initialization of variables and parameters
n=size(A,1); % the number of places
m=size(A,2); % the number of transitions
Et=zeros(1,m);
% E_t vector
     % E_t(i) = 1 if t_i is enabled
     % E_t(i) = 0 if t_i is not enabled or pre-set of t_i is empty set
Ov=zeros(1,m); % Overflow vector
     % Ov(i) = 1 if t_i is enabled and t_i causes overflow
     % Ov(i) = 0 if t_i is enabled and t_i doesn't cause overflow
     % Ov(i) = -1 if t_i isn't enabled
%-------------------------------------------------------------------------------
% beginning of the algorithm
i=0;
% for all transitions
while i < m
   i=i+1;
   Et(i)=1;
   preset=0;
   for j=1 :  n
      if A(j,i) == 1
         preset=1;
         Et(i)=Et(i) & A(j,i)*M(j);
      end
   end
   if preset == 0 % if pre-set of t_i is empty set
      Et(i)=0;
   end
```

```
%-------------------------------------------------------------------------------
   Ov(i)=-1;
   % Overflow-check for enabled transition
   if Et(i) == 1
      Ov(i)=0;
      preset=0;
      for j=1 :  n
         if A(j,i) == -1
            Ov(i)=Ov(i) | M(j);
            preset=1;
         end
      end
      if preset == 0 % if post-set of t_i is empty set
         Ov(i)=-1;
      end
   end
end
result=[Et Ov];
```

8.3 The deadlock detection algorithm - bounded Petri nets

```
%File:  BOUNDED_DETECTION.M
%Description:  Algorithm for decides a given hierarchical Petri net is deadlock
%               or live with a given initial marking
%Inputs:  main_A,main_M,subnet_A,subnet_M,place_subnet_number,
%         transition_subnet_number,Bound
%Output:  write "deadlock" and return 1 if the Petri net with a given marking
%         is deadlock, otherwise write "live" and return 0
%Author:  Erzsébet Németh
%Last modified:  05 03 2002
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% the number of required variables to encode values of elements of
%    incidence matrix and values of elements of marking vector
k=ceil(log2(Bound+1));
% encode the supernet
code_main_A=code(main_A,Bound);
code_main_M=code(main_M,Bound);
% analyze of the supernet
result=bounded_analyze_subnet(code_main_A(place_subnet_number+1:size(code_main_A,1),
               transition_subnet_number+1:size(code_main_A,2)),
               code_main_M(place_subnet_number+1:size(code_main_A,1)),Bound);
main_Et=result(1:size(main_A,2)-transition_subnet_number);
main_Ov=result(size(main_A,2)-transition_subnet_number+1:size(result,2));
for a=1 :  transition_subnet_number
   main_Et=[0 main_Et];
```

```
   main_Ov=[-1 main_Ov];
end
main_m=size(main_A,2); % the number of transitions in the supernet
main_n=size(main_A,1); % the number of places in the supernet
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% for all subnets which are substituting for a place
for which_replaced=1 :  place_subnet_number
   % select the corresponding subnet and its marking vector
   sub_A=subnet_A[which_replaced];
   sub_M=subnet_M[which_replaced];
   % encode the supernet
   code_al_A=code(sub_A,Bound);
   code_al_M=code(sub_M,Bound);
   % analyze the subnet
   result=bounded_analyze_subnet(code_al_A,code_al_M,Bound);
   al_Et=result(1:size(al_A,2));
   al_Ov=result(size(al_A,2)+1:size(result,2));
   al_m=size(al_A,2); % the number of transitions in the supernet
   al_n=size(al_A,1); % the number of places in the supernet
%-------------------------------------------------------------------------------
   %search adequate transition
   i=1;
   while i <= sub_m
      if sub_Et(i) == 1 & sub_Ov(i) == 0
         'live'
         return 0
      end
      i=i+1;
   end
   % if there isn't a token in the input place
   exist=0;
   for b=1 :  k
      if code_sub_M(which_replaced,b)
         exist=1;
      end
   end
   if exist == 0
      for b=1 :  main_m
         if code_main_A(which_replaced,b,k+1) == 1
            % transition in the supernet which consequence is the input place
            %  doesn't cause overflow
            main_Ov(b)=0;
         end
      end
   end
```

```
end
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% for all subnets which are substituting for a transition
for which_replaced=1 :  transition_subnet_number
   % select the corresponding subnet and its marking vector
   sub_A=subnet_A[place_transition_number+which_replaced];
   sub_M=subnet_M[place_transition_number+which_replaced];
   % encode the supernet
   code_sub_A=code(sub_A,Bound);
   code_sub_M=code(sub_M,Bound);
   % analyze the subnet
   result=bounded_analyze_subnet(code_sub_A,code_sub_M,Bound);
   sub_Et=result(1:size(sub_A,2));
   sub_Ov=result(size(sub_A,2)+1:size(result,2));
   sub_m=size(sub_A,2); % the number of transitions in the supernet
   sub_n=size(sub_A,1); % the number of places in the supernet

%-------------------------------------------------------------------------------
   %search adequate transition
   i=1;
   while i <= main_m
      if sub_Et(i) == 1 & sub_Ov(i) == 0
         'live'
         return 0
      end
      i=i+1;
   end
   main_Et(which_replaced)=1;
   exist=0;
   for j=1 :  main_n
      if code_main_A(j,which_replaced,k+1) == 0
         exist=1;
         result=bounded_analyze_subnet([code_main_A(j,which_replaced,:)],
                 [code_main_M(j,:)],Bound);
         enabled=result(1);
         main_Et(which_replaced)=main_Et(which_replaced) & enabled;
      end
   end
   if exist == 0 % if pre-set of t_(which_replaced) is empty set
      main_Et(which_replaced)=0;
   end
   contain=0;
   % recompute the value of sub_Ov(1)
   sub_Ov(1)=0;
   for j=1 :  sub_n
```

```
   if code_sub_A(j,1,k+1) == 1 % consequence of which_replaced
      contain=1;
      sub_Et(1)=1;
      newrow=zeros(1,sub_m);
      newrow(1)=1;
      result=bounded_analyze_subnet(code([sub_A;newrow],Bound),
                          code([sub_M;1],Bound),Bound);
      sub_Ov(1)=sub_Ov(1) | result(sub_m+1) ;
   end
end
if contain == 0
   sub_Ov(1)=0;
end
if main_Et(which_replaced) == 1 & sub_Ov(1) == 0
   'live'
   return 0
end
% analyze the output transition
contain=0;
% recompute the value of main_Ov(which_replaced)
main_Ov(which_replaced)=0;
for i=1 :  main_n
   if code_main_A(i,which_replaced,k+1) == 1 % consequence of which_replaced
      Overflow=0;
      contain=1;
      % for each post-set place compute the new value of M(p)
      % calculate the value of M(p)+W(t,p)
      carry_1=0;
      carry=0;
      newM_p=[];
      for l=1 :  k
         carry_i=carry;
         a=code_main_M(i,l);
         b=code_main_A(i,which_replaced,l);
         newM_p=[newM_p , (carry_1*( a* b+a*b)) | ( carry_1*(a* b+ a*b))];
         carry=carry_i*b | (a*(carry_i* b |  carry_i*b));
      end
      if carry == 1
         Overflow=1;
      else
         % newM_p > Bound ?
         valueM=newM_p(1);
         for q=2 :  k
            valueM=valueM+newM_p(q)*2^(q-1);
         end
         if valueM > Bound
```

```
            Overflow=1;
          end
        end
        main_Ov(which_replaced)=main_Ov(which_replaced) | Overflow;
      end
   end
   contain == 0
      main_Ov(which_replaced)=0;
   end
   if sub_Et(sub_m) == 1 & main_Ov(which_replaced)  = 1
      'live'
      return 0
   end
end
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% reanalyze the supernet
% is there an enabled transition which doesn't cause overflow
i=1;
while i <= main_m
   if main_Et(i) == 1 & main_Ov(i) == 0
      'live'
      return 0
   end
   i=i+1;
end
'deadlock'
return 1
```

8.4 The subnet analyze algorithm - bounded Petri nets

```
%File:  BOUNDED_ANALYZE_SUBNET.M
%Description:  Algorithm for compute the characteristic function for each
% transition and check the overflow
%Inputs:  code_A,code_M,Bound
%Output:  result
%Author:  Erzsébet Németh
%Last modified:  05 03 2002
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% initialization of variables and parameters
n=size(code_A,1); % the number of places
m=size(code_A,2); % the number of transitions
Et=zeros(1,m); % E_t vector
     % E_t(i) = 1 if t_i is enabled
     % E_t(i) = 0 if t_i is not enabled or pre-set of t_i is empty set
```

```
Ov=zeros(1,m); % Overflow vector
     % Ov(i) = 1 if t_i is enabled and t_i causes overflow
     % Ov(i) = 0 if t_i is enabled and t_i doesn't cause overflow
     % Ov(i) = -1 if t_i isn't enabled
%--------------------------------------------------------------------------------
% beginning of the algorithm
i=0;
% the number of required variables to encode values of elements of
%    incidence matrix and values of elements of marking vector
k=ceil(log2(Bound+1));
% for all transitions
while i < m
   i=i+1;
   Et(i)=1;
   % for all places
   summa=0;
   summa2=1;
   preset=0;
   for j=1 :  n
      if code_A(j,i,k+1) == 0
         preset=0;
         for l=1 :  k
            if code_A(j,i,l) > 0
               preset=1;
            end
         end
         if preset
            for ii=1 :  k
               product=1;
               volt=0;
               for jj=ii+1 :  k
                  product=product & ( code_M(j,jj) == code_A(j,i,jj));
                  volt=1;
               end
               summa=summa | ((code_M(j,ii) *  code_A(j,i,ii)) & product);
            end
            for l=1 :  k
               summa2=summa2 * (code_M(j,l) == code_A(j,i,l));
            end
         end
      end
   end
   summa=summa | summa2;
   Et(i)=Et(i) & summa;
   if preset == 0 % if pre-set of t_i is empty set
      Et(i)=0;
```

```
   end
%-------------------------------------------------------------------------------
   Ov(i)=-1;
   % Overflow-check for enabled transition
   if Et(i) == 1
      Ov(i)=0;
      postset=0;
      % for each post-set place compute the new value of M(p)
      for h=1 :  n
         if (code_A(h,i,k+1) == 1)
            % calculate the value of M(p)+W(t,p)
            postset=1;
            carry_1=0;
            carry=0;
            newM_p=[];
            for l=1 :  k
               carry_i=carry;
               a=code_M(h,l);
               b=code_A(h,i,l);
               newM_p=[newM_p , (carry_1*( a* b+a*b)) | ( carry_1*(a* b+ a*b))];
               carry=carry_i*b | (a*(carry_i* b |  carry_i*b));
            end
            if carry == 1
               Ov(i)=1;
            else
               % newM_p > Bound ?
               valueM=newM_p(1);
               for q=2 :  k
                  valueM=valueM+newM_p(q)*2^(q-1);
               end
               if valueM > Bound
                  Ov(i)=1;
               end
            end
         end
      end
      if postset == 0 % if post-set of t_i is empty set
         Ov(i)=-1;
      end
   end
end
result=[Et Ov];
```

8.5 The encode function - bounded Petri nets

```
%File:  CODE.M
```

```
%Description:  Encode the input with using binary encoding
%Inputs:  In,Bound
%Output:  result
%Author:  Erzsébet Németh
%Last modified:  05 03 2002
%--------------------------------------------------------------------------------
%--------------------------------------------------------------------------------
% the number of required bits for encoding a value kk=ceil(log2(k+1));
result=zeros([size(In) kk]);
for i=1 :  size(In,1)
   for j=1 :  size(In,2)
      if In(i,j) < 0
         result(i,j,k+1)=1; % sign bit
         In(i,j)=-In(i,j);
      end
      for h=1 :  kk
         if In(i,j) >= 2^(kk-h)
            result(i,j,kk-h+1)=1;
            In(i,j)=In(i,j)-2^(kk-h);
         end
      end
   end
end
```

References

[1] K. M. Hangos B. Kránicz, M. Gerzson. Invariant inheritance in structured operating procedures described by petri nets. *Computers and Chemical Engineering, Special Issue*, 22:969–972, 1998.

[2] S. Lafortune C. G. Cassandras. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[3] J. Cortadella R. M. Badia E. Pastor, O. Roig. Petri net analysis using boolean manipulation. *Proc. 15th International Conference Application and Theory of Petri Nets*, pages 416–435, June 1994.

[4] O. Roig E. Pastor, J. Cortadella. Symbolic analysis of bounded petri nets. *IEEE Transactions on Computers*, 50(5):432–448, May 2001.

[5] http://www.systemtechnik.tu-ilmenau.de/ drath/visual_e.htm.

[6] M. Gerzson K. M. Hangos, R. Lakner. *Intelligent Control Systems*, chapter 8, pages 153–190. Kluwer Academic Publishers, 2001.

[7] Tadao Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, Apr 1989.

[8] J.C. Madre O. Coudert, C. Berthet. Verification of sequential machines using boolean functional vectors. *Proc. IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Nov 1989.

[9] *MATLAB (V5.3) User's Guide*, 1999.