On the models for asynchronous circuit behaviour with OR causality

A. Yakovlev[a] M. Kishinevsky[b] A. Kondratyev[c] L. Lavagno[d]

[a]Department of Computing Science, University of Newcastle upon Tyne, England
[b]Department of Computer Science, Technical University of Denmark, Denmark
[c]Department of Computer Science, University of Aizu, Japan
[d]Dipartimento di Elettronica, Politecnico di Torino, Italy

Computing Science, University of Newcastle upon Tyne
Technical Report Series No. 463, November 1993

Abstract

Asynchronous circuits behave like concurrent programs implemented in hardware logic. The processes in such circuits are synchronised in accordance with the dynamic logical and causal conditions between switching events. The classical paradigm, easily represented in most process-oriented languages for concurrent systems modelling, is AND causality, which is often associated with a rendez-vous synchronisation. In this paper we investigate a different, less known paradigm, called OR causality. This paradigm is however different from the classical MERGE paradigm, which is based on mutually exclusive events. It has its own subtypes. Petri nets and Change Diagrams provide adequate modelling and circuit synthesis tools for the various OR causality types, yet they do not always bring the specifier to a unique decision about which modelling construct must be used for which type. We present a unified descriptive tool, called Causal Logic Net, which is graphically based on Petri net but has an explicit logic causality annotation for transitions. It is aimed as the least possible generalisation of Petri nets and Change Diagrams. The signal-transition interpretation of this tool is analogous to, but more powerful than, the well-known Signal Transition Graph. A number of examples demonstrate the usefulness of this model in the synthesis of asynchronous control circuits. It is shown that the extension of the basic, unconditional, firing rule with the one that depends upon the marking of the transition preconditions increases the descriptive power of the model to that of Turing Machine and allows the modelling of non-commutative state transition behaviour in a purely causal form.

1 Introduction

Asynchronous circuits can be seen as hardwired versions of concurrent programs. Such a circuit is an interconnection of primitive components, which can either be single-output logical gates or multi-output elements, such as mutual exclusion "gates". The switching events occurring on the circuit inputs and the outputs of the gates when some Boolean conditions in the circuit are not satisfied are those atomic computational actions that characterise the behaviour of any concurrent system, in which processes interact and communicate.

There have recently been many successful attempts of using concurrency models for the verification and synthesis of asynchronous, speed-independent or delay-insensitive circuits[1] [24, 2, 16, 6, 32, 15, 35, 8, 13, 20, 3, 14, 12], to name but a few. The models used in these papers can be broadly classified into three main groups:

- finite-state machine based models,
- process algebraic models,
- event-based or causality models.

The first group [20, 3] essentially builds on the traditional Huffman model of asynchronous circuit, which is closer to the standard synchronous approach used in sequential synthesis. This model assumes the so-called fundamental mode of operation between the circuit and its environment, in which the inputs may not be changed until the circuit elements have been brought into steady state.

[1] Informally, speed-independent circuits are asynchronous circuits whose correct operation does not depend on *output delays* of the constituting gates. Delay-insensitive circuits are asynchronous circuits whose correct operation does not depend on *interconnection delays* among those gates as well. The correct operation of a generic asynchronous circuit, on the other hand, may depend on specific information about gate and wire delays.

The second group [6, 15, 8] makes use of the various compositional and transformational techniques available through the description of a circuit as a collection of communicating processes. Each element stands for a process with its input and output signals being the communication channels or ports. This approach assumes that a set of basic components, generally more complex that the standard set of logic gates, is available during the design procedure. Moreover, it makes certain assumptions on the delays of the interconnections between such modules that require a great deal of care in the final layout phase. Another important shortcoming of this approach is its incapability of explicitly representing causality at the event level (e.g. the ordering relationship between the rising or falling edges of signals).

The third approach [24, 2, 16, 35, 13, 14, 12] is based on the inherently causal framework of Petri nets and avoids the main problems of the first two groups. It has been mainly developed through the Signal Transition Graph model (STG), which interprets Petri net transitions as signal transitions. The capability of modelling causality explicitly is crucial in designing control-dominated circuits such as interfaces, which have been traditionally modelled with timing diagrams. A number of analysis and synthesis techniques have been developed and automated. Such techniques generate hazard-free circuits from STGs under certain restrictions imposed on the structural and behavioural subclasses of STGs, logical element basis and delay models. An important fact has been formally proved in [34] that STGs are sufficient in their modelling power to represent speed-independent (more precisely, semi-modular as defined in [18]) circuit behaviour. As was shown in [11, 33, 34, 7] speed-independent behaviour is characterised by two major forms of causality between signal transitions, strong (AND) and weak (OR) causality. Let us assume for example, that event a has two cause events b and c. The strong form of causality assumes that both events b and c must have occurred before the given event a may occur. Therefore, in the "strong" case, every cause strongly precedes its effect. In the case of weak causality the event a may occur after any of the events b or c have occurred. I.e. in the "weak" case an event in question may be caused by any cause belonging to the set of weak causes for the event, provided that at least one such cause has occurred.

There are however certain limitations inherent in STGs that prevent them from efficient modelling of weak causality in circuits. The problem is thus with the underlying Petri nets, whose event dynamics is "biased" towards the strong form of causality (a Petri net transition fires only if *all* its input places contain tokens). Weak causality can only be represented in Petri nets *indirectly*, by using complicated place/transition interconnecting schemes (see, e.g., Figure 4). For this, the Petri net *must* be made unsafe (allow more than one token in a place). Most of the existing STG-based methods and tools require the underlying Petri net be safe. The non-safety should generally be no problem per se, but the problem arises because there is no precise way of saying how the second token arriving in the place that models the OR causality has to be "removed" before the place can be remarked again.

Such problems appear to be resolved in a slightly different model, called Change Diagram [33, 13, 12]. The Change Diagram, whose original target was circuit behaviour modelling, gives "equal priority" to the two major causality types because events have two mechanisms of enabling. To avoid the need for unsafeness when representing OR causality, Change Diagrams have a "token borrow" mechanism, which allows a place [2] to be marked with a *negative* value. Note that Change Diagrams can still be unsafe if this is required for other modeling purposes than representation of OR causality *per se*. It has been shown [34] that certain behaviours that are representable with finite but unbounded (i.e., with places with unlimited marking) Change Diagrams cannot be modelled using a finite STG representation.

Unfortunately, Change Diagrams have a number of shortcomings, one of which is the other side of the negative marking "medal". It is possible to model some conflict-free (corresponding to semi-modularity in circuits) behaviour with a finite and bounded Petri net, while the corresponding Change Diagram would be negatively unbounded [34]. Furthermore, Change Diagrams, in their present form cannot directly represent processes with conflicts or choice. This appears to be a serious limitation because the designer has to apply some external compositional mechanisms in order to model such behaviours, which are commonplace in multi-mode interface protocols (e.g., memory Read and Write operations).

In this paper we tackle the discrepancy between these otherwise closely related models. We first present a series of examples which show the importance of being able to model both types of causality in asynchronous hardware. Secondly, we demonstrate the ways in which both STGs and Change Diagrams model the weak form of causality, OR causality. With the aid of examples, we also show the problems of each of the two languages in modelling some "difficult" cases. Thirdly, we present a way for unifying the capabilities of both models into one model, called Causal Logic Net, that is based on a Petri net graphical notation augmented with the causal logic attribute for transitions. The latter is in the form of Boolean enabling functions for net transitions. The Causal Logic Net model is thus viewed as the main theoretical contribution of this paper.

We present an example of synthesizing circuits from the Causal Logic Nets, from which we elicit two major types of OR causality. This part of our study results in a crucial practical result, which is the precise recommendation for the specifier as to which modelling construct of the Causal Logic Net has to be used to represent a particular type of OR causality. Formerly [37], such a recommendation was not clear enough.

The final issue addressed in this paper is the extension of the basic Causal Logic Net model to the level of descriptive power equal to that of Inhibitor nets [21]. This requires defining another transition firing mechanism, more general that the one used in ordinary Petri nets. An important consequnce of this generalisation is that, in combination with OR causality, it allows the modelling of a non-standard form of nondeterminism in circuit behaviour. This form is called non-commutativity.

[2] To be more precise, an arc, since a Change Diagram is a directed graph in which marking is attributed to the arcs between vertices.

We demonstrate how the behaviour of a Fair Arbiter, originally presented in the form of a non-commutative state transition diagram in [34], can be described by means of an extended Causal Logic Net, which captures concurrency semantics in its explict form.

2 OR causality: Background and Motivation

2.1 Real life cases

The following two examples show that we often deal with the weak form of causality between events without paying too much attention to the way we should model it.

Cautious client. Consider the following situation that often occurs when we are, as clients or customers, looking for better service from some businesses, such as travel agencies or building companies. Imagine we have a job that cannot be done without the help of a professional, who knows this job better. We take a Yellow pages volume and select a number of firms who are supposed to be eager to do this job for us. Now, imagine that we are a bit overly cautious and we would not like to rely on just one agent but select at least two of them, whom we give the specification of the job. We set them to do the job at the same time, and then when the job is finished by at least one of them, we proceed further without caring much about the other agent, who may have not yet finished the job [3]. Since the job has been the same, we do not really mind whose results we shall be using afterwards. Perhaps, later we realise that the other agent also has completed the job. So we may take the results of the both agents and compare them to see that neither of them has cheated (this is the minimum redundancy and hence minimum error assurance unless we are able to pay for more agents involved). Such a comparison would be the advantage that we take from the initially redundant operation scheme.

Now, leaving alone the question of the benefits and overheads we have had from using the redundancy technique (which we *must* have), let us look at this situation only from the viewpoint of the order of actions involved and their temporal relationship.

As it has been described, the actions A_1 and A_2, standing for the operations performed by the corresponding agents, are executed in parallel, independently of each other. Then, as soon as either of them has been complete, we start the third action C_1, which denotes our further activity as a client. At some point later, in order to do something else (for example, ascertain the correctness of the results or activate the same two agents again), we need to check if both agents have finished their assignment. Thus we may have the fourth action C_2 which happens after both A_1 and A_2 have finished.

In the above situation, we say that action C_1 is weakly (OR) caused by actions A_1 and A_2, as opposed to action C_2 that is strongly (AND) caused by A_1 and A_2.

Scheduling n tasks on m ($m < n$) resources. Imagine a group of n independent tasks (say, in an operating system kernel) which *may* be executed concurrently because they carry no mutual dependency on the results of each other. Unfortunately, the number of resources available for their parallel execution, denoted as m is such that $m < n$. For simplicity, let $n = 3$ and $m = 2$.

Since there is no way to run the three tasks in parallel, as for example would have been possible had we one extra resource, the natural way for achieving maximum performance would be as follows. First, we run two of them on the pair of available resources (let us denote these two actions A and B) and then, upon the completion of the fastest of them, we allocate the released resource to the remaining task (start action C).

It is clear that actions A and B cause action C in a weakly causal manner.

The described situation is somewhat intermediate in performance between the case when $m = n = 3$ and the case in which the ordering is based on strong causality (i.e., C begins after both A and B have been completed). The first situation is impossible because in our case we have $m = 2$, while the second case is obviously not required in this case.

The above performance relationship is easily proved by simple timing analysis. Let t_A, t_B and t_C denote the duration of the corresponding tasks. Then the full execution times for the compared three strategies would be as follows:

$$T_{\|} = max(t_A, t_B, t_C), T_{OR} = max((min(t_A, t_B) + t_C), t_A, t_B), T_{AND} = max(t_A, t_B) + t_C.$$

Obviously, $T_{\|} \leq T_{OR} \leq T_{AND}$. Thus if the number of resources is limited, one should resort to a schedule with weak (OR) causality, which is more advantageous than the one based on partial order with just strong (AND) causality.

Both these very simple, yet vivid, examples demonstrate that there should be an adequate way of modelling this form of dynamic organisation of systems, even without considering its specific character in asynchronous circuits.

2.2 OR causality in circuit design

Let us now look at the examples where OR causality allows better functionality in hardware structures.

[3] Unfortunately, the real world does not offer us this luxury free of charge and we must pay for both jobs at the outlet anyway.

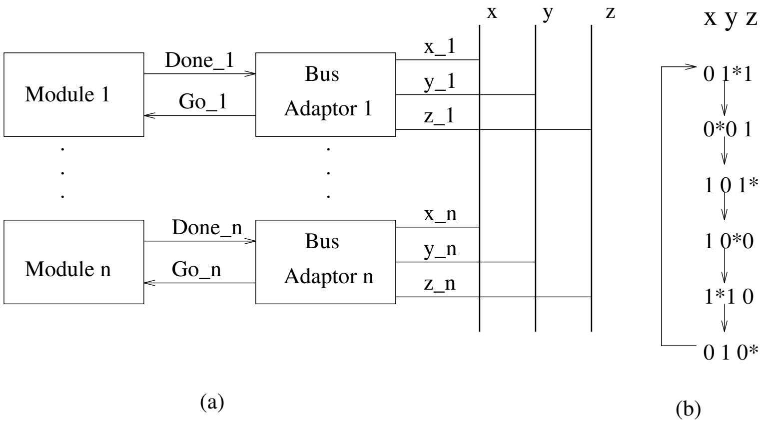


Figure 1: Synchronisation with wired-OR logic: a system structure (a) and a communication protocol on the wires x, y and z (b).

2.2.1 Wired OR synchronisation

One of the classic problems in asynchronous hardware is that of organising a group of modules in such a way that they execute a sequence of operations *synchronously* but without using a common clock. A standard way here can be the use of a multi-output C-element, sometimes called also Muller C-element due to its inventor, D.E. Muller (whose pioneering work was surveyed in [17]). The behaviour of a two-input C element is defined by the following logical equation:

$$Y = x_1 x_2 + (x_1 + x_2)y,$$

where x_1 and x_2 are input signals and Y is the output signal, which is fed back and given the name of y. In both transition phases (0-1 and 1-0), the value of Y is changed last with respect to the changes of x_1 and x_2. For example if in the state where all three signals are initially set to 0, both x_1 and x_2 concurrently switch to 1, the output Y begins to switch to 1 if and only if $x_1 = x_2 = 1$. Due to this effect, a C-element is also called an event-based AND gate [29, 23]. The form of causality used in it is the strong one.

Let us now assume that we need to provide synchronisation for a group of modules, whose number is so large that we cannot interconnect them to a central multi-input C-element. Nor we are allowed to construct a distributed chain of two-input C-elements, one for each module, since this serial interconnection would be prohibitively slow in action.

An alternative idea to synchronise these modules in a distributed way has been used for example in [28]. It is based on a bus interconnection and the so called *wired-OR logic*. All modules are interconnected to a fixed number of wires (independently of the number of modules synchronised) and performing the synchronisation in a hazard-free way with the speed that is again approximately independent of the number of modules, as shown in Figure 1. It was shown in [28] that the least possible number of wires required for such a synchronisation is three and the signal transitions on these wires are cyclically shifted through these three wires, denoted say as x, y and z, executing at each cycle both the causal AND and OR synchronisation actions.

The reason for both types of causality is simple. It is due to the operation of a bus wire with respect to the local ("input") wires. A bus wire, say x, when its signal state is to be changed from "high" to "low", becomes asserted to "low" immediately after the *fastest* of its "input" wires becomes "low" – here, it performs OR causality. In the opposite case, for the change from "low"' to "high", the bus wire x waits until the *slowest* of the "input" wires has been released to "high". Only then it can be released to "high" itself – the situation of AND causality. The overall operation on the three identical wires requires that AND and OR causality actions alternate and are performed on the adjacent bus wires. Namely, if the initial state for the order xyz is 011, first, the OR action is performed on y, then the AND action on x, then OR on z, then AND on y etc.

In this behaviour the use of OR causality is caused by the properties of a single bus wire that behaves like a logical AND gate. In terms of events such a gate exhibits strong causality in one phase and weak causality in the other.

2.2.2 Hardware structures with redundancy

An example, analogous to our "cautious client" example, can be illustrated in hardware structures for fault-tolerance, where a certain level of redundancy helps to ensure (at least by detecting the presence of an error) that the computations are performed correctly. Let us have a group (for simplicity, consider two) of *functionally equivalent* modules that are possibly implemented in a different way and hence work with different speeds. Assume that these modules, for the same tasks, present their results on a common bus, which we have called the Merging Bus as shown in Figure 2.

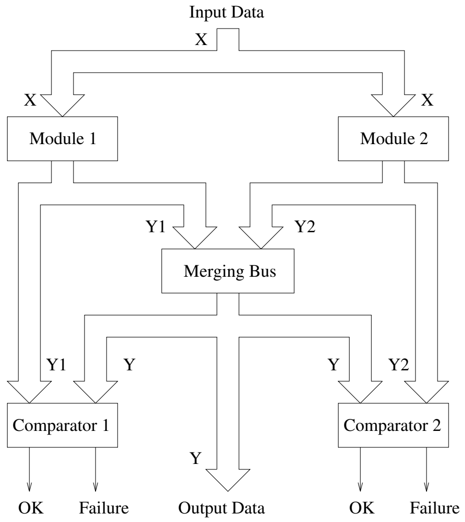


Figure 2: Hardware structure with redundancy

The modules operate in the manner where the computation result is taken from the first-to-compute module and placed on the Merging Bus in an OR-causal way. The subsequent completion of the task in the other module must be awaited in an AND-causal way only to check the successful comparison of the result of each module with the result established on the Merging bus, with the possibility of a *backward recovery* action in the case of a *mismatch*. If there is no mismatch, the normal computation flow can be already well ahead of the current point of the result checking, thus making the whole fault-tolerance mechanism operate generally *faster* than in conventional schemes, in which a *voting element* usually waits for the completion of the tasks in all the modules before producing the final result for the next computation stage.

2.2.3 Low latency arbitration

We have recently developed an arbiter [36] that makes use of the idea that was originally proposed in [38] and implemented in [9]. The idea to use OR causality for faster arbitration emerged in [38] in the context of the process algebraic approach. On the other hand, the circuit in [36] improves on the circuit in [9] by exploiting the explicit causality modelling achievable through the use of STGs, thus motivating the introduction of *explicit OR causality* in our model.

An asynchronous arbiter is a device that dynamically allocates a single shared resource to the user modules in a system without a common clock. Each user, when it needs the resource, issues an asynchronous request and waits until the arbiter produces a grant. The user then uses the resource and after finishing its action releases its request. This results in a subsequent release of the grant, after which the user can issue another request and so on. The arbiter, when it receives a number of active requests from different users, generates after some delay a grant to exactly one of them and leaves other requests pending until the granted user has released the request. The arbiter then releases the grant and, if there are pending requests, produces another active grant, again on a mutually exclusive basis.

Let us consider a typical cell of a multi-way arbiter that arbitrates between two users [26]. Multi-way arbitration is organised by cascading such cells to form a tree or a chain. Each cell propagates the request in the direction from the lower level to the upper level, while the grants are generated in the opposite direction. Figure 3.(a) shows one such cell with its three request-grant handshake links $(R1, G1)$, $(R2, G2)$ and (R, G), where $(R1, G1)$, $(R2, G2)$ stand for the links with lower levels, producing competing requests $R1$ and $R2$, and the (R, G) pair is the link with the upper level. Figure 3.(b) illustrates, with the help of a timing diagram, the handshaking protocol between the links. After the first request by $R1$ is granted and the resource is released, two simultaneous requests are made by $R1$ and $R2$ and granted in turn.

Asynchronous arbiters of this type are usually implemented using an SR flip-flop and an analogue mutual exclusion element that is aimed at resolving the metastability and oscillation anomalies occurring in the flip-flop. The time it takes to resolve the arbitration can be much longer than an ordinary switching delay [25]. The implementation of the arbiter described in [9] is advantageous over the one in [26] for it allows *not to wait* for the arbitration to be resolved by the local mutual exclusion element before propagating the request (signal R) to the upper level. The circuit presented in [9] uses a weak form of causality to produce the request on R from the arrival of $R1$ or $R2$ (the first of them causes R to be set), thereby allowing the arbitration resolution process to be executed in parallel with the process run in the upper level to generates the grant on G. Normally, of course, when the signal on G arrives the cell is ready to generate an appropriate grant either on $G1$ or $G2$ depending on which of the grants has been chosen by the mutual exclusion element.

The above example shows that using OR causality seems quite a natural way of organising the dynamic behaviour in

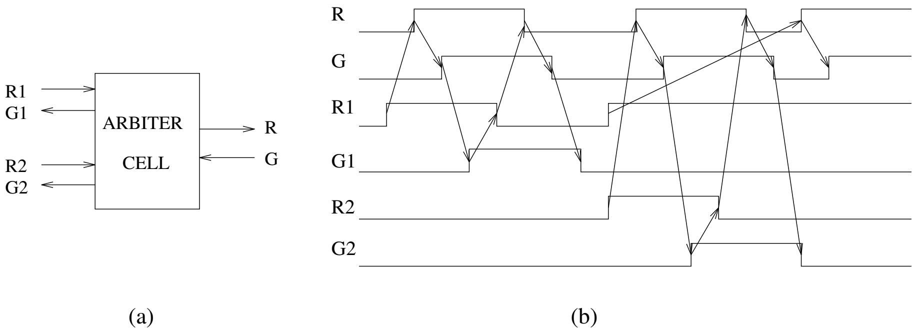


Figure 3: The arbiter example

circuits. Yet, the descriptive tools of even such event-oriented models as Petri nets are not very well suited for the modelling of this form of causality in a universal way. The following section more formally proves the need for the modelling enhancement of Petri nets.

3 Modelling OR Causality in Signal Transition Graphs and Change Diagrams

3.1 Petri Nets and Change Diagrams

When talking about Signal Transition Graphs and their ability to model the forms of causality, in this paper we often address the properties of their underlying model, Petri nets. On the other hand, there has been no separate, uninterpreted, notation defined for Change Diagrams in the literature ([13, 12]), so we shall use the same name to refer to the more abstract unlabelled version of such a model.

The following table summarises the unified terminology that we will follow in the remainder of the paper.

PN/STG	**CD**	**this paper**
transition/event	event	transition/event
signal transition	change	signal transition
marking	activity	marking

3.1.1 Petri nets

Petri nets [21, 19] are a widely used model for concurrent systems, because they have a very simple and intuitive semantics, that directly captures concepts like causality, concurrency and conflict between events.

A *Petri net* (PN) is a triple $\mathcal{P} = \langle T, P, F \rangle$ where:

- T is a non-empty finite set of transitions,
- P is a non-empty finite set of places, and
- $F \subseteq (T \times P) \cup (P \times T)$ is the flow relation between transitions and places.

A PN can be represented as a directed bipartite graph, where the arcs represent elements of the flow relation.

A PN marking is a function $m : P \to \{0, 1, 2, \ldots\}$, where $m(p)$ is called the number of *tokens* in p under marking m. A *marked* PN is a quadruple $\mathcal{P} = \langle T, P, F, m_0 \rangle$, where m_0 denotes its initial marking. A transition $t \in T$ is *enabled* at a marking m if all its predecessor places are marked. An enabled transition t *may fire*, producing a new marking m' with one less token in each predecessor place and one more in each successor place (denoted by $m[t > m'$).

A sequence of transitions and intermediate markings $m[t_1 > m_1[t_2 > \ldots m'$ is called a *firing sequence from* m. The set of markings m' reachable from a marking m through a firing sequence is denoted by $[m >$. The set $[m_0 >$ is called the *reachability set* of a marked PN with initial marking m_0, and a marking $m \in [m_0 >$ is called a reachable marking.

A PN marking m is *live* if for each $m' \in [m >$ for each transition t there exists a marking $m'' \in [m' >$ that enables t. Similarly, a transition t is *live* if for each $m' \in [m >$ there exists a marking $m'' \in [m' >$ that enables t. A marked PN is live if its initial marking is live.

A marked PN is *k-bounded* (or simply "bounded") if there exists an integer k such that for each place p, for each reachable marking m we have $m(p) \leq k$. A marked PN is *safe* if it is 1-bounded.

A transition t_1 *disables* another transition t_2 at a marking m if both t_1 and t_2 are enabled at m and t_2 is not enabled at m' where $m[t_1 > m'$. A marked PN is *persistent* if no transition can ever be disabled at any reachable marking.

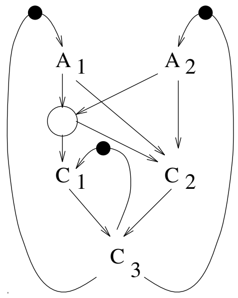


Figure 4: Petri net model for the "cautious client" example

A PN is a Marked Graph if every place has exactly one predecessor and one successor. A PN is *free-choice* if for any two transitions t_1 and t_2 that share a predecessor place t_1 and t_2 have only one predecessor. (i.e. any two transitions with a common predecessor place have only one predecessor).

A marked PN $\mathcal{P} = \langle T, P, F, m_0 \rangle$ generates a state graph, called Reachability Graph, $\langle [m_0 >, E, T, \delta \rangle$, where for each edge $(m_1, m_2) \in E$, such that $m_1, m_2 \in [m_0 >$ and $m_1[t > m_2$, we have $\delta(m_1, m_2) = t$.

Earlier work on Signal Transition Graphs [2, 16, 31] required the underlying PN to be live, safe and free-choice. These requirements severely restricted the class of modelled behaviours. On the other hand, [33, 37] showed that in order to model OR causality in PNs as a *relationship between PN transitions* (rather than through their possible labels, thus introducing an extra level of interpretation), we must extend the class of nets to *unsafe non-free-choice* ones. Furthermore, such an extension is required even if we model behaviour without conflicts (alternatives or choice). On the other hand, conflict-free behaviour with only AND causality can be modelled with Marked Graph nets ([5]).

Figure 4 shows a labeled PN model of the system described in our "cautious client" example. Note that PNs used for the representation of Signal Transition Graphs have traditionally been depicted in a "shorthand" form, which seems convenient to the circuit designer and which is adopted in this paper unless it creates confusion. In this form, PN transitions are denoted by their corresponding labels (instead of bars or boxes) and PN places are explicitly denoted by circles only if such place has more than one predecessor or successor transitions. If a place has only one predecessor and one successor, the corresponding circle is omitted and the token marking is associated with the arc (a similar notation is often used to represent Marked Graphs [5]). The names of transitions correspond to the actions. C_1 is the action of the client weakly caused by either A_1 or A_2, whichever occurs first. C_2 strongly caused by both A_1 and A_2. An additional transition, C_3, models the situation in which the client may reuse the agents for further assignments. Note that due to the inherent AND causality of C_2, the client never issues a new assignment to the agent before the previous one has been accomplished.

3.1.2 Signal Transition Graphs

Interpreted Petri nets, where transitions represent changes in the values of circuit signals, were proposed independently as specification models for Asynchronous Logic Circuits by [24] (where they were called Signal Graphs) and [2] (where they were called Signal Transition Graphs, STGs). Both papers proposed to interpret a PN as the specification of a circuit defined on a set of signals Y, by labelling each transition with an element of $Y \times \{+, -\}$. A label y_i^+ means that signal $y_i \in Y$ changes from 0 to 1, and y_i^- means that y_i changes from 1 to 0, while y_i^* denotes either y_i^+ or y_i^-.

An STG is a quadruple $\mathcal{G} = \langle \mathcal{P}, X, Z, \Delta \rangle$ where $\mathcal{P} = \langle T, P, F, m_0 \rangle$ is a *marked* PN, X and Z are (disjoint) sets of input and output signals respectively ($Y = X \cup Z$), and $\Delta : T \rightarrow (X \cup Z) \times \{+, -\}$ labels each transition of $\mathcal{P}$ with a signal transition. An STG is *autonomous* if it has no input signals (i.e. $X = \emptyset$).

Both [24] and [2] gave also synthesis methods to translate the PN into a State Transition Diagram (called Transition Diagram in [24] and State Graph in [2]) and hence into a circuit implementation of the specified behavior.

Given an STG $\mathcal{G} = \langle \mathcal{P}, X, Z, \Delta \rangle$ and the Reachability Graph $([m_0 >, E, T, \delta)$ corresponding to its PN $\mathcal{P}$ (where δ labels each edge in E with a transition in T), we define the associated State Transition Diagram (STD) $\mathcal{S} = \langle [m_0 >, E, \lambda \rangle$ as follows. For each state (marking) $m \in [m_0 >$, $\lambda(m)$ is a vector of signal values (also denoted s^m for simplicity). We say that vector s^m is a *state label* corresponding to the marking m. Obviously the STD labeling must be *consistent* with the interpretation of the STG signal transitions. Let s_i^m denote the value of signal y_i in s^m. Each arc $e = (m, m') \in E$ in the STD must obey the following *consistency condition*:

- if $\Delta(\delta(e)) = y_i^+$, then $s_i^m = 0$ and $s_i^{m'} = 1$.
- if $\Delta(\delta(e)) = y_i^-$, then $s_i^m = 1$ and $s_i^{m'} = 0$.
- otherwise $s_i^m = s_i^{m'}$.

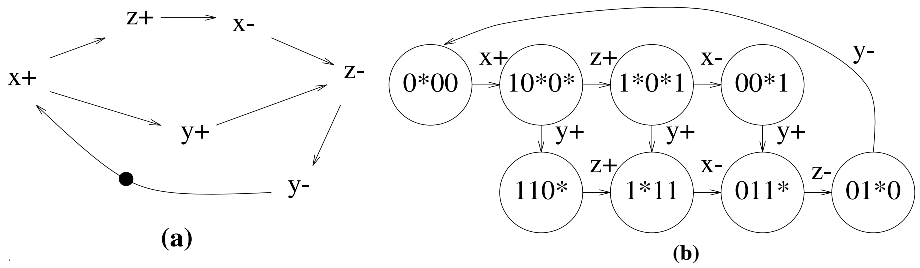


Figure 5: A Signal Transition Graph and its State Transition Diagram

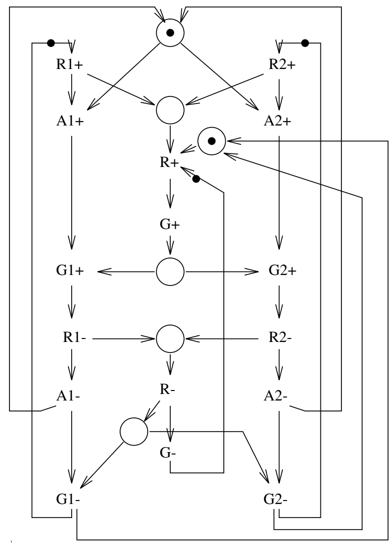


Figure 6: Signal Transition Graph for the arbiter in Figure 3

A *finite, bounded* STG is defined as *valid* if its underlying reachability graph has a *consistent labeling* as defined above.

Figure 5 shows an example of an STG and the corresponding STD. Following the convention of [18], a signal value is denoted by $1*$ in a label if it is currently 1 and a falling transition for it is enabled in the corresponding state. Similarly $0*$ denotes a signal that is currently at 0 but enabled to rise. 0 and 1 denote stable values for the corresponding signal. The initial marking of the PN in Figure 5.(a) (corresponding to the leftmost state in Figure 5.(b)) appears on the edge between y^- and x^+.

Another example of STG is shown in Figure 6. This is a model of the low latency arbiter described in Section 2.2.3[4]. The STG clearly demonstrates that after the arrival of either $R1$ or $R2$, the handshake R/G is set in parallel with resolving the mutual exclusion inside the arbiter cell (by means of an internal mutual exclusion element whose output signals are denoted as $A1$ and $A2$ in Figure 6).

3.1.3 Change Diagrams

Change Diagrams, described more in detail in [33, 13, 12], are an event-based model for Asynchronous Logic Circuits that bears some resemblance to Signal Transition Graphs, but has some interesting properties of its own.

The definition of Change Diagrams is based on two types of precedence relations between transitions in Asynchronous Logic Circuits.

1. the *strong precedence* relation between transitions a^* and b^*, usually depicted by a solid arc in the graphical representation of Change Diagrams, means that that b^* *cannot occur without the occurrence of* a^*.

2. the *weak precedence* relation between transitions a^* and b^*, usually depicted by a dashed arc in the graphical representation, means that b^* *may occur after an occurrence of* a^*. But b^* may also occur *after some other transition* c^*, which is also weakly preceding b^*, without the need for a^* to occur.

[4] The figure is taken from [36], in which due to the further refinement of causality between the signal transition actions we have been able to improve on both [26] and [9].

A Change Diagram (CD) is therefore formally defined as a tuple $\mathcal{D} = \langle A, \rightarrow, \vdash, M, O\rangle$, where:

- A is a set of *transitions* or *events*.
- $\rightarrow \subseteq (A \times A)$ is the *strong* precedence relation between transitions.
- $\vdash \subseteq (A \times A)$ is the *weak* precedence relation.
- M is a set of initially *marked* arcs.
- O is a set of so-called *disengageable* arcs.

For the purpose of circuit specification all transitions from A are labelled with signal transitions of a set of signals Y similarly to signal transition labelling of STG. The relations $\rightarrow$ and $\vdash$ are mutually exclusive (i.e. $(a^*, b^*) \in \rightarrow$ implies that $(a^*, b^*) \notin \vdash$ and vice-versa), and all the predecessors of a transition a^* must be either of the *strong* type or of the *weak* type. Hence the set of transitions A is partitioned into *AND-type* transitions (with strong predecessors) and *OR-type* transitions (with weak predecessors).

The firing rule of CDs is similar to that of PNs, with arcs playing the role of places and flow relation elements at the same time. Each arc is assigned an integer *marking* which, unlike PN marking, can be *negative*. Initially each arc in M has marking 1, and each arc not in M has marking 0.

- An *AND-type* transition is enabled if *all* its predecessor arcs have marking greater than 0.
- An *OR-type* transition is enabled if *at least one* predecessor arc has marking greater than 0.

When an enabled transition fires, the marking of each predecessor arc is decremented, and the marking of each successor arc is incremented[5] A CD is *bounded* if the marking on each arc is bounded (both above and below) in all possible firing sequences.

Disengageable arcs are "removed" from the CD after the *first firing* of their successor transition. They are used to represent the *initialization sequence* of a circuit, and we will not enter into details concerning their usage.

Following [33], a State Transition Diagram $\mathcal{S} = \langle S, E, \lambda\rangle$ can be associated with a CD, as we did above for STGs. Let S be the set of reachable marking vectors. An arc $(s, s') \in E$ joins two marking vectors $s, s' \in S$ if there exists a transition $y_i^* \in A$ that is enabled in s and whose firing produces s'. The labelling must be consistent, so for each arc $(s, s') \in E$ corresponding to transition y_i^* we must have:

- $\lambda(s)_i = 0$ and $\lambda(s')_i = 1$ for an arc associated with y_i^+.
- $\lambda(s)_i = 1$ and $\lambda(s')_i = 0$ for an arc associated with y_i^-.
- otherwise $\lambda(s)_i = \lambda(s')_i$.

A CD is *correct* if it satisfies the following conditions, ensuring that the above labelling is consistent:

- for all firing sequences, the signs of the transitions of each signal alternate.
- no two transitions of the same signal can be concurrently enabled in any reachable marking vector.
- the CD is connected and *bounded* (i.e. the set S is *finite*).

The main theoretical result concerning CD correspondence to semi-modular STD claims that

1. each semi-modular STD without transient cycles has a corresponding correct CD, and
2. each correct CD has a corresponding semi-modular STD [12].

A *transient cycle* in an STD is defined as a cycle where at least one variable is continuously excited with the same value. See, for example, the cycle of states labeled: $0*0*0.000 \rightarrow 0*10.0*00 \rightarrow 0*1*0.100 \rightarrow 0*00*.100 \rightarrow 0*01.10*0 \rightarrow 0*01*.110 \rightarrow 0*00.110* \rightarrow 0*00.1*1*1 \rightarrow 0*00.01*1 \rightarrow 0*00.001* \rightarrow 0*0*0.000$ in Figure 14. In all states of this cycle signal b has value $0*$.

CDs are useful in practice because of the availability of low-complexity polynomial time *analysis* algorithms to decide, e.g.:

- whether a given CD is correct, and hence it can be used as a valid specification of a semi-modular circuit.

[5] The marking of a predecessor arc $a^* \vdash b^*$ of an *OR-type* transition b^* can become negative as a consequence of a firing of b^* due to positive marking on some other arc $c^* \vdash b^*$. It can then return to 0 when a^* fires in turn. After the next a^* firing the marking of arc $a^* \vdash b^*$ can become zero or positive again.

- whether a given circuit has a distributive CD, and hence a distributive STD. Note that this analysis can be performed by direct construction of the CD, without going through the exponential size STD ([12]).

Furthermore synthesis algorithms from CDs to circuits in various technologies are outlined in [12].

The main limitation of CDs however is their inability to describe *choice* among alternative behaviors, as modeled by places with more than one successor in PNs. So a designer faced with the description, for example, of a self-timed memory element, must describe the various possible read/write cycles of each data value as an *alternation* rather than a *choice* between them.

3.2 Problems in modelling OR causality

Although both Petri nets and Change Diagrams are capable of modelling the causality paradigms of semi-modular circuits, there are some problems, and even discrepancy, in the way these formalisms represent OR causality. Here we shall briefly revisit the two examples of conflict-free behaviour from [34]. But let us first look at the problems in modelling OR causality by Petri nets from a formal side.

3.2.1 Problems with using Petri nets

To show formally what are the difficulties of modelling OR causality by PNs let us define the notion of *observation equivalence* between PNs and CDs. We will think of such an equivalence as the similarity of the partial behaviours generated by a PN and a CD for a given subset of their transitions. In other words, if we establish a one-to-one correspondence between a selected set of transitions in a PN and a CD, the observable behaviour of the PN and CD, seen only from the viewpoint of the selected transitions, must be exactly the same (up to renaming the selected transitions). For transitions that are not in the selected subset the behaviour of both the CD and the PN can be different. I.e., if we delete (hide) from a firing sequence transitions not in the selected sets, then the sets of firing sequences of the PN and the CD must be identical. This approach allows us to have some freedom in modelling a PN using a CD and vice-versa.

The notion of behaviour in both PNs (CDs) can be defined in terms of the accepted languages. Suppose that transitions of a CD $\mathcal{D}$ and of a PN $\mathcal{P}$ are labelled by the elements from some set B. These labels are not necessarily unique, i.e. a number of transitions can have the same label. More formally, two partial labelling functions are defined, one on the set of PN transitions T and another on the set of CD transitions A (i.e., $T \rightarrow B$ and $A \rightarrow B$). For STGs and CDs used in circuit specification these labelling functions assign signal transition names, as described in Sections 3.1.2 and 3.1.3 but the definition is not limited to this application.

Given a labelled CD $\mathcal{D}$ or a labelled PN $\mathcal{P}$ the *language* accepted by $\mathcal{D}$ or $\mathcal{P}$, denoted as $L(\mathcal{D})$ or $L(\mathcal{P})$, is a set of *feasible* sequences of transition labels [6] that the model generates from its initial marking. Return to the example of the "cautious client" in Figure 4. Sequence A_1, C_1, A_2, C_2, C_3 is feasible and belongs to the language accepted by this PN, while sequence A_1, C_1, C_2 is not feasible.

Denote by $\mathcal{Q}$ a CD or a PN under consideration. We now define the *projection* operator, denoted by $\downarrow$ for the labelled CDs and PNs with respect to a subset of the labels $B_1 \subset B$ as follows.

1. If p is a feasible sequence for $\mathcal{Q}$, then its projection $p \downarrow B_1$ is sequence p with all labels from $B - B_1$ deleted.

2. If $L(\mathcal{Q})$ is a language accepted by $\mathcal{Q}$, then its projection $L(\mathcal{Q}) \downarrow B_1$ is a set of sequences $\{p \downarrow B_1 \ : \ p \in L(\mathcal{Q})\}$

Two models, a CD $\mathcal{D}$ and a PN $\mathcal{P}$, are *observation equivalent* with respect to the subset of labels B_1 if $L(\mathcal{D}) \downarrow B_1 = L(\mathcal{P}) \downarrow B_1$

The introduced equivalence relation is not very restrictive, because it compares behaviour only up to the "observable points" of specifications. Moreover, we have no restrictions on the corresponding number of transitions labelled by the same symbol in the models. Therefore one can model behaviour of some particular labelled transition from the CD by a set of transitions from the PN labelled by the same symbol, and vice versa.

Let us introduce a notion of *observation isomorphism* to disallow several PN transitions to be associated with one CD event and vice versa. This notion is useful due to the following two reasons. First, to save compactness and locality of the specification and, second, which is more important, to represent an OR-causal event directly, as a single transition in the PN model, rather than through its "simulation" by the firing of several transitions, each occurring after its own OR-cause.

Two models, a CD $\mathcal{D}$ and a PN $\mathcal{P}$, are *observation isomorphic* with respect to the subset of labels B_1 if they are observation equivalent with respect to B_1 and each $b \in B_1$ labels the same number of transitions both in $\mathcal{D}$ and in $\mathcal{P}$.

Figure 7.(a) shows a simple CD $\mathcal{D}$ with an OR-event c, while Figures7.(b) and (c) show two different PNs $\mathcal{P}1$ and $\mathcal{P}2$ that are equivalent to $\mathcal{D}$ with respect to transitions $\{a, b, c\}$ [7].

[6] Feasible sequences of transitions are also called firing sequences in Petri net terminology.
[7] In case of unique labels, we do not distinguish between a transition and its label.

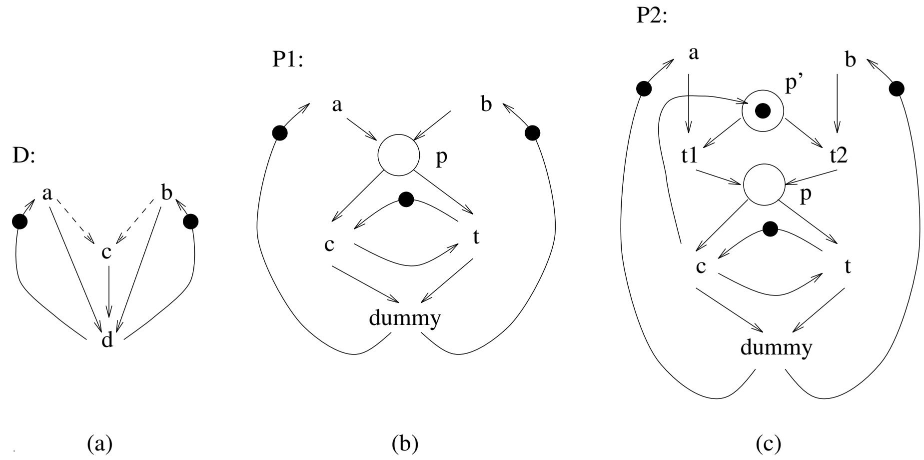


Figure 7: A Change Diagram with equivalent Petri Nets

In $\mathcal{P}1$ the transition c will fire after a or b have put a token into the place p. If both a and b have fired, two tokens will be placed in p. To prevent c from repeatedly firing in the latter case, we introduce the additional "hidden" transition t to remove the second token from p. Thus, in the functioning of $\mathcal{P}1$ the firings of c and t alternate; c fires from the "first' token in p and t from the "second". $\mathcal{P}2$ operates in a similar way, with the exception that in $\mathcal{P}2$ the second token is not allowed to pass into p, but is "delayed" on transitions $t1$ and $t2$ by the choice place p'.

It is easy to see that $\mathcal{P}1$ is unsafe (for place p) and $\mathcal{P}2$ is non-persistent (for place p'), moreover both PNs are non-free-choice (e.g., for places p). This is not just a shortcoming of this particular example since it is intrinsic to the OR-causality modelling in PNs as the following Proposition shows.

Proposition 3.1 *If a safe CD $\mathcal{D}$ is observation isomorphic to a PN $\mathcal{P}$ with respect to the set of live* [8] *transitions $\{a, b, c\}$, where $a \vdash c$, $b \vdash c$, then the PN P is either* non-persistent and non-free-choice *or* unsafe and non-free-choice.

The proof of the proposition is given in the Appendix.

This proposition shows the difficulties of using PNs for the design of circuits with OR-causal behaviour. Indeed, the equivalence notion introduced here is aimed at establishing the equivalence only with respect to some subset of signals. Usually these are *input* and *output* signals of the circuit, and we are looking for different implementations that have the same input-output behavior. In this case the signals that are not participating in the equivalence relation are the *internal* signals of the circuit. If any signal is non-persistent, *no speed-independent logic circuit can implement the specified behavior*. The non-safety and non-free-choice features of specification may also be a problem, since most known STG synthesis methods work only with free-choice and safe descriptions. Note that all these unpleasant characteristic (non-persistency, non-safety, non-free-choice) of PNs modeling OR causality is the consequence of their inherent AND causality paradigm, because the equivalent CD is safe and correct by assumption.

The situation may however become worse if we have to deal with unsafe OR relations, as in the CD shown in Figure 8. In this CD the OR-event $g+$ can fire twice from the event $d+$ without any occurrence of $b+$ (e.g., the sequence $a+,\ d+,\ g+,\ e+,\ d-,\ g-,\ a-,\ d+,\ g+$). But if event $b+$ happens after $a+,\ d+,\ g+$, then it will not trigger $g+$ and the firing of $b+$ will be "ignored". In ordinary PNs there is no means to distinguish from which event the token is coming to the common place p. But due to the operation rules of the unsafe OR, we have to distinguish the token that comes from the second firing of $d+$ (it affects $g+$) and the token that comes from the $b+$ firing (it does not affect $g+$). This is why there is no PN observation isomorphic to a CD with unsafe OR. However the CD in Figure 8 allows the following semi-modular implementation:

$$a = \overline{ce} + ag; \quad b = e + a \quad c = bc + \overline{a}b + be\overline{d}$$

$$e = aeg + adg + \overline{c}e; \quad d = \overline{ac + ae}; \quad g = d + b\overline{ce}$$

The only way to represent the unsafe OR in PNs is to use the fact that any unsafe but k-bounded CD can be reduced to a safe form by *unfolding* it into k periods ([12]). Such solution, however, is not observation isomorphic because it implies to construct a PN that is equivalent not to the initial CD but to its k period unfolding. Therefore more than one transition in the PN has to be in correspondence to one CD transition. This not only may lead to a significant increase in the size of the specification, but also requires the designer to think in terms of the unfolded behavior, rather than in terms of the more natural and compact unsafe one.

[8] A CD transition is live if it can be enabled infinitely many times in the CD operation.

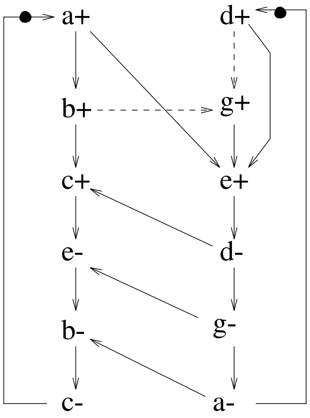


Figure 8: Unsafe CD with OR causality

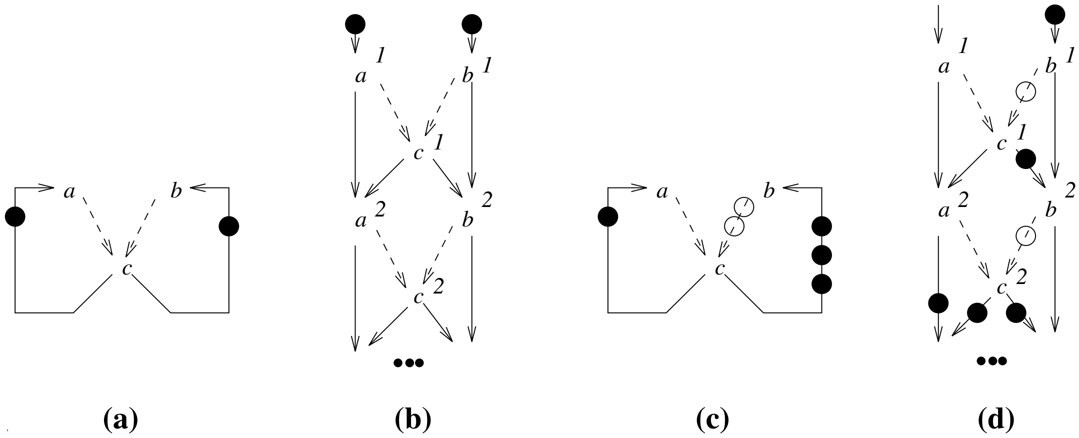


Figure 9: A CD without an equivalent finite Petri net

3.2.2 "Unbounded" cases

The above analysis was applied to the case in which both PN and CD were bounded and could represent the same behaviour. Let us look at a number of "unbounded" cases.

The first example is depicted by a simple CD in Figure 9.(a) with unbounded arc marking.

Such an unbounded behaviour, in which the i-th ($i = 1, 2, ...$) occurrence of transition c is caused either by the i-th occurrence of a or by the i-th occurrence of b, is represented in Figure 9.(b) as a CD unfolding ([12]). In the unfolding each transition a^i represents a *unique occurrence* of the corresponding transition a in a firing sequence of the CD (similarly for b^i with respect to b and c^i with respect to c).

It was first noted in [34], and later formally proved in [22], that there exists no finite PN representing such a behaviour[9].

[9] The proof in [22] is based on reaching a contradiction from the assumption that there is a finite Petri net with exactly three transitions corresponding to

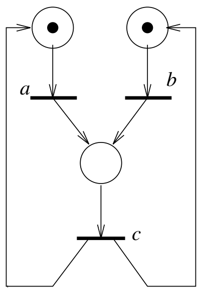


Figure 10: A Petri net only seemingly equivalent to Figure 9.(a)

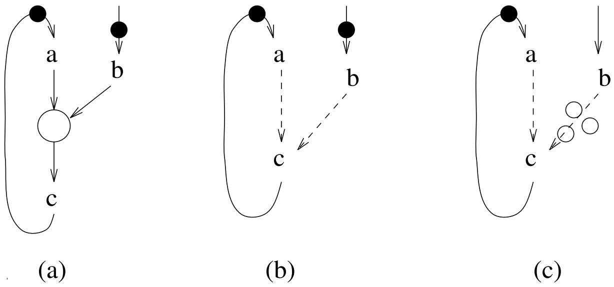


Figure 11: A bounded Petri net with a Change Diagram that is not equivalent to it

The seemingly equivalent PN shown in Figure 10 describes in effect a different behaviour. In it the i-th occurrence of transition c can be caused by any combination of pairs of the form a^k and b^{i-k} where k can be of any value between 0 and i. The difference between these behaviours is obvious.

The CD is able to remember the number of occurrences of transitions a and b, using the negative marking mechanism. So if a fires twice, as represented in Figures 9.(c) and 9.(d) (empty circles represent negative marking on the arc between b and c), and then it stops firing, c can fire again only after b has fired *three* times, in order to "re-absorb" the negative marking. On the other hand in Figure 10.(a), if a fires twice and then stops, c can begin firing again as soon as b fires, because there is no way to remember an *unbounded* "debt" of tokens.

Another example is a PN shown in Figure 11.(a). It models an initially one-place buffer that becomes two-place when transition b occurs. The behaviour is semi-modular, because no transition is disabled. Yet there is no connected equivalent CD, because connected CDs can represent only semi-modular behaviors without *transient cycles*. In this example transition b is continuously enabled during the cyclic firing of a and c.

The CD shown in Figure 11.(b), whose behaviour is only a subset (in terms of the corresponding languages build on the set of firing sequences) of that of the PN in Figure 11.(a), has an *unbounded* negative marking on the arc between b and c (Figure 11.(c) shows such negative marking after the occurrence of b, followed by four occurrences of a and c). The difference in their behaviours begins after the occurrence of transition b.

In order to fully model this behaviour in the CD language, we have to represent the modes of one-place and two-place buffer as separate CDs. Their composition would then require some additional selection mechanism, which would however lead us outside the descriptive domain of the original CDs notation.

Both these examples as well as the fact that CDs (at present [12]) cannot model processes with nondeterminism and conflicts, clearly demonstrate the need for a unified formal notation that would be free from the shortcomings of both.

4 Causal Logic Nets: a Unified Model for Causality in Hardware

In this section we propose a new model that is based on the PN graph but has more general rules defining its dynamics. The model, called Causal Logic Net, inherits the power of both PNs and CDs. We show that both these models are only special cases of the Causal Logic Net. On the other hand, this model is a sort of "least upper bound" of PNs and CDs, so we hope that its analysis will not be drastically more complex than analysing its prototypes. The topic of analysis of Causal Logic Nets is outside the scope of the present paper.

A *Causal Logic Net* (CLN) is a quadruple $\mathcal{N} = \langle T, P, F, \beta \rangle$ where:

- T is a non-empty finite set of transitions,
- P is a non-empty finite set of places,
- $F \subseteq (T \times P) \cup (P \times T)$ is the flow relation between transitions and places, and
- $\beta : T \to \mathcal{F}$ is a function which assigns each transition a Boolean function from the set $\mathcal{F}$ of Boolean functions defined on subsets of P, i.e. $\mathcal{F} = \{f | \exists P' \subseteq P \wedge f : \{0,1\}^{|P'|} \to \{0,1\}\}$, in such a way that $\forall t \in T : \beta(t) : \{0,1\}^{|{}^\bullet t|} \to \{0,1\}$. Note that here and further on we use standard notation for the sets of input and output places of a transition: ${}^\bullet t = \{p | (p,t) \in F\}$ and $t^\bullet = \{p | (t,p) \in F\}$. Analogous notation will be used for the sets of input and output transitions of a place: ${}^\bullet p = \{t | (t,p) \in F\}$ and $p^\bullet = \{t | (p,t) \in F\}$.

It is thus clear that a CLN can be represented as a PN (bipartite) graph, in which each transition is associated with a Boolean function defined on its input places. This function is called the *enabling function* of the transition. It can be

the transitions a, b and c in the CD shown in Figure 9, such that the set of feasible sequences generated by the net is equal to that of the CD.

written as an expression (with brackets) using the standard mathematical notation accepted for Boolean functions, in which each place $p \in {}^\bullet t$ is associated with a literal (for simplicity, we shall use the same name p). For each transition t the enabling function $\beta(t)$ is evaluated according to the marking of the net. A CLN marking is defined similar to a PN marking $m : P \rightarrow \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ except that the range of values is the full integer range. Thus, a *marked* CLN is a quintuple $\mathcal{N} = \langle T, P, F, \beta, m_0 \rangle$, where m_0 denotes its initial marking.

We say that a Boolean literal p associated with a place $p \in P$ is evaluated with the logical 0 if $m_0(p) \leq 0$ and it is evaluated with the logical 1 if $m_0(p) > 0$. A transition $t \in T$ is *enabled* at a marking m if all its predecessor places are marked in such a way that the enabling function evaluates to 1. For example, if for t with ${}^\bullet t = \{p_1, p_2\}$ and $\beta(t) = p_1 p_2$ the initial marking m_0 is such that $m_0(p_1) = 0, m_0(p_2) = 1$ t is not enabled because $\beta(t) = 0$. On the other hand, if we define $\beta(t) = p_1 + p_2$, then the same initial marking makes t enabled since now $\beta(t) = 1$. The first situation corresponds to the case of strong (AND) causality for t, while the second is an example of weak (OR) causality.

The remaining task is now to define the transition firing rule. First of all, like in PNs, we say that any transition $t \in T$ *may fire* under m (initially, $m = m_0$), producing a new marking m', if it is enabled under m.

Firing Rule 1 ("Unconditional firing"). The new marking m' is defined in the same way as in ordinary PNs: $\forall p \in P$: $m'(p) = m(p) - 1$ if $p \in {}^\bullet t,\ p \notin t^\bullet, m'(p) = m(p) + 1$ if $p \in t^\bullet,\ p \notin {}^\bullet t$, and $m'(p) = m(p)$ otherwise. Thus, according to this rule some of the places, that may originally be marked by m with nonnegative number of tokens, can be marked negatively ("tokens are borrowed") in m'.

Let us call this firing rule Unconditional Firing because we impose no condition on its realisation except for the fact that the transition t needs to be enabled. Later we shall introduce another firing rule which will take into account some condition; then using both rules would enable us to model a larger class of behaviours.

With the effect of this firing rule we say that m' is directly reachable from m through the firing of t, using the ordinary PN notation $m[t > m'$.

We define all other notions standard for PNs, such as firing sequence, reachability, reachability set and reachability graph in an analogous way.

We now give a number of propositions showing the relationship between CLN and the previously used models. They are quite important for understanding why a particular class of CLNs can be the "least" possible generalisation of PN and CD. The following proposition is rather obvious.

Proposition 4.1 *A CLN $\mathcal{N}$ is a PN iff*

1. *for each transition $t \in T$ its enabling function $\beta(t)$ is a positive unate conjunction on all its input place literals (β assigns only AND expressions with positive literals), and*

2. *the initial marking of each place is nonnegative.*

Consider now a CLN $\mathcal{N}$ together with a signal transition labelling $\Delta : T \rightarrow A$, where A is a set of signal changes for a set of signals Y. We can thus state that such an interpreted $\mathcal{N}$ is an STG iff its underlying CLN satisfies the conditions of Proposition 4.1.

An analogous proposition holds for the reduction of CLN to CD.

Proposition 4.2 *A CLN $\mathcal{N}$ together with the signal transition interpretation Δ is a CD without disengageable arcs iff*

1. *for each transition $t \in T$ the enabling function $\beta(t)$ is either a positive unate conjunction or a positive unate disjunction of literals associated with all places from ${}^\bullet t$ ($\beta(t)$ can either be an AND or OR expression with positive literals),*

2. *for each place $p \in P$ the sets of predecessor and successor transitions contain at most one transition, i.e. $|p^\bullet| \leq 1$ and $|{}^\bullet p| \leq 1$, and*

3. *the initial marking of a place is either 0 or 1.*

The proof of this proposition is trivial. Furthermore, we can demonstrate that the CLN with the qualities given by item 1 of this proposition is also capable of adequate modelling of disengageable arcs. As stated in [12], in well-formed CDs, every disengageable arc connects a non-repeated transition with a cyclic transition. In this case, the modelling is very simple. It is shown in Figure 12(a). For the general case when the disengageable arc is outgoing from a transition which can have other outgoing arcs that are not disengageable we have to refer to Figure 12(b). It shows a CD fragment with such an arc together with its associated PN fragment, that is observation isomorphic with respect to the set of transitions in the CD.

Based on the notion of CLN we can define an updated version of an STG, by analogy with the original version of STG defined on a PN. It would however be more appropriate to give it a separate name, Causal Logic Signal Transition Graph (CL-STG). Thus, a CL-STG is a tuple $\mathcal{G} = \langle \mathcal{N}, X, Z, \Delta \rangle$ where $\mathcal{N}$ is a *marked* CLN, X and Z are disjoint sets of input and output signals respectively and $\Delta : T \rightarrow (X \cup Z) \times \{+, -\}$ labels each transition of $\mathcal{N}$ with a signal change symbol.

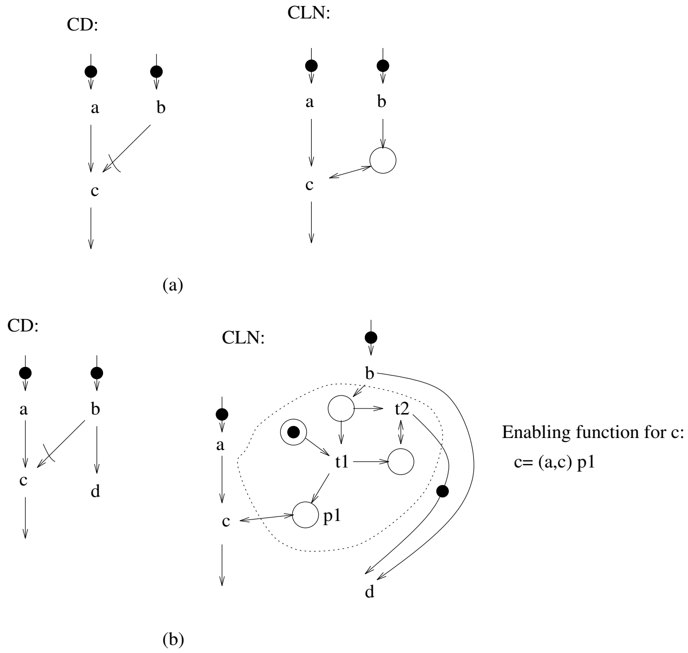


Figure 12: Modelling disengageable arcs in Causal Logic Nets

We redefine the properties of a CL-STG by analogy with STGs. The most important of them is *validity* which allows the construction of the STD model. Validity (finiteness, boundedness and consistent labeling) remains unchanged for CL-STGs, since the underlying reachability graph for a CLN is defined in the same way as for ordinary PNs.

In this paper we do not attempt to investigate all the problems of analysis of properties and classes of CLNs. Instead, we have only demonstrated that this model allows the representation of any type of causality describable by means of Boolean logic. Later, we shall show how a slight extension of the firing rule, making the resulting marking conditional upon the enabling marking, can allow the CLN model to have the descriptive power of a Turing Machine. The same extension also allows to represent a non-deterministic behaviour with non-commutativity that can be useful in modelling such devices as the Fair Arbiter from [34].

5 Circuit Synthesis Examples

In this section we show two examples of synthesis of speed-independent circuits from initial specifications using CL-STGs. These two examples also illustrate the two major types of OR causality that we originally claimed to be a major motivation for the introduction of the CLN model.

The first example is an event-based *inclusive OR* element (see [23]). The CL-STG description of this element is shown in Figure 13.(a). The element, having two inputs x_1 and x_2 and two outputs y_1 and y_2, behaves exactly as our "cautious client" described earlier. Starting from the initial states where all the signals are at 0, y_1 changes its output from 0 to 1 whenever *either* of its inputs changes from 0 to 1. The inputs cannot change until the other output y_2 has also been set to 1, which happens when both are at logical 1. Later, inputs can change back to 0 in any order, and the output y_1 follows the first of these, while y_2 again ensures "safe" operation by checking that both inputs are at logical 0.

We assume that in the CL-STG in Figure 13.(a) the β function for the transitions labelled with the changes of y_1 is a simple disjunction between the two input arcs [10]. The β function of all the remaining transitions is AND.

Figure 13.(b) shows an STD, which is semi-modular with respect to all signals. We can thus derive the circuit implementation using any of the existing techniques (e.g., [24, 32, 4, 14, 12]). A set of Boolean functions for the circuit is:

$$\begin{aligned} y_1 &= x_1x_2 + (x_1 + x_2)\overline{y_2} \\ y_2 &= x_1x_2 + (x_1 + x_2)y_2 \end{aligned}$$

[10] We resort to the usual "shorthand" notation style, in which places are explicitly shown only where they have more than one incoming or outgoing arc.

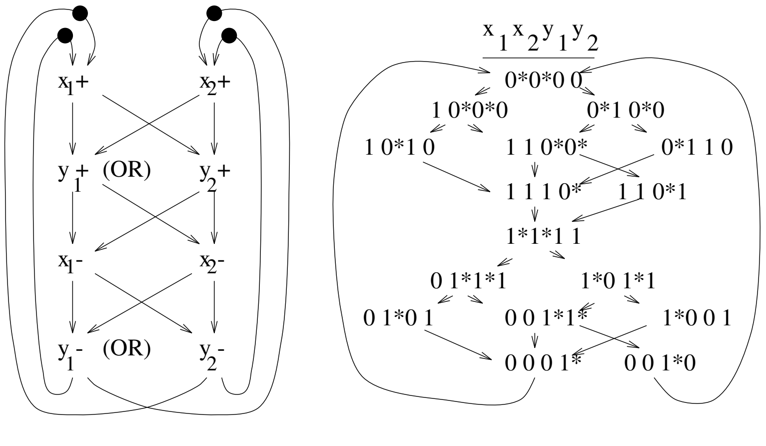


Figure 13: A Causal Logic State Transition Graph and its State Transition Diagram for the inclusive OR example

The second example is a *variable capacity* buffer, which operates initially as a one-place buffer, but upon the arrival of a request signal becomes a two-place buffer. Note that we use the term "buffer" meaning its main control flow functionality, without concerns about data path functions.

The buffering property is therefore characterised here by the number of times one handshake pair (the input of the buffer) can change its value while the other handshake pair remains in the same state. A purely abstract model, where each handshake was denoted simply by one symbol, was shown in Figure 11.(a). We shall draw upon this model in constructing a corresponding "signalling expansion", which is shown in Figure 14.(a). Here signals a and c stand for the outputs of the handshakes. The inputs are "hidden" because they are simply delayed versions of the outputs. The environment is assumed to be acknowledging the requests of the circuit, where the semantics of request is "the circuit is ready" (this is the so called "passive environment" interaction [1]).

Note that the CL-STG in Figure 14.(a) is actually an STG because, in this example, we do not want to use the causal logic model for transition $c+$. This decision is due to the idea of keeping an extra token, arriving as a result of the firing of $b+$, inside the cycle. Had we used the explicit OR causality on $c+$ in conjunction with the adopted firing rule, the behaviour would have been inadequate to our original intentions because the number of tokens in the cycle would not have increased, so we would not have got the effect of a two-place buffer. The purpose of this example is to illustrate a second type of OR causality, as described in the next section.

The initial STG model is valid, but cannot be implemented directly in logic, because it first requires the introduction of hidden *state signals*. For this purpose, we introduce three internal signals, x_1, x_2 and x_3, in such a way that their transitions do not change the original ordering of the initial specification. This modified STG is shown in Figure 14.(b). The STD generated by this STG is shown in Figure 14.(c). From this STD it is now possible to derive the Boolean functions of the circuit implementation of the buffer (both software systems, SIS [27] and Forcage [12], produced the same solution):

$$
\begin{aligned}
a &= \overline{x_1 x_3} \\
x_1 &= a + \overline{x_3} x_1 \\
x_2 &= a + \overline{x_3} x_2 \\
x_3 &= \overline{a}\overline{c} x_1 x_2 + (x_1 + x_2) x_3 \\
c &= b\overline{x_2 x_3} + \overline{a} x_1 \overline{x_2 x_3}
\end{aligned}
$$

6 The Two Types of OR Causality

The above two examples demonstrate the two major paradigms in which OR causality can be distinguished. The first is the case when the actions, say a and b, that weakly cause another action, c, *do not insist* on the effect of their completion to be applied to c independently of each other. This means that for every possible occurrence of c after only one of the causes, say a, the other cause, say b, has no effect over the same occurrence of a. The unbounded case of such a paradigm was shown in Figure 9.(a), for which we could not build a finite PN representation. Now, using CLNs, the required model would be a trivial re-drawing of the CD in Figure 9.(a) in such way that the transition labelled with c must have its β function equal to the simple disjunction of the input arcs.

Let us call this type of OR causality *joint* OR causality.

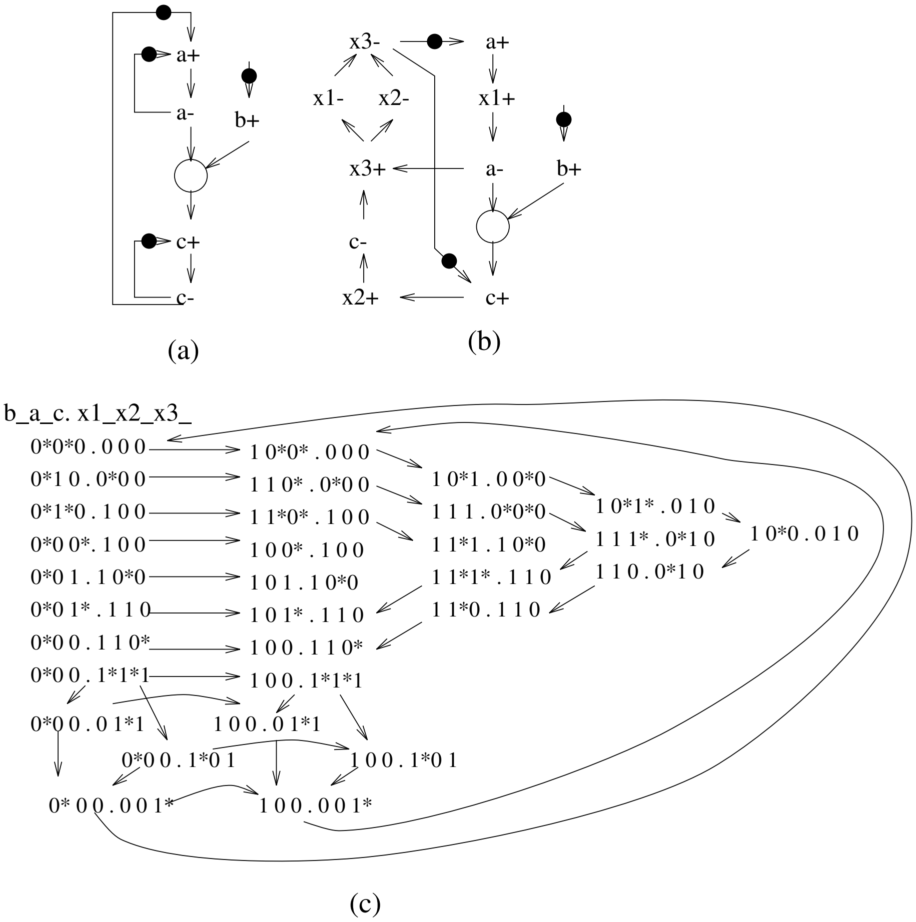


Figure 14: The “variable capacity” buffer example

The other type, called *disjoint* OR causality happens to be in the example of the variable capacity buffer. It also took place in the example of the low latency arbiter, whose STG was shown in Figure 6. This type of OR causality, inherited intact from PNs, is called disjoint because we do not allow the tokens independently arriving in one place p from several cause actions to be removed or annihilated. This *additive* effect of the PN marking mechanism is adequate for the purposes of modelling. A more transparent illustration of the fact that a joint OR causality construct will not be able to represent this effect is shown in Figure 15. If we analyse the behaviour of the CLN shown in Figure 15.(a), we will see that it is negatively unbounded with respect to either of the dashed arcs. Furthermore, we cannot satisfy the requirement that in every execution sequence the action labelled with c has to occur as many times as the *sum* of the occurrences of actions a_1 and a_2. This requirement would be necessary to guarantee, for example, that *none of the requests to an arbiter are lost*. A satisfactory CLN model of this causality (in this case an ordinary PN) is shown in Figure 15.(b).

To summarise, the above differentiation of the OR causality resolves the uncertainty as to how the designer has to specify OR causality. Although the formal relationship between the class of semi-modular behaviours and CDs has been formally shown (as well as the corresponding classes of PNs and STGs), there has been no clear recommendation for the designer as to what construct had to be employed for modelling OR causality. It is now clear that the demonstrated modelling mismatches were due to:

- the inadequacy of PNs to model *joint* OR causality and
- the inadequacy of CDs to model *disjoint* OR causality.

On the other hand, using the unified model of CLNs as an underlying description tool for CL-STG, the designer can use:

- a disjunctive transition enabling function for joint OR causality, or
- a single place with multiple predecessors for disjoint OR causality.

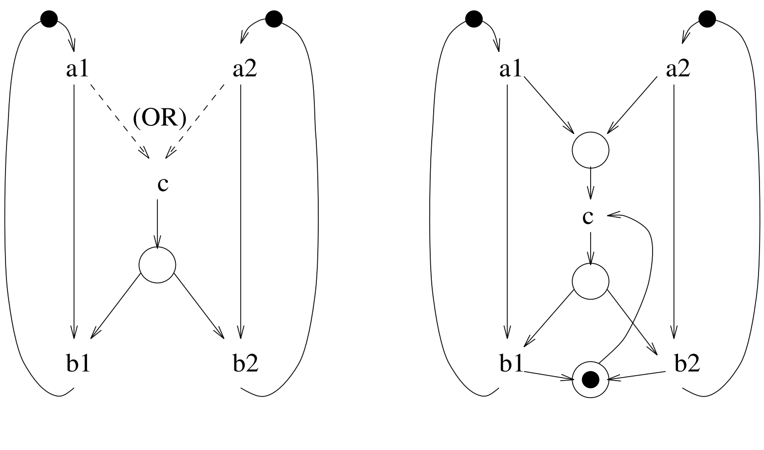


Figure 15: Illustration of disjoint OR causality

7 Extensions of Causal Logic Nets

7.1 Modelling Inhibitor Nets

There are two major parts in the present day theory of Petri nets. One (by analogy with Theory of Relativity, it can be called Special Theory of Petri Nets) studies the model of ordinary P/T-nets, while the other (that can be called General Theory of Petri Nets) deals with the various modelling extensions of ordinary nets. Such extensions (e.g., Inhibitor Nets, Self-modifying Nets etc.) increase the descriptive power of the original model (however, at the cost of losing the ability to analyse the model in its most general class). These generalisations bring the classical Petri net model to the level of Turing Machine [21]. It is known for example that Inhibitor Nets have such descriptive power[30].

It is worth noting that Inhibitor Nets have the same structure as ordinary nets, with the exception that some pairs (p, t) in the flow relation F can be declared as *inhibitor* arcs. Let denote such pairs by F^i, separating them from F. As a result the enabling rule for such nets is slightly different from that of ordinary PNs. A transition $t \in T$ is called enabled under the marking m if $\forall p \in {}^\bullet t : m(p) \geq 1$ and $\forall p' : (p, t) \in F^i : m(p') = 0$, where ${}^\bullet t$ refers to the ordinary flow relation F. Informally, the transition can *only* be enabled if its input places connected by inhibitor arcs contain no tokens.

The firing rule of Inhibitor Nets is the same as for ordinary nets except for the fact that the inhibitor places incident to the firing transition cannot be decremented of tokens (because these places are empty). In other words, some condition is checked before applying the marking re-computation during the transition firing.

In this section we demonstrate that similar functionality can be added to CLN to increase its descriptive power.

First of all, let us define this alternative firing mechanism.

Assume that transition $t \in T$ is enabled under the marking m:

Firing Rule 2 ("Conditional firing"). The new marking m' is defined in the following way: $\forall p \in P : m'(p) = m(p) - 1$ if $p \in {}^\bullet t \wedge m(p) \geq 1$, $m'(p) = m(p) + 1$ if $p \in t^\bullet$, and $m'(p) = m(p)$ otherwise. This rule is called conditional because a place in the set of input places of the enabled transition is decremented only if it has at least one token. Thus, a net with this firing rule never has a negative marking.

The class of CLNs, in which transitions can be associated with these two firing rules (but each transition must obviously be associated with just one rule, either Rule 1 or Rule 2) will be called Extended CLNs or E-CLNs.

Depending on which firing rule we apply in each concrete case of transition firing we shall say that m' is directly reachable from m through either Rule 1 or Rule 2. We shall denote it respectively as either $m[t >^{R1} m'$ or $m[t >^{R2} m'$. If it creates no confusion we simply use the ordinary, unspecialised notation $m[t > m'$.

The following proposition, which is quite trivial, is now true.

Proposition 7.1 *An E-CLN N is an Inhibitor Net iff*

1. *each of its transitions is associated with a simple conjunction on its input place literals (some literals can be with inversion), i.e. both AND and NOT Boolean functions can be used,*
2. *$\mathcal{N}$ has an nonnegative initial marking, and*
3. *for every transition the applied firing rule is Rule 2.*

Using this proposition and the result of [30] about the modelling power of Inhibitor Nets, we can now easily state that the class of E-CLNs has the same descriptive power as Turing Machine.

7.2 Modelling Non-commutative Behaviour

At this point we should stop introducing new modelling functionalities and examine more carefully some important semantic properties achieved at the state-transition level by the features which have been added with the models of CLN and E-CLNs. Besides different types of causality captured in explicit way through the use of Boolean enabling functions, one important aspect of behaviour that has been affected is the way in which non-determinism and behavioural divergence (or non-confluence) can be modelled in CLN based descriptions.

The classical PN formalism is known to depict non-deterministic behaviour by using the notion of a conflict between enabled transitions, which is achieved through the modelling of choice in PNs. The notion of a conflict, which reflects a relational approach to this phenomenon, has a precise operational characterisation of the net's state transition semantics, which is known as the property of non-persistency. The results of Keller [10] on the characterisation of confluence in concurrent systems show that there are two fundamental ways in which the divergence can be introduced into the model. One is non-persistency, the other is non-commutativity.

According to Keller, a state transition system is called *commutative* if for every reachable state (marking, if we want to applying the Petri net terminology) m and a pair of transitions t_1 and t_2 enabled in m, such that they can fire in any order, t_1, t_2 or t_2, t_1 (i.e. both firing sequences are feasible from m), the system reaches the same state m' no matter which is the order of firing between t_1 and t_2. A state transition system is called *deterministic* if for each state m, the state m' such that $m[t > m'$ is unique. A state transition system is called *confluent* if for all states m, m' and m'' such that m' and m'' are both reachable from m, then there exists a state m''' reachable from both m' and m''.

The main result of [10] states that a state transition system is confluent if it is deterministic, persistent and commutative.

In this paper, we use a more general property than commutativity, which is called permutability. A state transition system is called *permutable* if for every reachable state m and any pair of feasible sequences σ_1 and σ_2 of transitions $t_1, t_2, \ldots, t_n$, such that σ_2 is a permutation of σ_1, the system reaches the same new state m' through either of these sequences.

An interesting property of ordinary PNs as well as their powerful modelling extension, Inhibitor Nets is that they are intrinsically permutable (they are certainly deterministic for a single transition, as well), which implies that the only way to model global divergence in their state transition semantics is through adding non-persistency. The following proposition states this in a slightly more general form (i.e. the proposition trivially implies the permutability of ordinary PNs and Inhibitor Nets).

Proposition 7.2 *1. For every CLN $\mathcal{N}$ the associated state transition system (i.e. Reachability graph) is permutable.*

2. For every E-CLN $\mathcal{N}$ in which every enabling function is a simple conjunction consisting of simple literals and literals with inversion the associated state transition system (i.e. Reachability graph) is permutable.

The proof is given in Appendix.

This proposition raises the following questions. Can we model non-permutable behaviour with the new CLN formalism? Why is the modelling of such behaviour so important to us? In other words how is this relevant to the problems of asynchronous circuit design?

The first question can be answered yes. The following example illustrates this fact.

Consider a simple E-CLN shown in Figure 16(a), in which two transitions a and b are concurrently enabled. Let $\beta(a) = p_1 p_2$ and $\beta(b) = p_1 + p_2$. The result of the firing of these transitions in either order is not the same however. The Reachability Graph shows this in Figure 16(b). The marking after the firing sequence a, b will be p_5 whereas after the sequence b, a it will be p_4, p_5.

The question of usefulness of non-permutability is concerned with demonstrating a specification of some useful circuit. One such example is the behaviour of a Fair Arbiter that was originally presented [34]. Figure 17 shows the structure of the system consisting of two Request modules (labelled with **a** and **b**) and Arbiter (marked with **c**), as well as an STD describing the system's behaviour. Here, R_a and R_b denote two request signals from the Request modules, while A_a and A_b denote the corresponding acknowledgement, or grant, signals generated by the Arbiter.

It is easy to see that this STD is not permutable because the behaviour is sensitive to the order in which the requests are asserted. For the sequence R_a, R_b, the system enters state $s7$, while for the sequence R_b, R_a it goes to $s13$.

Due to the ability to distinguish two states labelled with the same binary code 1100 this model is capable of remembering the fact that the second request (e.g., R_b) must be served immediately after the first (e.g., R_a) before the next activation of the first request is served. This capability makes the Arbiter's serving discipline fair.

The key point in providing fairness in the model is in "splitting" the state, in which two requests are asserted at the same time, into two different states. This is possible only by allowing the STD be non-permutable.

The problem of defining such a behaviour within an event-based framework of the purely causal model, is in the way how to represent each signal transition by an individual event, so that the model will preserve the idea of concurrency between two

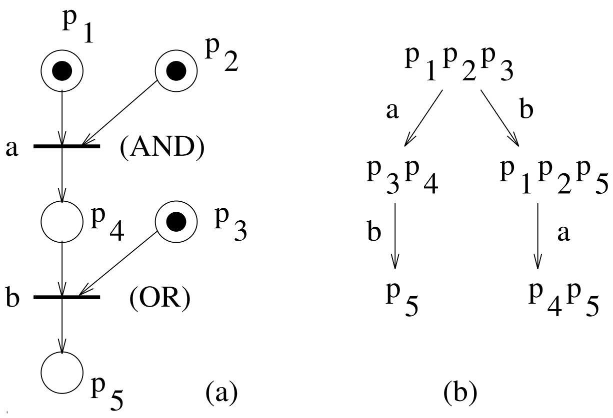


Figure 16: An Extended Causal Logic Net whose behaviour is not permutable

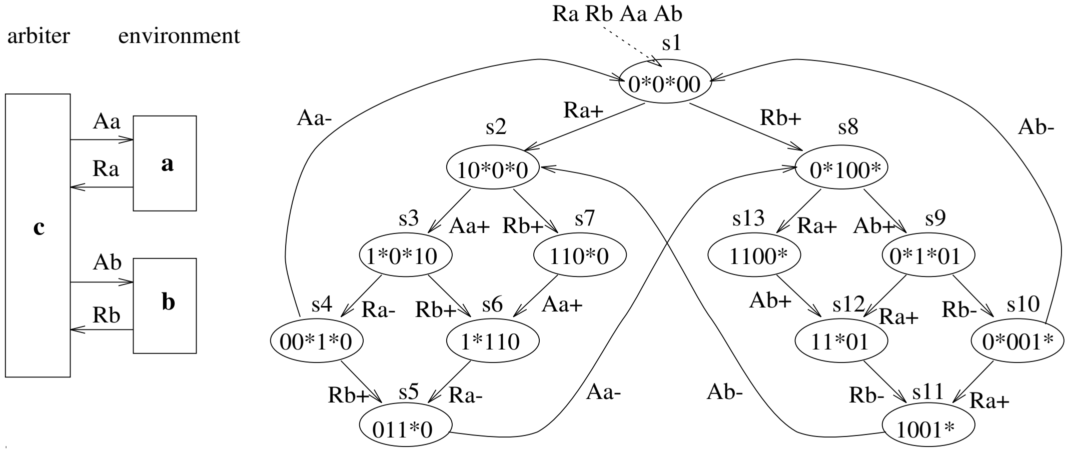


Figure 17: State Transition Diagram for a Fair Arbiter

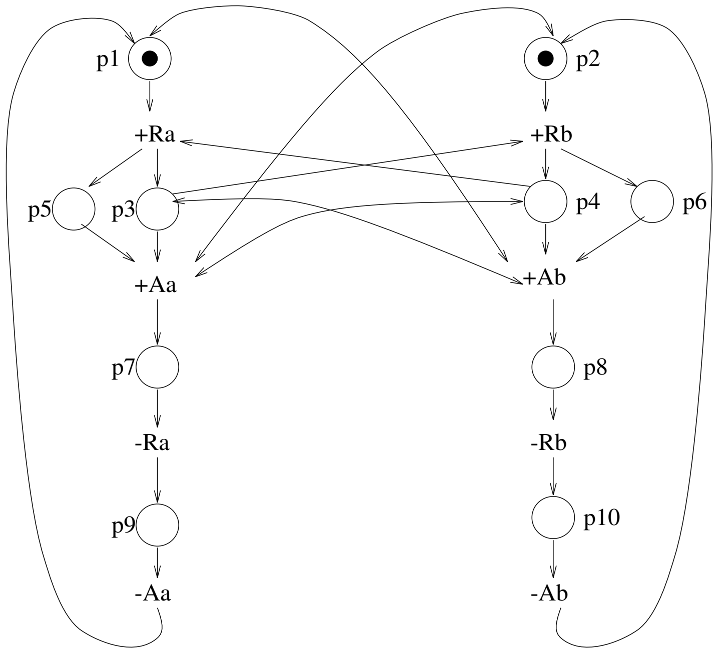
requests, R_a+ and R_b+. The arbiter's behaviour can thus be modelled in a descriptively "fairer" way by the E-CLN shown in Figure 18.

This net has exactly the same number of transitions as there are signal transitions involved in the STD. It uses both features that have been brought with the E-CLN model. Firstly, the enabling functions need to involve AND, OR and NOT. Secondly, the Firing Rule is used to enable the modelling of non-permutative behaviour.

8 Conclusions

In this paper, we have demonstrated the role of OR causality in modelling asynchronous circuits. We have shown that within the framework of only the PN or the CD model the designer has certain limitations which can be crucial in capturing some useful semantic details. We have thus justified the introduction of a unified formalism, Causal Logic Nets, which is based on the classical graphical background of Petri nets but augmented with the notion of a complex causal enabling function defined for each transition of the net. We have been able to identify the class of Causal Logic Nets which is the least possible generalisation of PN and CD. Such a class, as follows from Propositions 4.1 and 4.2, is formed by CLNs whose transitions have enabling functions that are either nonnegative simple conjunctions or nonnegative simple disjunctions of literals associated with input places. This class of CLNs enables the designer to model two types of OR causality inherent in circuit behaviour. The joint OR causality can be represented through the enabling function mechanism of transitions, while the disjoint OR causality can be modelled by net places and their input transitions.

This model can however be insufficiently powerful to represent more complex features, such for example as those of extended Petri nets. By means of separating the notions of enabling and firing conditions we achieve the ability of having the modelling power of Turing Machine (reducing our Extended Causal Logic Net to Inhibitor Nets, known to have such descriptive power). Furthermore, the combination of a wider causal logic types and a more flexible firing mechanism results in the ability to model global nondeterminism of the operational (state transition) semantics not by the traditional notion



Firing Rule 2 is applied

(a input place is decremented ONLY if it has positive number of tokens)

Enabling functions:

+Ra: p1	+Rb: p2
+Aa: p5(p2+p4)	+Ab: p6(p1+p3)
-Ra: p7	-Rb: p6
-Aa: p9	-Ab: p10

Figure 18: Causal Logic Net model of a Fair Arbiter

of conflicts (non-persistency) but rather by the effect of non-commutativity. Here we used a more general property called (non)permutability. The latter has allowed the representation of the behaviour of a fair arbiter in a purely causal modelling framework.

The following table summarises our classification of models and properties in terms of the essential features of the new CLN formalism.

model/property	**CLN features:**	
	enabling type	**firing type**
PN (STG)	AND	Rule 1
PN(STG)+CD	AND+OR	Rule 1
Inhibitor Net	NOT+AND	Rule 2
Non-permutability	OR	Rule 2

Thus, for example, the modelling non-permutability requires both using OR causality and Firing Rule 2. Using either of these features, without the other, appears to be insufficient.

Acknowledgements

We are grateful to Leonid Rosenblum and Alexander Taubin who co-authored our research in Signal Graph and Change Diagram models. Thanks to Eike Best, Maciej Koutny and Marta Pietkewicz-Koutny for discussions of this paper and to Michael Yoeli and other participants of the discussion on Petri Nets and Change Diagrams at the Workshop on “Self-Timed Design” (Dagstuhl, 1992).

References

[1] K. van Berkel. *Handshake circuits: an intermediary between communicating processes and VLSI.* PhD thesis, Technical University of Eindhoven, 1992.

[2] T.-A. Chu. On the models for designing vlsi asynchronous digital systems. *Integration: the VLSI journal*, 4:99–113, 1986.

[3] T.-A. Chu. Automatic synthesis and verification of hazard-free control circuits from asynchronous finite state machine specifications. In *Proceedings of ICCD'92*, Cambridge, MA, October 1992.

[4] T.A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications*. PhD thesis, MIT, 1987.

[5] F Commoner, A.W. Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer and Systems Sciences*, 5:511–523, 1971.

[6] J.C. Ebergen. *Translating Programs into Delay-Insensitive Circuits*, volume 56 of *CWI Tract*. CWI, Amsterdam, 1989.

[7] J. Gunawardena. Causal automata. *Theoretical Computer Science*, 101(2):265–288, 1992.

[8] M. Josephs and J.T. Udding. Delay-insensitive circuits. In *Lecture Notes in Computer Science, Vol. 458*. Springer-Verlag, 1990.

[9] M.B. Josephs and J.Yantchev. Low latency asynchronous arbiter: Patent application 9308161.0. Technical report, Oxford University Computing Laboratory, 1993.

[10] R.M. Keller. A fundamental theorem of asynchronous parallel computation. *Lecture Notes in Computer Science*, 24:103–112, 1975.

[11] M. Kishinevsky. *Analysis and implementation of speed-independent circuits*. PhD thesis, Leningrad Electrical Engineering Institute, 1982. (in Russian).

[12] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. John Wiley and Sons, London, 1993.

[13] M.A. Kishinevsky, A.Y. Kondratyev, and A.R. Taubin. Formal method for self-timed design. In *Proc. EDAC'91*, pages 197–201, 1991.

[14] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Kluwer Academic Publishers, Boston, 1993.

[15] A.J. Martin. Synthesis of asynchronous vlsi circuits. In J. Staunstrup (ed.), editor, *Formal Methods for VLSI Design*, chapter 6. North Holland, Amsterdam, 1990. IFIP WG 10.5 Lecture Notes.

[16] D.G. Messerschmitt, T.H.-Y. Meng, and R.W. Brodersen. Automatic synthesis of asynchronous circuits from high-level specifications. *IEEE Trans. on CAD*, 8:1185–1205, November 1989.

[17] R.E. Miller. *Switching Theory*, volume 2: Sequential Circuits and Machines, chapter 10. Wiley-Interscience, New York, 1965.

[18] D. Muller and W. Bartky. A theory of asynchronous circuits. In *Annals of Computation Laboratory*, pages 204–243. Harvard University, 1959.

[19] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4):541–580, April 1989.

[20] S.M. Nowick and D.L. Dill. Automatic synthesis of locally-clocked asynchronous state machines. In *Proceedings of ICCAD'91*, Santa Clara, CA, November 1991.

[21] J.L. Peterson. *Petri Net Theory and Modeling of Systems*. Prentice-Hall, Englewood Cliffs, N.J., 1981.

[22] Marta Pietkiewicz-Koutny. Proof of the conjecture that language generated by certain change diagram cannot be generated by Petri nets. Manuscript, September 1993.

[23] D.A. Pucknell. Event-driven logic (edl) approach to digital systems representation and related design processes. *IEE Proceedings-E*, 140(2):119–126, 1993.

[24] L.Ya. Rosenblum and A.V. Yakovlev. Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets, Torino, Italy, July 1985*, pages 199–207. IEEE Computer Society, 1985.

[25] C. L. Seitz. *System timing. In: Mead, C.A. and L.A. Conway, Introduction to VLSI Systems*, pages p. 218–262. Addison-Wesley, 1980.

[26] C.L. Seitz. Ideas about arbiters. *Lambda*, 1(1):10–14, 1980.

[27] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, R.K. Brayton H. Savoj, P.R. Stephan, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, University of California at Berkeley, 1992.

[28] I. Sutherland, C. Molnar, R. Sproull, and T. Mudge. The TRIMOSBUS. In *First Caltech Conference on VLSI*, Pasadena, CA, January 1979.

[29] I.E. Sutherland. Micropipelines. *Communications of ACM*, 32(6):720–738, 1989.

[30] R. Valk. On the computational power of extended Petri nets. In *Proceedings of the Seventh Symposium on Mathematical Foundations of Computer Science*, September 1978. Lecture Notes in Computer Science, 64, Springer-Verlag, Berlin.

[31] P. Vanbekbergen et al. Optimized synthesis of asynchronous control circuits from graph-theoretic specifications. In *Proceedings of Int. Conf. on CAD (ICCAD'90)*, pages 184–187, November 1990.

[32] V. Varshavsky, M. Kishinevsky, V. Marakhovsky, V. Peschansky, L. Rosenblum, A. Taubin, and B. Tzirlin. *Self-Timed Control of Concurrent Processes*. Kluwer AP, Dordrecht, 1990.

[33] V.I. Varshavsky, M.A. Kishinevsky, A.Y. Kondratyev, L.Y. Rosenblyum, and A.R. Taubin. Models for specification and analysis of processes in asynchronous circuits. *Soviet Journal of Computer and System Sciences*, 26(2):61–76, 1989. Russian Edition - 1988.

[34] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli. A unified STG model for asynchronous control circuit synthesis. In *Proceedings of ICCAD'92*, Santa Clara, CA, November 1992.

[35] A. Yakovlev and A. Petrov. Petri nets and parallel bus controller design. In *Proc. 11th Int. Conf. on Applications and Theory of Petri Nets, Paris, June 1990*, pages 244–264, 1990.

[36] A Yakovlev, A. Petrov, and L. Lavagno. High speed asynchronous arbiter. Technical Report 427, University of Newcastle upon Tyne Computing Science, May 1993.

[37] A.V. Yakovlev. On limitations and extensions of STG model for designing asynchronous control circuits. In *Proceedings of ICCD'92*, Cambridge, MA, October 1992.

[38] J. Yantchev and I. Nedelchev. Implementation of a packet switching device as a delay-insensitive circuit. In *Proceedings of the Symposium of Integrated Circuits*, 1993.

Appendix

Proof of Proposition 3.1

In the given CD $\mathcal{D}$ the following sequences are feasible due to the definition of OR-causal relations: $\{abc, bac, acb, bca\}$. In any observation isomorphic PN $\mathcal{P}$, however, the firing of transitions a, b and c can be mixed up with the firing of some auxiliary transitions that are not used under the considered observation isomorphism. We will prove the statement by induction on the number of "intermediate" transitions between a and c firing and between b and c firing.

Induction basis. Assume that $\mathcal{P}$ has no transitions "between" a and c and b and c. So there is a marking m in which both a and b are enabled and after firing any of them c can fire. Consider the set of input places of c. It can be divided in two subsets: P_1, consisting of the places that contain some token under m, and P_0, which consists of the places that are empty under m. It is easy to see that any place p in P_0 is an output place of both a and b, because c has to fire immediately after the enabled a or b will fire.

Case 1. Assume that the firing of a does not disable b. Then after the occurrence of b two tokens will appear in each $p \in P_0$ and the PN $\mathcal{P}$ is thus unsafe. Let us show that $\mathcal{P}$ is also non-free-choice. Evidently, $P_1 \neq \emptyset$, otherwise $\mathcal{P}$ generates a feasible sequence a, b, c, c which has no equivalent in $\mathcal{D}$. Since c is a live event in $\mathcal{D}$, it can fire repeatedly. Moreover, $\mathcal{P}$ is observation isomorphic to $\mathcal{D}$, and therefore c can also fire repeatedly. This means that some time after the first firing of c the tokens have to appear in all the places of P_1. This cannot be due to the new firing of a or b because the occurrence of c is not mediated from a and b by other transitions. So the marking m' in which a or/and b is again enabled and all places from P_1 contain a token is reachable in $\mathcal{P}$ after the first occurrence of a, b and c. Recall that after the first firing of a and b all places from P_0 contain two tokens. These places cannot be the input places only to c, otherwise c would be enabled in m'. Then the sequence $a, b, c \ldots c$, which has no equivalent in $\mathcal{D}$ (c can only occur in $\mathcal{D}$ due to the occurrence of either a or b) would be feasible in PN $\mathcal{P}$. So at least one place $p \in P_0$ must be an input place to c and to some other transition t. This place is not free-choice because c has at least one more input place that belongs to P_1 ($P_1 \neq \emptyset$). Thus, our statement is valid. PN $\mathcal{P}$ is unsafe and non-free-choice.

Case 2. Assume that the firing of a disables b. Thus a and b are non-persistent and they share the common input place p. Let us show that this place p is non-free-choice. Suppose the opposite. Due to the observation isomorphism of PN $\mathcal{P}$ and CD $\mathcal{D}$ and the existence in the latter of the feasible sequence a, b the sequence a, s, b has to be feasible in $\mathcal{P}$, where $c \notin s$. By our assumption the place p is free-choice and thus is the only input place to a and b. Therefore after sequence s both transitions a and b are enabled. Hence we have the feasible sequence a, s, a in the PN $\mathcal{P}$. No equivalent sequence exists in the CD $\mathcal{D}$, because $\mathcal{D}$ is safe. Thus in Case 2 $\mathcal{P}$ is non-persistent and non-free-choice.

Induction step. Let both a and b be enabled under marking m and assume that all the transitions that can occur without a and b and bring a token to the input places of c have already fired. Denote by P_1 (P_0) the set of input places of c that has (has no) token under marking m. We are considering the case when the firing of c after a (b) requires the firing of intermediate transitions $T_a = \{t_1^a, \ldots, t_k^a\}$ ($T_b = \{t_1^b, \ldots, t_n^b\}$). Let the set T_a be minimal (no transition can be omitted from it). Clearly we need to consider only non-persistent transitions a and b, otherwise Case 2 in the Induction basis applies. After the firing of a and T_a the tokens have to arrive to all places from P_0. Similarly for the firing of b and T_b.

Case 1. $T_a \cap T_b = \emptyset$ and after the firing of T_a all transitions from T_b can occur without firing of c. Then after the firing of T_a and T_b all the places in P_0 will contain two tokens at least. (This is true even if some place p from P_0 is an input to some transition t from T_b, because both T_a and T_b by our assumption are realized independently and both add tokens to the places in P_0.) Now considerations similar to Induction basis Case 1 prove the statement.

Case 2. Let $T_a \cap T_b = \{t\}$. The following feasible sequence must exist in PN $\mathcal{P}$:

1. $a, b, \ldots, t$ and $b, a, \ldots, t$ as a and b are persistent.
2. $a, \ldots, t, b$ and $b, \ldots, t, a$ as T_a can be realized without b and T_b without a.

Thus instead of considering the transitions a, b and c we can consider the triple a,b, t but the number of intermediate transitions from a to t would be strictly less and we can use the induction assumption.

Case 3. Let $T_a \cap T_b = \emptyset$, but after the occurrence of a, T_a some transition $t_i^b \in T_b$ cannot fire. Let t_i^b be the first such transition in T_b. Consider the result of the firing of $s1 = a, T_a, b, t_1^b, \ldots, t_{i-1}^b$. As t_i^b is not able to fire after $s1$ but is able to fire after $s2 = b, t_1^b, \ldots, t_{i-1}^b$, then some place p which is input for t_i^b contains a token in the initial marking m and this token is removed via the firing of a, T_a. It means that p is a common input place for t_i^b and $t_j^a \in T_a$. Consider sequence $s' = a, t_1^a, \ldots, t_{j-1}^a, b, t_1^b, \ldots, t_{i-1}^b$. It is easy to understand that after it both t_i^b and t_j^a are enabled and share the common input place p with one token, i.e are non-persistent. This place p cannot be free-choice because otherwise t_j^a would be enabled in the initial marking M, its firing does not depend on b and a and the set T_a was not chosen minimal. ∎

Proof of Proposition 7.2

1. Consider a reachable marking m and a firing sequence σ which is feasible under m. Let σ bring $\mathcal{N}$ into marking m'. Firing Rule 1, which is applied for $\mathcal{N}$, is similar to the firing rule used for ordinary PNs. Recall that this rule is unconditional (once the transition is enabled) and that we allow the marking can be negative. Due to this rule, the new marking for each place can be computed by the following well-known canonical firing sequence equation [21]:

$$m'(p) = m(p) + \sum_{t \in {}^\bullet p} [\sigma](t) - \sum_{t \in p^\bullet} [\sigma](t),$$

   where $[\sigma](t)$ is the number of occurrences of $t \in T$ in sequences σ. The $|T|$-dimensional integer vector $[\sigma]$ which can be built from all $[\sigma](t)$ for firing sequence σ is called *firing vector.* It is clear that the value of this vector does not depend on the order in which the transitions involved in sequence σ are actually fired. Therefore, for any two (or more) feasible sequences which are permutations of one another, this value will be the same, and hence all such sequences will bring the net into the same marking. The Reachability Graph of such a CLN is permutable.

2. Consider an E-CLN in which each enabling function is a simple conjunction consisting of literals in direct or inverted forms. Let us take an arbitrary reachable marking m and a sequence σ feasible under this marking. Let the net reach marking m' through σ. Note that if σ contains only transitions whose enabling functions are positive (no inverted literals), then no matter which firing rule is used for transitions, the argument used in the previous item will bring us to the fact that every feasible permutation of σ will also bring the net into m'. Indeed, even if Firing Rule 2 is applied, this rule has the same effect on the marking change process as Rule 1 because each transition is fired when all its input places contain tokens.

   Let σ have at least one transition t whose enabling function $\beta(t)$ is a conjunction containing inverted place literals. Denote the subset of ${}^\bullet t$ whose literals are inverted in $\beta(t)$ by $({}^\bullet t)^i$.

   Assume that t is fired under marking m_1 and this brings the net into marking m_2.

   Consider two possibilities concerned with the firing rule for t. If t is assigned Firing Rule 1, then again the marking m_2 can be calculated unconditionally, decrementing and incrementing the places in ${}^\bullet t$ and $t^\bullet$ in a unique way.

   If t is assigned Firing Rule 2, then m_2 can be found as follows: $m_2(p) = m_1(p) + 1$ if $p \in t^\bullet$, $m_2(p) = m_1(p) - 1$ if $p \in {}^\bullet t \wedge m(p) \geq 1$, and $m_2(p) = m_1(p)$ otherwise.

   Now, since t can only fire if it is enabled, and it is enabled if $\forall p \in {}^\bullet t \backslash ({}^\bullet t)^i : m_1(p) \geq 1$ and $\forall p \in ({}^\bullet t)^i : m_1(p) \leq 0$, the above firing rule for the new marking m_2 can be rewritten as follows: $m_2(p) = m_1(p)+1$ if $p \in t^\bullet$, $m_2(p) = m_1(p)-1$ if $p \in {}^\bullet t \wedge p \in {}^\bullet t \setminus ({}^\bullet t)^i$, and $m_2(p) = m_1(p)$ otherwise.

   Since the set $({}^\bullet t)^i$ is always fixed for each a transition regardless of the marking under which it can fire, t always changes any such marking in the same way. I.e., it decrements and increments its ${}^\bullet t$ and $t^\bullet$ in a unique way.

   Thus, no matter which firing rule is used, any t decrements and increments its ${}^\bullet t$ and $t^\bullet$ in a unique way. Now, by induction on the length of σ, we can rewrite that the canonical firing sequence equation for our CLN $\mathcal{N}$ in a slightly modified form:

$$m'(p) = m(p) + \sum_{t \in {}^\bullet p} [\sigma](t) - \sum_{t \in p^\bullet \backslash (p^\bullet)^i} [\sigma](t),$$

   where $(p^\bullet)^i$ denotes the set of those output transitions for p whose functions contain literal p in the inverted form. Indeed, since set $({}^\bullet t)^i$ is defined uniquely for each $t \in T$, the set $(p^\bullet)^i$ is also unique for each $p \in P$.

   Therefore the new marking m' is determined only by the value of the firing vector and not the order in which transitions fire. The Reachability graph is thus permutable. ∎