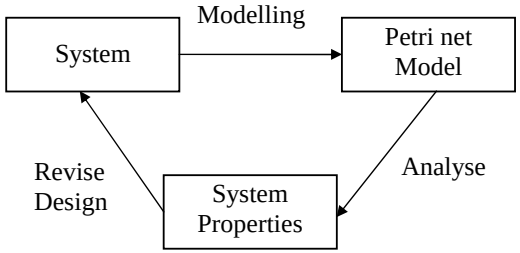
PETRI NET MODELLING

Petri nets are a form of computational model for designing and modelling systems, and are particularly useful where concurrent processing and dynamic sequential dependencies exist. Petri nets are applied through modelling the system as an abstract machine that embodies all the salient features of the system under study.

The most common approach using Petri nets is to use conventional design techniques to specify a system, then model it as a Petri net. The Petri net is analyzed and any problems encountered in the net suggest flaws in the original design, which is then modified, and the process iterates:



A more radical approach is to perform the entire design and specification process in terms of Petri nets (PNs). This approach requires an additional transformation of the PN model into a working system.

The major characteristics of PNs that make them suitable for systems modelling can be itemised:

- PNs support an explicit representation of causal dependencies and independencies.
- PNs support system description at various levels of abstraction.
- PNs represent system properties using the same methods as the system itself, e.g. explicit concurrency.
- Correctness proof systems use the same methods as system model construction.

Definition

A *Petri net structure* is a four-tuple C = (P, T, I, O) where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of *places*
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of *transitions*
- $I : T \rightarrow P^{\infty}$ is an *input function*
- $O : T \rightarrow P^{\infty}$ is an *output function*

A place p_i is an *input place* of transition t_j if $p_i \in I(t_j)$
A place p_i is an *output place* of transition t_j if $p_i \in O(t_j)$
The *multiplicity* of input and output places can be defined as $\#(p_i, I(t_j))$ and $\#(p_i, O(t_j))$ respectively.

Example: C = (P, T , I, O), $P = \{p_1, p_2, p_3, p_4, p_5\}$, $T = \{t_1, t_2, t_3, t_4, t_5\}$

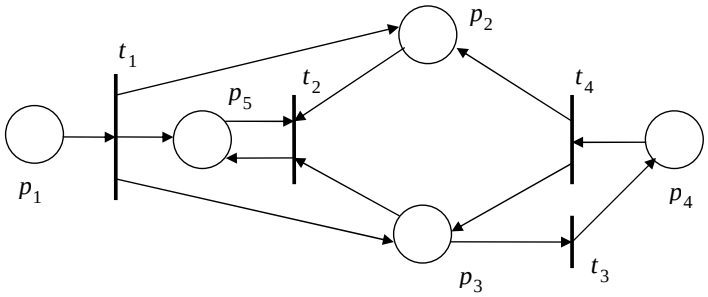
$I(t_1) = \{p_1\}$, $O(t_1) = \{p_2, p_3, p_5\}$
$I(t_2) = \{p_2, p_3, p_5\}$, $O(t_2) = \{p_5\}$
$I(t_3) = \{p_3\}$, $O(t_3) = \{p_4\}$
$I(t_4) = \{p_4\}$, $O(t_4) = \{p_2, p_3\}$

Definition

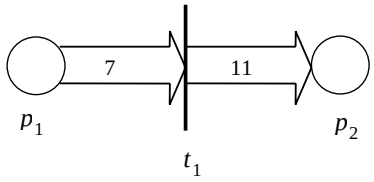
A *Petri net graph* G = (V, A) , is a bipartite directed multi-graph where:

- $V = \{v_1, v_2, \ldots, v_s\}$ is a set of *vertices*
- $A = \{v_1, v_2, \ldots, v_s\}$ is a *multiset* (or *bag*) of *directed arcs*.
- The set V can be partitioned into two disjoint sets P and T where P and T are defined in the Petri net structure, and $V = P \cup T$ and $P \cap T = \varnothing$.
- Each directed arc $a_i \in A$ has vertices, i.e. $a_i = (v_j, v_k)$ and $v_j \in P$, $v_k \in T$ or $v_j \in T$, $v_k \in P$.
- The graphical notation assigns circles centred on vertices $v_j \in P$ (or places), and bars centred on vertices $v_k \in T$ (or transitions).

Example: the previous example PN structure can also be represented graphically as:



As a notational convenience, where we have a high multiplicity of arcs on the PN graph, then they can be *bundled*, e.g:



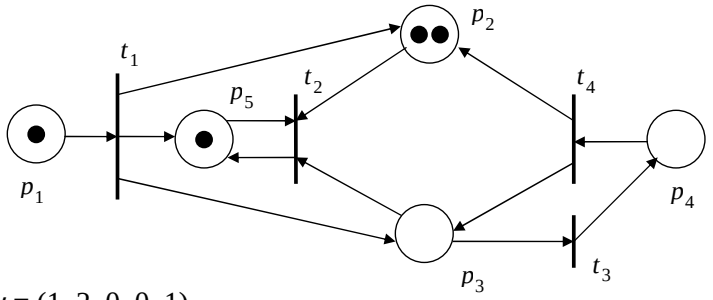
Definition

A *marking* μ of a Petri net C = (P, T, I, O) is a function from the set of places P to the non-negative integers $\mathtt{N}$, i.e. $\mu : P \rightarrow \mathtt{N}$. The PN marking can also be defined as an n-vector, $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$ where $n = |P|$, and it can be interpreted as a number of *tokens* μ_i in each place p_i for $i = 1, \ldots, n$.

A *marked* Petri net M = (C, μ) is a Petri net structure C with marking μ. On the graphical representation, tokens are denoted by "•" which are confined to places. As the number of tokens in any place is (theoretically) unbounded, there is an infinite set of possible markings for a PN.

For notational convenience, when the number of tokens becomes too large to represent with •'s, a number is used within the place.

Example



$\mu = (1, 2, 0, 0, 1)$

Execution Rules for Petri nets

The *execution* of a PN is controlled by the number and distribution of tokens in the PN. A PN executes by *firing* transitions which fire by removing tokens from their input places and creating new tokens to be distributed to their output places.

Transitions can only fire if *enabled* by having at least as many tokens in their input places as the number of arcs from the input places. These tokens are referred to as *enabling tokens*, and multiple tokens are required for multiple input arcs.

A transition $t_j \in T$ is enabled for all $p_i \in P$ where:

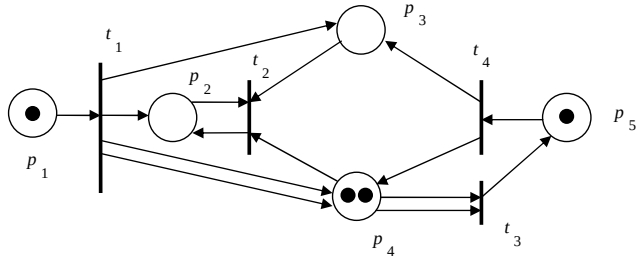
$$\mu(p_i) \geq \#(p_i, I(t_j))$$

and it may fire when enabled to produce a new marking μ' which is defined by:

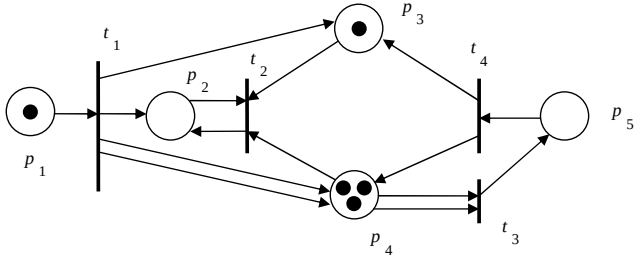
$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))$$

Transitions can continue to fire as long as at least one is enabled, otherwise execution *halts*.

Examples



Enabled transitions: t_1, t_3, t_4



Fire transition t_4, enabled transitions: t_1, t_3

Petri Net State Spaces

The state space of a PN with n places is the set of all markings, i.e. $\mathbb{N}^n$.

A *next-state function* can be defined, denoted by δ, which gives the change in markings after a transition fires. Thus if a transition t_j is enabled, then $\delta(\mu, t_j) = \mu^1$.

Given an initial marking μ^o, firing an enabled transition t_j produces a new marking $\mu^1 = \delta(\mu^o, t_j)$. From this marking, another enabled transition t_k can fire to produce a new marking $\mu^2 = \delta(\mu^1, t_k)$, and so on. Thus the execution of a PN can be defined by a *sequence of markings* $(\mu^o, \mu^1, \ldots)$ and a *sequence of transitions* $(t_{jo}, t_{j1}, \ldots)$ which are connected through the relationship:

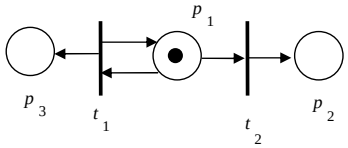
$$\delta(\mu^k, t_{jk}) = \mu^{k+1}, \text{ for } k = 0, 1, 2, \ldots$$

For a Petri net C = (P, T, I, O) with marking μ, a new marking μ' is said to be *immediately reachable* from μ if there exists a transition $t_j \in$ T such that $\delta(\mu, t_j) = \mu'$.

If a marking μ' is immediately reachable from some marking μ and a marking μ'' is immediately reachable from the marking μ', then μ'' is *reachable* from μ.

A *reachability set* R(C, μ) for a PN C = (P, T, I, O) can be defined as the smallest set of markings for which if $\mu \in$ R(C, μ) and $\mu' \in$ R(C, μ), where $\mu'' = \delta(\mu', t_j)$ for some $t_j \in$ T, then $\mu'' \in$ R(C, μ).

Example



μ = (1,0,0), immediately reachable markings are (0,1,0) and (1,0,1), and then (0,1,1) and (1,0,2) and so on.
Thus R(C, μ) = {(1,0,n}, (0,1,n) | $n \geq 0$}.

An *extended next-state function* can be defined to map a marking and a *sequence* of transitions into a new marking. Given a sequence of transitions $t_{j1}, t_{j2}, \ldots, t_{jk}$ and a marking μ, a new marking $\mu' = \delta^*(\mu, t_{j1}, t_{j2}, \ldots, t_{jk})$ results from the firing of the sequence of transitions.

Alternative Basic Petri Net Forms

A PN can also be represented as:

- a 4-tuple C = (P, T, F, B) where F and B are functions mapping places and transitions into the number of tokens needed for input (F) or produced for output (B)
- a triple C = (P, T, A) where A is a set of arcs
- C = TP where TP is a structure of appropriate pairs of places given linking between them through arcs and transitions.

Modelling with Petri Nets

There are two primitive concepts in the Petri net view of a system's behaviour: *events* and *conditions*. Events are system actions which are controlled by the state of the system, and the system state can be described by a set of conditions.

For events to occur certain conditions must hold, i.e. *preconditions*. On event occurrence, the preconditions may cease to hold and other conditions, i.e. *postconditions* become true.

Example

Consider a simple machine shop modelling problem, in which a machine waits until an order appears, then machines the ordered part and sends it out for delivery.

The *conditions* for the system are:

a. Machine shop is waiting
b. Order has arrived and is waiting
c. Machine shop is processing order
d. Order is complete

and the *events* are:

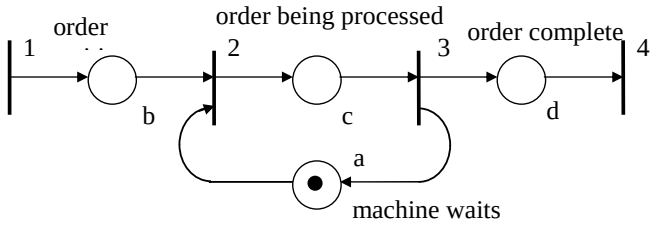
1. Order arrives
2. Machine shop starts on order
3. Machine shop finishes the order
4. Order is sent for delivery

Represented as a event/condition table:

Event	Preconditions	Postconditions
1	-	b
2	a, b	c
3	c	d, a
4	d	-

To map this to a PN it is only necessary to recall that conditions are modelled by places and events are modelled by transitions.

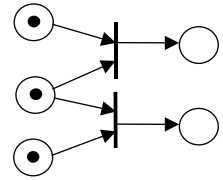
Preconditions are inputs to transitions and postconditions are outputs from transitions:



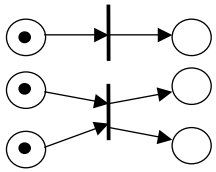
Tokens are used to represent the holding of a condition at initialisation.

The PN model allows non-interacting enabled events to occur independently without synchronization, i.e. PNs are asynchronous by nature - there is no inherent measure or flow of time built in. Time is indirectly represented as a partial ordering of the occurrence of a sequence of events which can take varying amounts of 'real' time.

Concurrency in PNs is simply modelled as independent nondeterministic and non-simultaneous firing of transitions:

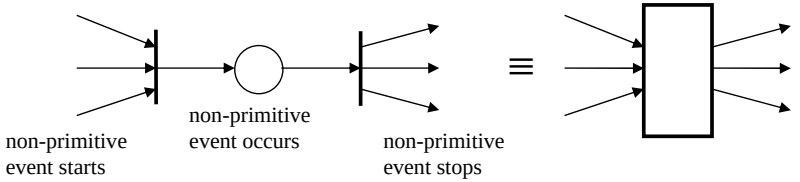


Conflict in PNs is modelled by coupling transitions through common places (hence sharing common preconditions):



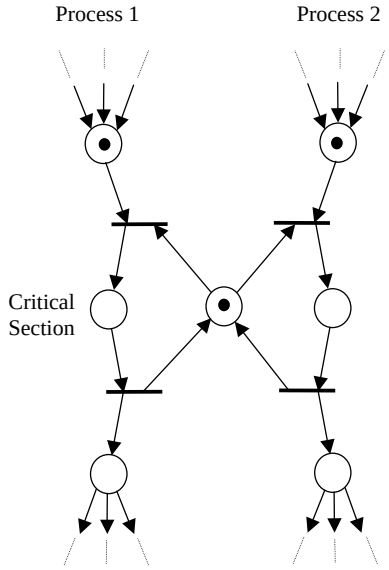
PNs also execute *nondeterministically*, i.e. the selection of which transition to fire when several are enabled is made randomly. In the basic PN model, transition firings are considered to be instantaneous and event occurrences cannot be simultaneous (since time is a continuous real variable). Conventional PN events are taken to be *primitive* - i.e. instantaneous and non-simultaneous.

Non-primitive events can take non-zero-time and thus may overlap other events - but they can be decomposed into primitive events, e.g:



The same non-zero width transition bar is useful for abstracting multi-level PNs in a hierarchical structure.

Although PNs are inherently asynchronous they can be used to model synchronisation type problems, e.g mutual exclusion mechanisms, where only one process can access a shared data object at a time:



PNs can also be applied to other 'classical' synchronisation problems, such as the producer/consumer problem, the dining philosopher's problem, the reader/writer problem, and P/V operations on semaphores.

PNs have also been applied to a wide variety of non-computational modelling applications, e.g. planning and scheduling on large projects, chemical reaction systems, communication networks, brain models, the rules of propositional calculus, and legal systems.

See J.L. Petersen, "Petri net theory and the Modelling of Systems", 1981, Prentice-Hall, for more examples.

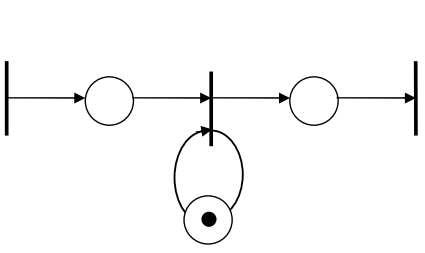
PETRI NET ANALYSIS

Given a Petri net (PN) model of a system it should be possible to analyse it to lead to useful insights into the behaviour of the system. Thus it is useful to consider what types of problems can be solved with PNs and the important properties that can be used to classify PN behaviour.

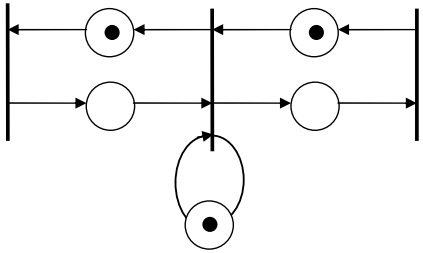
Safeness

A place in a PN is *safe* if the number of tokens in that place cannot exceed one. A PN is *safe* if all places in the PN are *safe*, i.e. a place $p_i \in P$, for a PN C = (P, T, I, O) with a initial marking μ, is *safe* if:

$$\forall \mu' \in R(C,\mu),\ \mu'(p_i) \leq 1$$



This PN is not safe



This PN has been made safe

Boundedness

Safeness is a special case of a more general boundedness property: a place is *k-safe* or *k-bounded* if the number of tokens cannot exceed *k* at that place, i.e: a place $p_i \in P$, for a PN C = (P, T, I, O) with a initial marking μ, is *k-safe* if:

$$\forall \mu' \in R(C,\mu),\ \mu'(p_i) \leq k$$

If every place is at worst *k-safe* then the PN can be described as *k-safe*, and a PN is *bounded* if all places are *bounded*.

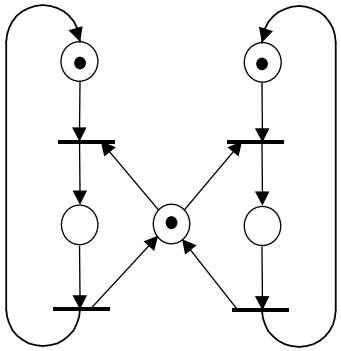
Conservation

For PNs that model resource allocation systems, *conservation* of tokens (hence resources) is an important property. A PN C = (P, T, I, O) with initial marking a μ, is *strictly conservative* if:

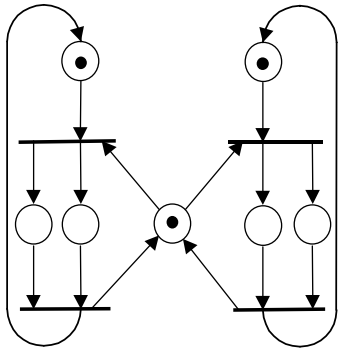
$$\forall \mu' \in R(C,\mu), \quad \sum_{p_i \in P} \mu'(p_i) = \sum_{p_i \in P} \mu(p_i)$$

For this strongly constraining relationship to hold, it should be clear that the number of inputs to each transition must equal the number of outputs, i.e. $|I(t_j)| = |O(t_j)|$.

Consider the examples:



A non-strictly conservative PN



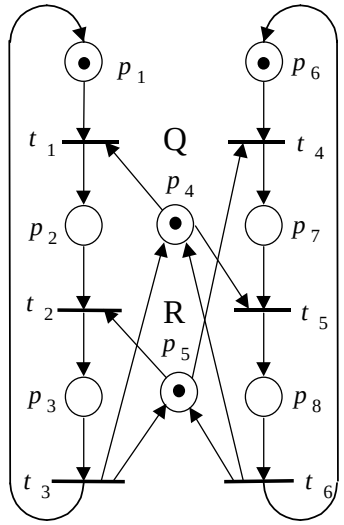
An equivalent strictly conservative PN

In general, there may not be a simple one-to-one mapping between tokens and resources as some tokens may represent several resources. To generalize the conservation concept, a *weighed sum* of all reachable markings should be constant, i.e. a weighting factor w_i can be applied to each place and we then have:

$$\sum_i w_i \mu'(p_i) = \sum_i w_i \mu'(p_i)$$

Liveness

Another problem that may arise in resource allocation is *deadlock* - for example, where a process A has resource Q and a process B has a resource R and wants resource Q → neither process can proceed. This situation can be modelled by a PN:



Process A Process B

Where in the above p_4 represents resource Q and p_5 represents resource R. We can see that transition firings in the sequence $t_1t_2t_3t_4t_5t_6$ or $t_4t_5t_6t_1t_2t_3$ do not result in deadlock, but any sequence starting with t_1t_4 or t_4t_1 results in deadlock.

In a PN, a transition is live if it is not deadlocked, i.e. if it can be enabled. Thus a transition t_j of a PN, C is *potentially fireable* in a marking μ if $\exists \mu' \in R(C,\mu)$ and t_j is enabled in μ'.

A *level* of liveness can also be specified for each transition in a PN, from level 0 (dead) to level 4 (live for all markings) and intermediate levels specify liveness for an increasing number of possible firing sequences.

Reachability and Coverability

Given a PN C with marking μ and a marking μ' the following question can be posed: Is $\mu' \in R(C,\mu)$? This can be an important question, for example in the earlier example if a marking $\mu = (0,1,0,0,0,0,1,0)$ is *reachable*, then deadlock can occur.

A closely related PN analysis problem is *coverability* and it can be stated as: Given a PN C with initial marking μ and a marking μ', is there a reachable marking $\mu'' \in R(C,\mu)$ such that $\mu'' \geq \mu'$ (i.e. is the marking μ' covered by some other marking μ'').

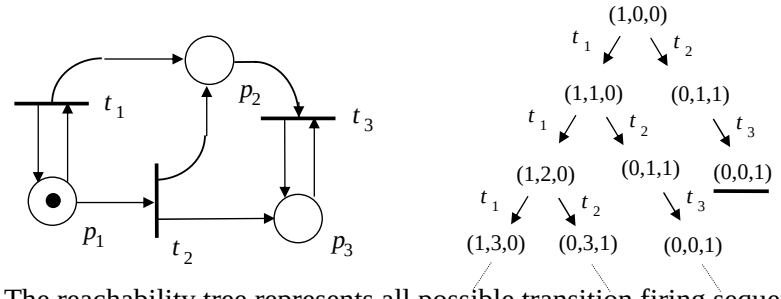
Interest in reachability or coverability can be confined to a few key places (say representing resources) so a *submarking reachability* or *submarking coverability* modulo a set of places can be defined.

Firing Sequences

Another analysis approach is to concentrate on sequences of transition firings, as distinct from state changes. This approach is more useful to resolve questions of liveness, for example, in the earlier deadlock PN example, firing sequences t_1t_4 or t_4t_1 result in deadlock. This analysis question has a direct connection with the *languages* associated with PNs.

The Reachability Tree

Consider the following PN which has initial marking (1,0,0) and two reachable markings resulting from firing the two enabled transitions:

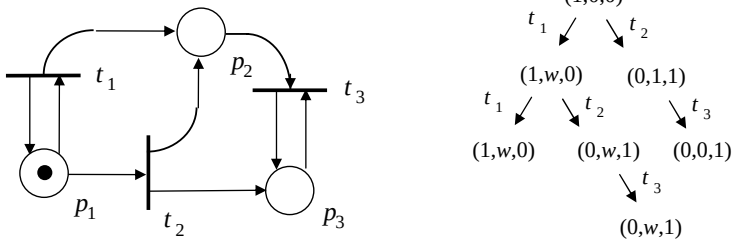


The reachability tree represents all possible transition firing sequences, and it is easy to see that a PN with infinite reachability set will have a infinite reachability tree.

For analysis purposes, it is useful to limit the tree to a finite size, i.e. the marking pattern that is repeated can be extracted and an arbitrarily large number of tokens represented with the symbol '*w*'.

Example

For the 3 place/3 transition PN below, a finite reachability tree can be constructed:



The reachability tree can now be used an analysis tool for some of the PN properties that have been defined earlier.