




Petri Nets: Properties, Applications, and Variations

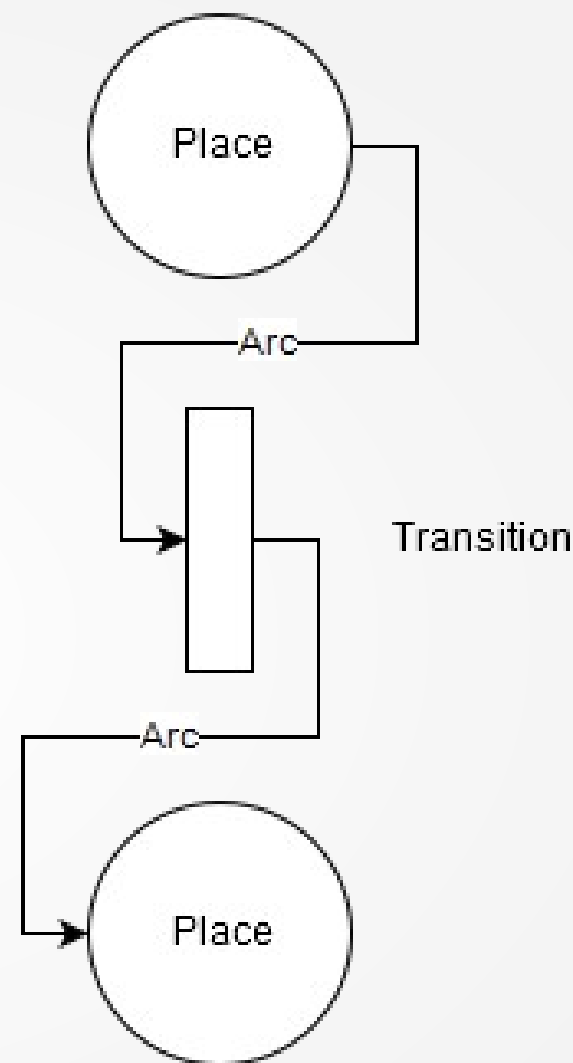


Matthew O'Brien
University of Pittsburgh

Introduction

- A Petri Net is a graphical and mathematical modeling tool used to describe and study information processing systems of various types.
- Petri Nets originate from the dissertation of Carl Adam Petri to the faculty of Mathematics and Physics at the Technical University of Darmstadt, West Germany in 1962.
- As a mathematical tool, it can be used to set up algebraic equations, state equations, and other mathematical models governing systems.
- Due to the nature of the tool, it also lends itself rather handily to the modeling of logical systems, including those that may occur in computer science or communication systems.

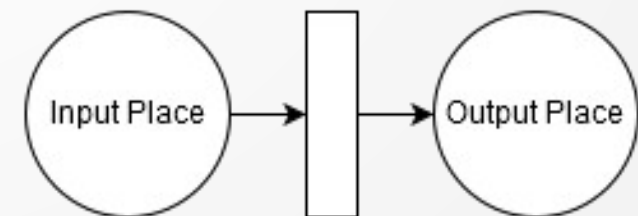
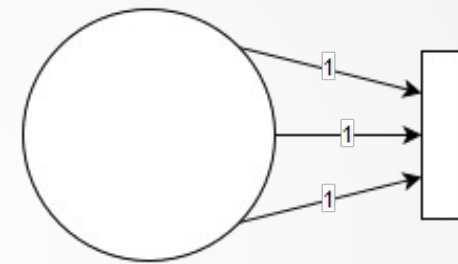
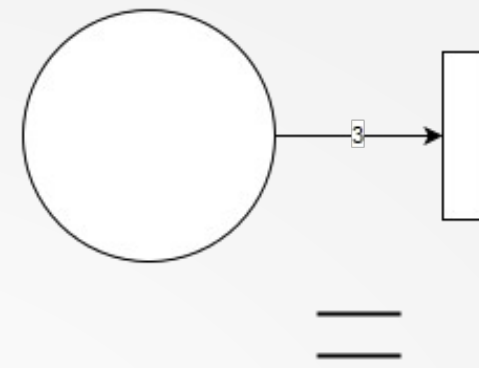
- A petri net is drawn as a directed, weighted, bipartite graph.
- A bipartite graph is a graph with two distinct sets of nodes such that there are no edges between nodes of the same set.
- In the case of petri nets, these two sets are defined as *places* and *transitions*.
- Typically, places are represented as circles and transitions as either bars or boxes.
- The edges between nodes are defined as *arcs*.





- A petri net has states, designated as *markings*. Each marking corresponds to an assignment of some non-negative integer k to each place p .
- This corresponds to the place being 'marked' with k *tokens*, which are represented on the graph typically as smaller black circles within each place.
- Normally, there is no limit to the number of tokens which may mark a given place.

- An arc, either from a place to a transition or from a transition to a place, has some weight k .
- An arc with weight k is functionally the same as there being k arcs of weight 1.
- A place with an arc from itself to a transition is an *input place*, and a place with an arc from a transition to itself is an *output place*.



Formally, a Petri Net is defined as a 5-Tuple in the form: $PN=(P,T,F,W,M_0)$

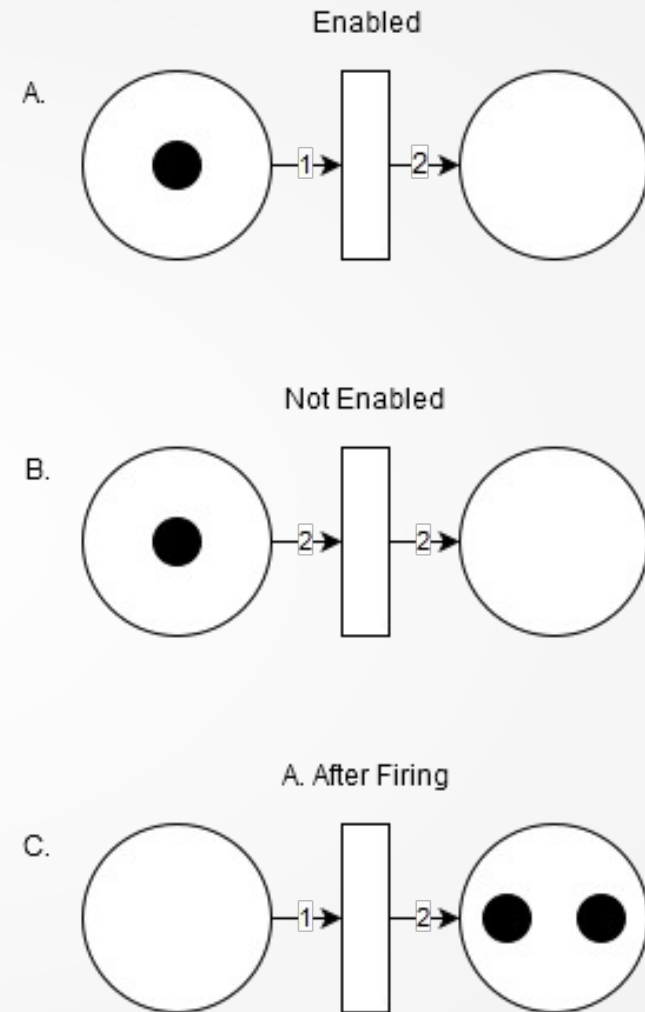
Where:

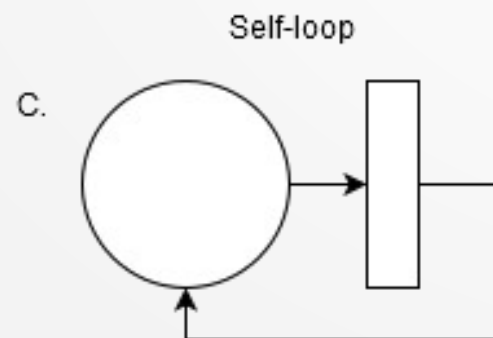
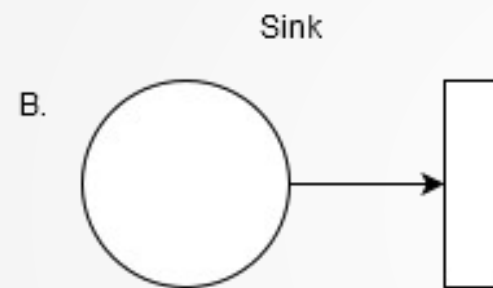
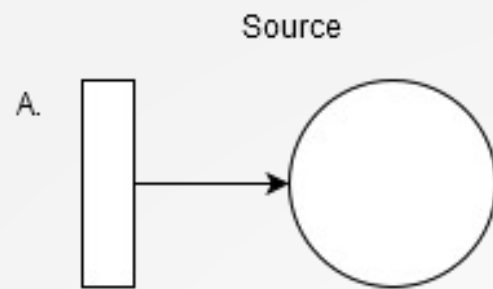
- $P = \{p_1 \dots p_n\}$ is a finite set of places.
- $T = \{t_1 \dots t_n\}$ is a finite set of transitions.
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs.
- $W : F \rightarrow \{1,2,3 \dots\}$ is a weight function.
- $M_0 : P \rightarrow \{0,1,2 \dots\}$ is some initial marking.
- $P \cap T = \{\}$
- $P \cup T \neq \{\}$

A Petri Net structure with no initial marking may be defined as $N=(P,T,F,W)$.

Likewise, given a structure N with no initial marking, a Petri Net may be defined as $PN = (N,M_0)$.

- Given this formalization, there is a *firing rule*, which has three parts.
 - A transition t is *enabled* if each input place p of t is marked with at least $w((p,t))$ tokens (where $w((p,t))$ is the weight of the arc from p to t).
 - An enabled transition t may or may not fire.
 - The *firing* of an enabled transition t removes $w((p,t))$ tokens from each input place p of t and adds $w((t,p'))$ tokens to each output place p' of t .





A transition does not need to have both input and output.

- A transition with no input places is referred to as a *source transition* and a transition with no output places is called a *sink transition*. A source transition is always enabled.
- A tuple (p,t) is called a *self-loop* if p is both an input place and output place of t . A petri net which contains no self loops is called *pure*.
- A petri net whose arcs all have weight of 1 is called *ordinary*.

Applications

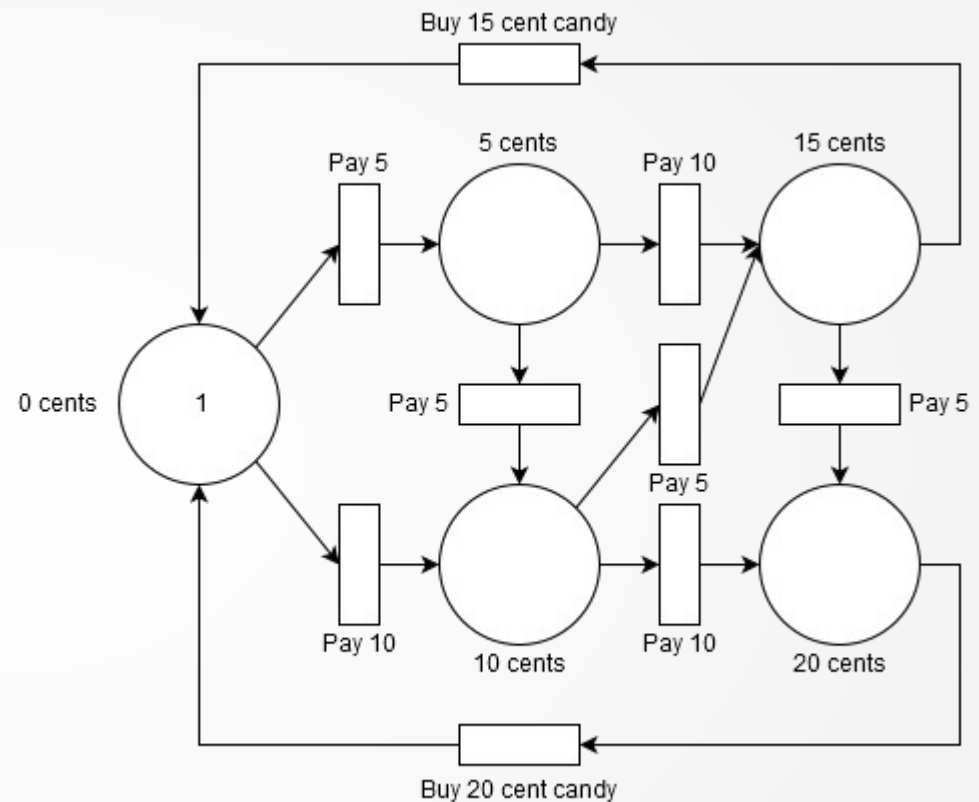
There are a number of different scenarios which petri nets are particularly useful in modeling.

In this section, some of the examples are meant to show potential applications of petri nets in computation/system modeling, while a few of them are intended to express the power of the petri net as a model in general.

State Machines:

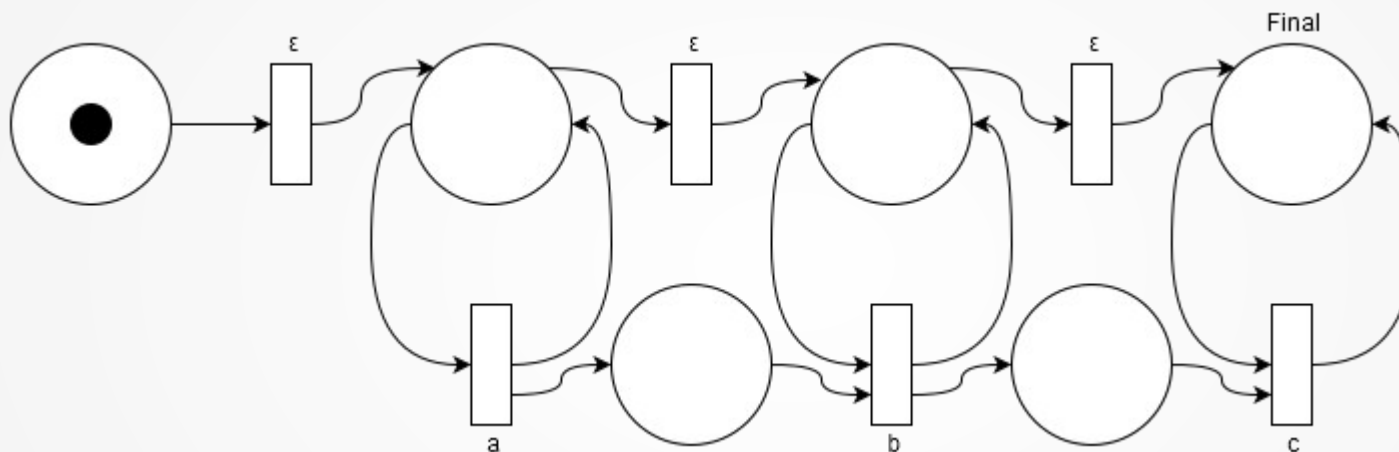
This example is somewhat trivial, a *pure, ordinary* petri net with an initial marking of one token in a 'start' state represents a state machine.

The petri net to the right represents the state of a candy machine which only accepts nickels and dimes, doesn't return change, and sells 15 and 20 cent candies. Note that while a petri net's transitions are non-deterministic, this does not mean it cannot model deterministic machines.



Formal Languages:

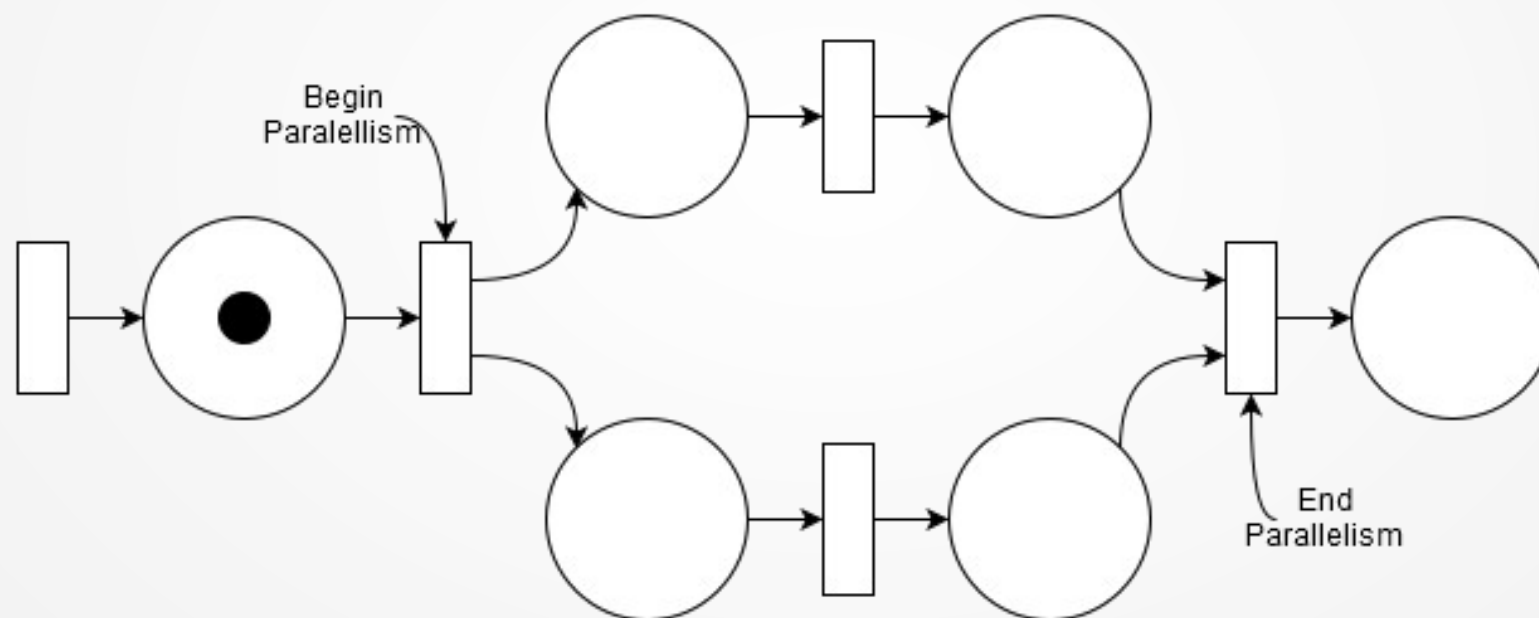
The following petri net recognizes the formal language of $L = \{a^n b^n c^n \mid n \geq 0\}$, which is a context sensitive language. For the following net, we say a string has been recognized iff the final petri net marking has a token only in the place labeled 'final'.

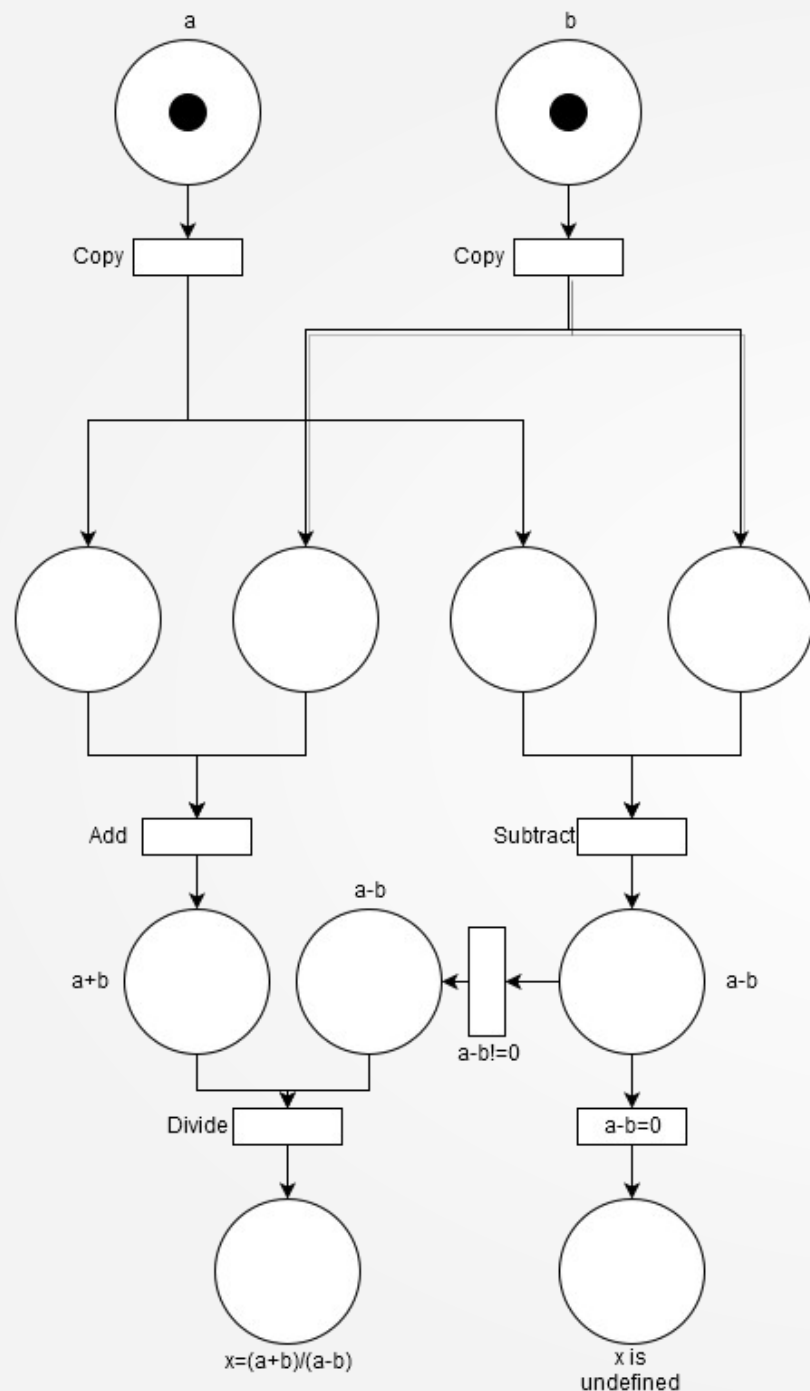


Following this, it is the case that a Petri Net can recognize any context sensitive, context free, or regular language. It follows that the modeling power of the Petri-net is equivalent to that of a linear bounded Turing Machine. There is a variant of the Petri-net whose power is equal to that of a full Turing Machine.

Parallelism:

Consider the simple case where we have a program that does one thing, then splits into two threads each of which performs an independent task, then continues in a single thread once both threads are done. We could model that behavior as follows.



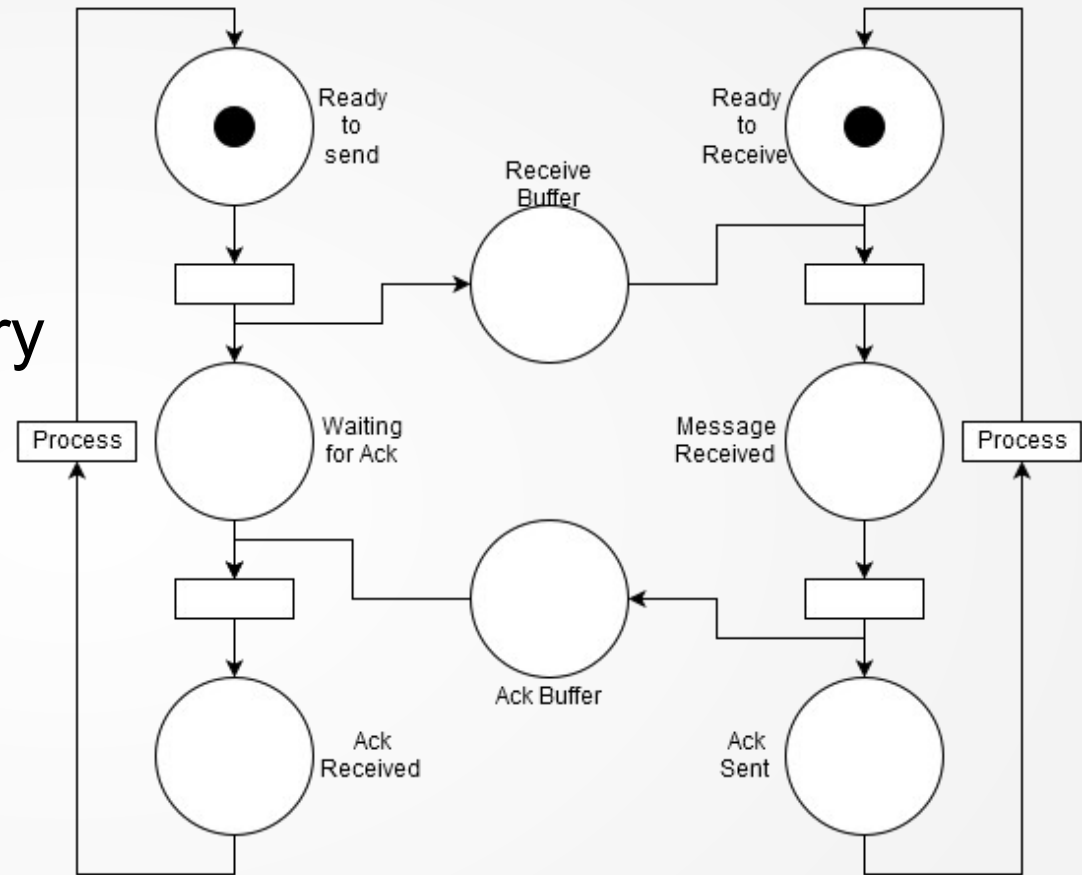


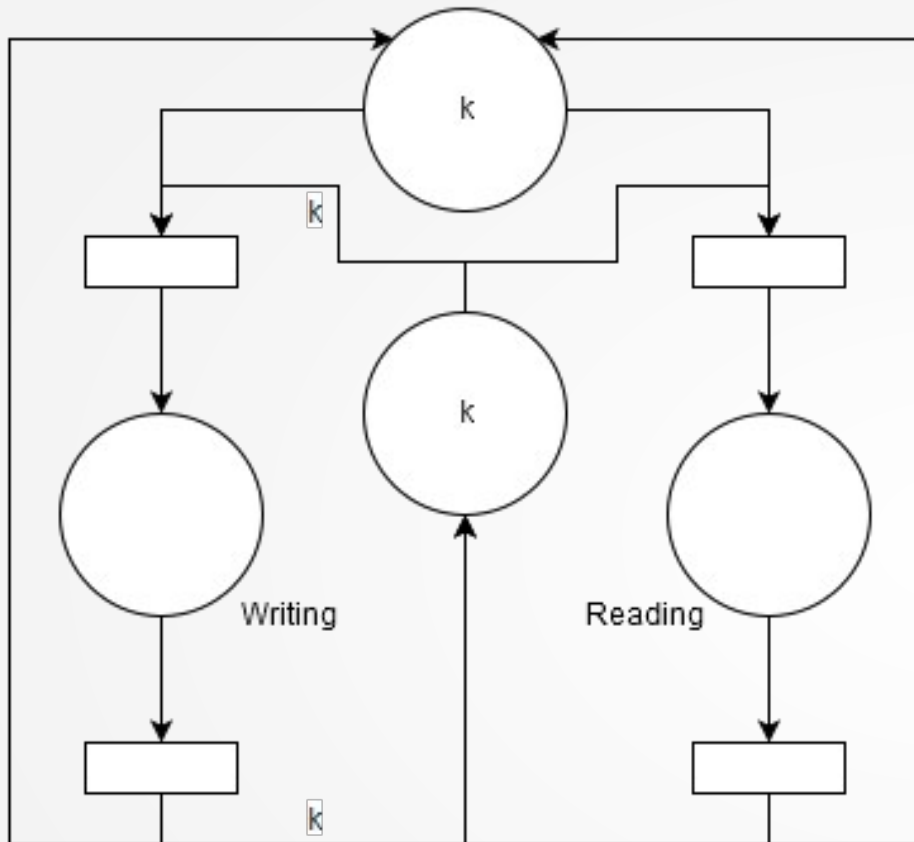
Data Flow:

A petri net can be used to map out the flow of data through a calculation. By assigning names to the places which represent value, we can easily model a calculation, where a token in a place marks that the data it represents is available for use.

The petri net to the left represents the dataflow for the calculation: $x = (a+b)/(a-b)$. Note that the petri net does not contain any mechanism to actually perform the calculations as depicted.

Communication Protocols:
 Consider a basic communication protocol; you send a message of some number of bytes, and while your message is never lost in transit, your receiver has a very limited buffer. As such, you must wait until the receiver processes the message and sends a acknowledgment before sending the next message. This can easily be modelled with petri nets.





Synchronization Control:
Petri nets can model various synchronization mechanisms, like mutex, read-write synchronization, etc.

To the left is a model of a read write system, the tokens in p_1 represent k processes, and the tokens in p_2 represent some abstract concept of permission. No one can write while someone is reading, and vice versa, but as many people can simultaneously read as they want.

Variants

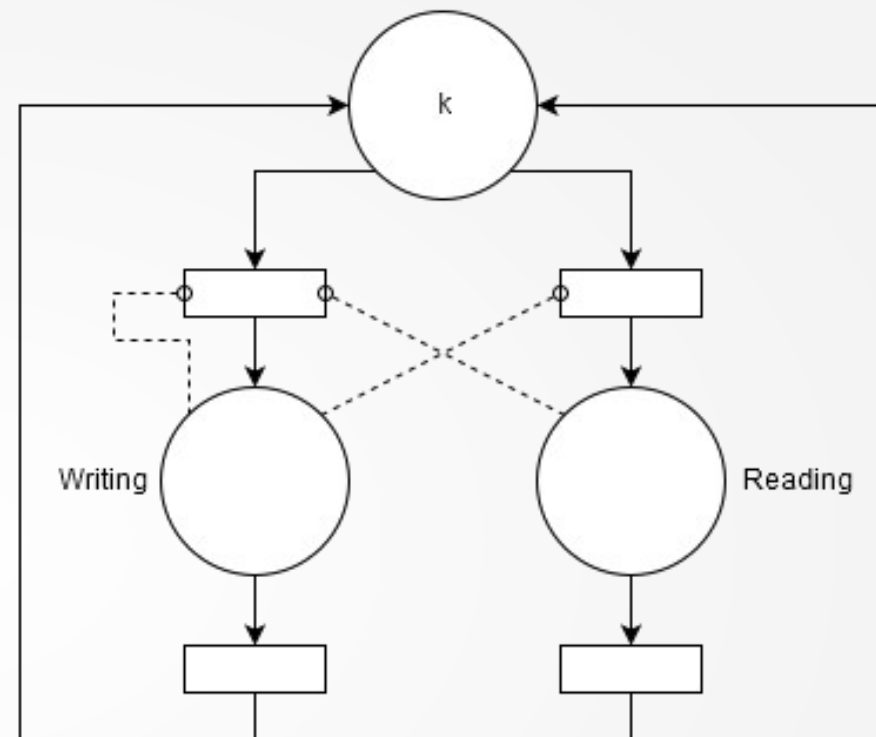
Petri-nets, like many models, can be supplemented with additional functionality in order to increase their modelling power or general utility (through increased ease of use, clearer notation, or merely some interesting mechanic that makes a particular phenomena easier to model).

Following are descriptions of some noteworthy variants of the petri-net.

Extended Petri-net:

The extended petri net differs from a standard petri net in that it has a form of arc called an *inhibitor arc*. An inhibitor arc is drawn from a place p to a transition t and means that t is disabled when p is marked with at least one token. This makes certain phenomena (see right) much easier to model.

In addition, this mechanic makes the extended petri-net equal in modeling power to a Turing Machine.



This models the same read-write synchronization system as before. The dotted lines ending in circles are *inhibitor arcs*. Note how a process can only go to read as long as no one is writing, and how a process can only go to write as long as no other process is reading or writing.

Timed Nets and Stochastic Nets:

Timed and Stochastic Petri-nets are very similar to each other; the basic addition for each one is that every transition has associated with it some delay d such that transition t may not fire until d_t time units have passed since t became enabled. The difference between timed and stochastic nets is that for timed nets the delay is deterministically given, and in stochastic nets it is probabilistically specified.

Timed nets and Stochastic nets are generally used for system performance evaluation, scheduling problems for dynamic systems, and similar time sensitive models.

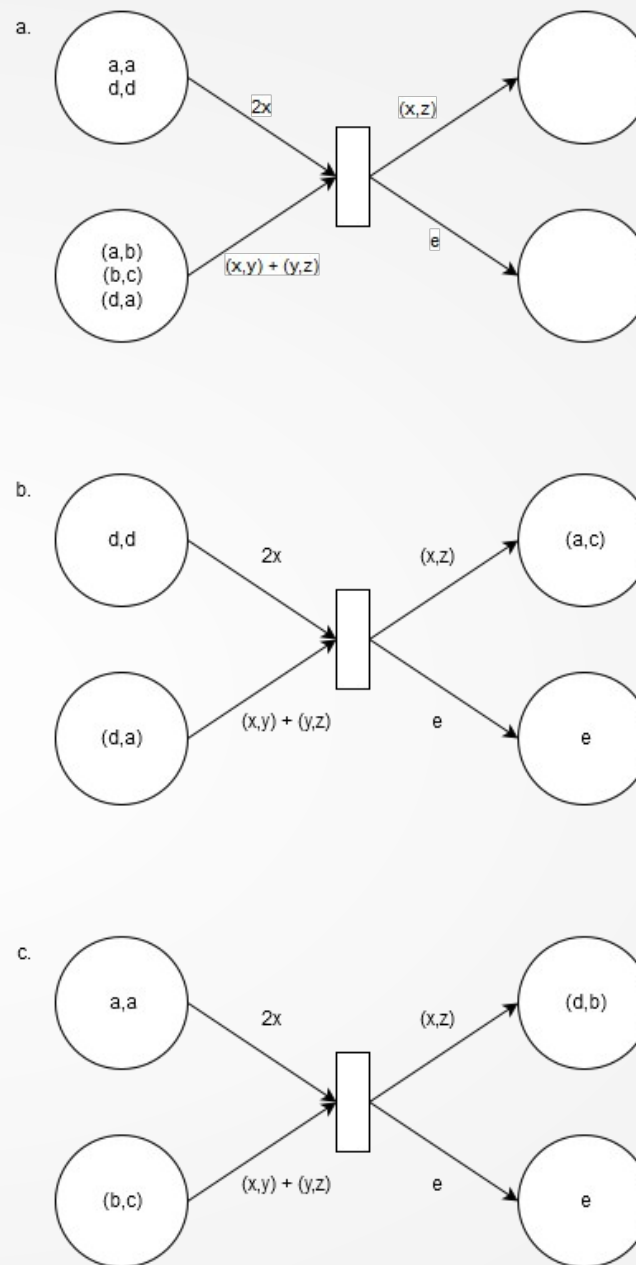
High-Level Nets:

High-level nets can be broadly defined by the firing rule illustrated to the right.

Informally, a high level net incorporates the following qualities:

- Ability for tokens to carry values.
- Ability for arcs to specify predicates.
- Ability for transitions to output different tokens than are input.
- Ability for tuples of tokens to be treated as single tokens.

High-level Petri-nets, given that they allow for predicates, evaluation, and value passing, allow for much easier representation of actual computation than other variants, and are particularly useful for modelling and analysing logic programs.



a. represents a starting state for the net, b. represents one possible marking after the firing of the transition, and c. represents the other.

Minor Variants:

- Color Nets
 - A color net has the same property of a high-level net to assign values to tokens. Each value is called a 'color', hence the name. This alone doesn't add a significant amount of utility to the net.
- Finite Capacity Nets
 - A finite capacity net is one where each place has some maximum capacity of tokens which they can hold. This slightly changes the firing rule, adding a clause that in order to be enabled, in addition to the existing requirements, a transition must also have enough capacity remaining in each of its output places such that after firing none of its output places would be above its capacity. This is useful for more easily representing limited capacity buffers, etc in models; however, this adds no functionality to the model, as a normal petri net can also enforce capacity by producing a p' corresponding to each place to have capacity enforced on it, which starts with a number of tokens equal to p 's capacity – the amount of tokens p starts with. p' is an input place to each transition which has p as an output place, and an output place to each transition which has p as an input place, with equal weights as the arc from p to t in each case. So whenever a transition outputs into t , it also extracts the same amount of 'capacity' from p' .

Questions?

References

·Murata, Tadao. "Petri nets: Properties, analysis and applications." Proceedings of the IEEE 77.4 (1989): 541-580.

·Wikipedia contributors. "Chomsky hierarchy." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 19 Oct. 2016. Web. 19 Oct. 2016.