A Net-based Formal Framework for Causal Loop Diagrams

Guillermina Cledou and Shin Nakajima

Abstract Causal Loop Diagrams (CLDs) are a modeling tool employed in Business Dynamics. Such a diagram consists of many tightly coupled loops to capture dynamic behavior of systems. Intuitive operational semantics, describing how changes are propagated among the loops, provide a basis for model animation or manual inspection. They are, however, not precise enough to enable automated property checking. This paper proposes and defines a net-based formal framework, allowing true concurrency, so that automated analysis is made possible.

1 Introduction

Smart transportation is a strategic area to focus on enabling sustainable societies. Managing such large-scale complex systems is mandatory, which needs their modeling from various viewpoints. Ontology captures common vocabulary with its focus on structural aspects of domain concepts (e.g. [2]). System Dynamics (SD) approach is elucidating dynamic behavior (e.g. [9]). The two approaches are complementary, and dynamics are often complex and tightly coupled implicitly, leading to unknown deficiencies. Removing vulnerabilities is essential to avoid *normal accidents* [8]. For example, safety analysis in Aerospace engineering [5] is one successful application of SD to study how to prevent disastrous accidents.

Modeling dynamic behavior of systems may start with qualitative causal loop diagrams (CLDs) followed by a quantitative Stock-Flow approach [10]. A CLD illustrates causal links between concepts, and brings out *mental images* of various stakeholders involved, describing system structure and a hypothesis on its dynamic behavior. A modeling goal is representing complex systems, and thus a straight for-

Guillermina Cledou
HASLab INESCTEC & University of Minho, Portugal, e-mail: mgc@inesctec.pt

Shin Nakajima
National Institute of Informatics, Tokyo, Japan, e-mail: nkjm@nii.ac.jp

ward style of constructing a whole CLD is infeasible. Building a CLD is a part of an iterative modeling process, and involves trial-and-error steps, interleaving of description and analysis, before obtaining a satisfactory CLD. Because a CLD is basically representing dynamic behavior, manual inspection is cumbersome as a CLD becomes large. Automated analysis tools can bring great benefits at reduced cost, which is recognized well in modeling of software at an abstract level [3]. Unfortunately, definitions of CLD [10] are not precise enough to enable systematic automated analysis such as model checking [1][7]. This hinders from adapting a tool-supported iterative process of building CLDs.

The present paper proposes a new net-based formal framework, *causal loop net (CLN)*, as a basis of formal representation and analysis of CLD. CLN is inspired by Petri-nets [6], especially CP-nets [4], because of their *true concurrency* characteristics, but introduces notions of abstractions to take into account the qualitative nature of CLD. Main contributions of this paper are introducing a new net-based formal framework CLN, and demonstrating the ideas with a proof-of-concept (PoC) tool written in Scala.

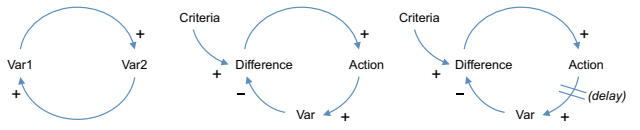
The reminder of the paper is organized as follow. Section 2 and 3 recall CLDs and Petri-nets. Section 4 summarizes issues and our approach to solving them. Section 5 proposes a formal definition of CLN, and then Section 6 discusses ways to specify properties to check. Section 7 demonstrates example cases. Section 8 concludes the paper.

2 Causal Loop Diagrams

2.1 Diagram Representation

Causal loop diagams (CLDs) are a modeling tool to qualitatively represent feedback control systems of non-linear dynamics (Chapter 5 in [10]). A CLD illustrates variables connected via causal links to form loops. Each causal link refers to a fact that a source variable affects a variable at a destination end. Causal links are annotated with either a positive ($+$) or negative ($-$) polarity.

CLD offers three basic components. Figure 1 (a) is a loop in which two connected variables reinforce each other; if `Var1` increases, then `Var2` increases, and



(a) Reinforcing (b) Balancing (c) Balancing with Delay

Fig. 1 Three Basic Components

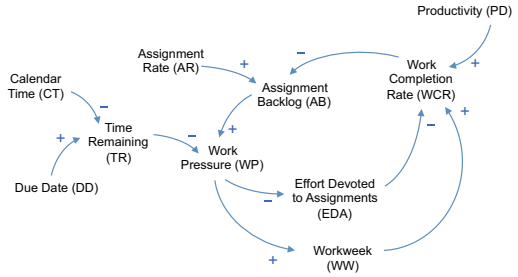


Fig. 2 An Example Composed Loop (Figure 5-21 in [10])

similarly for cases of decreasing. Figure 1 (b) illustrates a situation where two variables `Difference` and `Var` are balancing; if `Var` increases, then `Difference` decreases because of the negative polarity on the link, and similarly for cases of decreasing `Var`. A balancing loop represents a negative feedback. Figure 1 (c) is an example component to have a delay annotation on a causal link between `Action` and `Var`. Changes are not propagated immediately, but deferred, which gives systems inertia, posing much effects on dynamics. These basic components are combined to form large loops to represent complex system dynamics. Figure 2 is an example to have feedback control on a variable `WP` via two loops.

2.2 Informal Semantics

Structural aspects of a CLD are basically captured by a four-tuple $\langle V,\ I,\ \ell,\ \tau\rangle$. V is a finite set of variable names, and I is an initialization relation. ℓ and τ, representing causal links, are transfer relations of $V \times Polarity \times V$ where $Polarity \stackrel{def}{=} \{+,-\}$. ℓ are simple causal links while τ are those with a delay annotation. They are disjoint; $\ell \cap \tau = \emptyset$. Elements of *Polarity* are $sign(\partial v_d / \partial v_s)$ when a causal link directs toward a destination v_d from a source v_s ($v_d, v_s \in V$).

Dynamic behavior of a CLD is a set of possible sequences of *fired* transfer relations. Let S_t be a set of *enabled* transfer relations at a time point t in a global time-line. Enabled transfer relations can be fired to propagate changes in a source variable to a destination. Get such a relation r from S_t. If $r \in \ell$, r is fired immediately to propagate changes according to the above mentioned rules on the polarity. If $r \in \tau$, r is rewritten to an ℓ form, and is added to S_{t+d} for a certain d ($d > 0$). It effectively makes r to be fired after passing d ticks; namely r is *delayed*.

CLD defines a notion of being *fired* in an intuitive manner. Tentatively we may have a naive view of starting with initialized terminal source variables and tracing changes in all the variables along causal links. As a number of causal links together with variables becomes large, inspecting these changes manually is cumbersome. Although, for example, the CLD of Figure 3 in [9] is not large, how the variables

change their values is not intuitively clear. Some automated analysis methods are desirable.

3 Net-based Formal Frameworks

Petri-nets [6] is a net-based formal framework for modeling concurrent systems. A Petri-net is a weighted directed bipartite graph, $\langle P, T, F, W, M_0 \rangle$. P is a finite set of *places*, and T is a finite set of *transitions*, both of which constitute nodes of a graph. F is a flow, a subset of $(P \times T) \cup (T \times P)$, describing edges between nodes. W is a weight function[1] $F \longrightarrow \mathcal{N}$. M_0 is an initial marking discussed below. A place can hold more than one *token* and thus is a multi-set or a bag of tokens. Tokens are indistinguishable with each other. Figure 3 illustrates a simple example Petri-net, actually a Place/Transition net (PT-net), in which $\bullet$ refers to a token.

Dynamic behavior of a PT-net is defined in terms of *markings*. Let $M(p)$ represent a multi-set of tokens at a place p. Informally, a transition t is fired when its input places contain enough number of tokens to enable the transition. Furthermore, more than one transition can be fired at the same time. It is referred to as *true concurrency*, which can be compared with interleaving semantics adapted in most of concurrent computation models.

Figure 3 shows a simple example PT-net[2] shows that an initial marking $M_0 = \{p0 \mapsto \{| \bullet |\}\}$. Transitions of markings, starting from M_0, constitute reachability graphs, which are basis of analyzing dynamical aspects of PT-net. Reachability graphs are graphs of markings, and thus are called *marking graphs* in this paper.

In the example PT-net in Figure 3, starting with M_0, the transitions $t0$ and $t1$ can fire consecutively. A resulting new marking ($M_2 = \{p2 \mapsto \{| \bullet |\}\}$) may enable two transitions $t2$ and $t3$. The two are in *conflict*; namely, only one of them can fire even if both are enabled. Assume that $t3$ is fired, which leads to $M_4 = \{p2 \mapsto \{| \bullet, \bullet |\}\}$ after firing $t4$ whose output edge has a weight annotation of 2. As this M_4 has two tokens, two enabled transitions $t2$ and $t3$ can fire at the same time, showing true concurrency. This results in $M_5 = \{p1 \mapsto \{| \bullet |\},\ p3 \mapsto \{| \bullet |\}\}$. A transition sequence

[1] $\mathcal{N}$ is a set of Natural Number and $\mathcal{N}_0$ includes 0.

[2] This is an infinite capacity net, in which each place can hold any number of tokens.

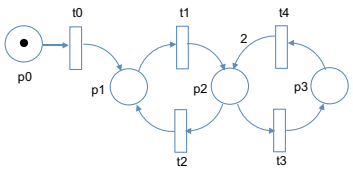


Fig. 3 Petri Net

may continue to a situation where $M_9 = \{p2 \mapsto \{| \bullet, \bullet, \bullet |\}\}$. The number of tokens is increasing. This PT-net is unbounded, and its reachability graphs are infinite.

Apart from the basic PT-net, various high-level Petri-nets have been proposed so far, including Coloured Petri Nets (or CP-nets) [4]. CP-nets can work on colored tokens. Tokens with different colors are distinguishable, and transitions may have guard conditions on token colors; only tokens with a certain color can enable such transitions. Intuitively, a marking in CP-nets M is partitioned into color-indexed markings M^c. Occurrence graphs of CP-nets are defined similar to reachability graphs of PT-nets.

Note that analysis methods using either reachability graphs or matrix-equations are studied for PT-nets [6]. Flow of tokens can be represented uniformly with matrix-equations, because tokens are indistinguishable. However, analyzing CP-nets is possible only with occurrence graphs [4] because tokens with different colors are distinct.

4 Abstractions Qualitatively

CLD represents system dynamics qualitatively, and introducing qualitative abstractions is one of key issues. We will study abstractions from three viewpoints.

4.1 Qualitative Values

As presented in Section 2, variables do not take values, but represent qualitatively an increase or a decrease. We, however, introduce a notion of values that a variable can take, to make defining operational semantics easy.

Firstly, we view that a variable v_X in CLD may stand for δX of an accompanying hypothetical variable X. Secondly, we introduce a set Q to be $\{up,\ down\}$, and an extension $\hat{\mathrm{Q}} \stackrel{def}{=} Q \cup \{none\}$. The values up and $down$ refer to an increase and a decrease respectively, and *none* stands for being unknown or no change. A function *rev* , of $\hat{\mathrm{Q}} \longrightarrow \hat{\mathrm{Q}}$, is defined such that $rev(up)$=$down$, $rev(down) = up$, and $rev(none) = none$.

For a variable $v \in V$, $[\![\ \_\ ]\!]$ is a dereference function, of $V \longrightarrow \hat{\mathrm{Q}}$. An element in an initialization I specifies an initial value of a variable v_i ($[\![v_i]\!] \in Q$). Variables not appeared in I take *none* as their initial values. Transfer relations specify how a value in a source v_s is transferred to a destination v_d, and an annotated polarity affects the transferred values. With $[\![v_s]\!] \in Q$, the equations below illustrates how the polarity acts in the transfer.

$$[\![\ v_d\ ]\!] = [\![\ v_s\ ]\!] \quad \textbf{if } \ell(v_s, +, v_d), \qquad [\![\ v_d\ ]\!] = rev([\![\ v_s\ ]\!]) \quad \textbf{if } \ell(v_s, -, v_d)$$

Each element of $\hat{\mathrm{Q}}$ may be encoded as a distinct color if we use CP-nets as a formal framework to encode CLDs. CLD may be encoded as a kind of CP-nets.

4.2 True Concurrency and AMAN Strategy

As presented in Section 3, Petri-nets, either CP-nets or PT-nets, can represent true concurrency in a faithful manner. However, the example scenario of Figure 3 illustrates a situation that places may have an infinite number of tokens, which is clearly not desirable from a view point of automated analysis methods. This infiniteness comes from characteristics of tokens in Petri-nets.

Tokens in Petri-nets can have more than one role. Firstly, a token is representing a computation thread, and thus multiple tokens can refer to multi-threaded computations. Secondly, a token is a single computing resource, and moving a token from a place to another illustrates a situation where an existing resource is consumed at a source place and a new one is produced at a destination.

CLD is abstract, and does not have any notion of computing resources, but its true concurrency aspects are essential. This observation leads to an idea that we duplicate tokens as many as needed, which might be called an *AMAN strategy* in this paper, so as to satisfy desirable degrees of concurrency. While a place in Petri-nets is a multi-set of tokens, we use a set of tokens instead. In addition, we adapt an AMAN strategy to duplicate tokens in a place (p) to fire all transitions connected outwards from p. Note that with an AMAN strategy, no *conflict* occurs (cf. Figure 3). We abstract multi-sets of tokens to be sets of tokens, which ignores the computing resource aspects of tokens. CLN is different from CP-nets.

4.3 Non-deterministic Delay

Delay is quantitative in nature (Chapter 11 in [10]). A delayed causal link in qualitative CLD is an abstraction of *stock* of quantitative Stock-Flow models. Introducing a delay distinguishes an event of writing values to a stock from another event of reading values. Because these two events are distinct, not occurred at the same time, CLD adapts an abstraction such that transfer values is *delayed*, but a delay is not accompanied with any quantitative amount of time.

Delay makes much effects on a possible transition sequences, and thus on dynamic behavior of CLD [10]. Intuitively, when a particular causal link t_d is annotated with delay (τ), the link is not fired immediately even if true concurrency semantics allow firing the transition transfer of t_d. Now, compare two cases where a particular transition t is in τ (delay) and it is in ℓ (simple). Possible transition sequences of the second simple case is different from the first. Firing t, in the first delay case, appears later in transition sequences than the second. Furthermore, such transition sequences involving delayed t_d might be different for different quantitative amount of delay times. As CLD does not refer to quantitative values, we are not able to specify particular situations selectively.

We introduce a notion of non-deterministic delay time as an abstraction. Given an arbitrary upper bound $d^{upper} \in \mathcal{N}$, we choose a delay time d ($d < d^{upper}$) in a non-deterministic manner when constructing marking graphs. Marking graphs ob-

tained in this way are different for different chosen values of d. Covering all possible cases is impossible, and thus their dynamic behavior is under approximation, namely searching in a part of all possible sequences only.

5 Causal Loop Nets

Causal loop net (CLN) is a directed bipartite graph that we propose in this paper as a formal framework of CLD. CLN is structurally isomorphic to CLD (Figure 4), in which places of CLN correspond to variables of CLD. We may consider CLN as an internal representation of CLD; modelers need not care about CLN.

5.1 Formal Definitions

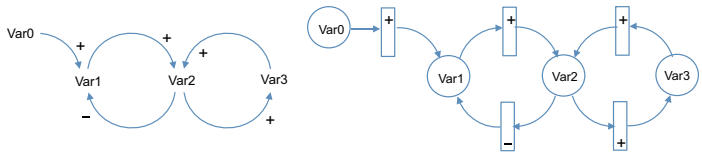
A CLN is a six-tuple D over $Token$, $D = \langle P, T, F, \tau, \nu, M_0 \rangle$. P and T are finite sets of places and transitions respectively, and $F \subset (P \times T) \cup (T \times P)$. τ is a polarity annotation, $T \longrightarrow \{+, -\}$. ν is a delay annotation, $T \longrightarrow \mathcal{N}_0$. M_0 stands for an initial marking, which will be explained below. For a transition t, a set of input places ${}^\bullet t$ and a set of output places $t^\bullet$ are defined.

$${}^\bullet t = \{\ p \mid (p,t) \in F\ \}, \qquad t^\bullet = \{\ p \mid (t,p) \in F\ \}$$

Tokens in CLD are either basic tokens or delay tokens; $BasicToken = \{\uparrow, \downarrow\}$ and $DelayToken = \{\uparrow_d, \downarrow_d\}$ for $d \in \mathcal{N}$, where $\uparrow$ is *up* and $\downarrow$ is *down*. We introduce a whole set of tokens $Token$ to be $BasicToken \cup DelayToken \cup \{\_\}$, where $\_$ is *none*. d in $\uparrow_d$ or $\downarrow_d$ refers to an amount of delay if $d > 0$. For a technical reason, we assume the relationships that $x_0 = x$ for a token x; namely, $\uparrow_0 = \uparrow$ and $\downarrow_0 = \downarrow$. Intuitively, a delay token x_d decreases its amount one as ticks proceed, reaching x_0 to contribute enabling of transitions because $x_0 = x$. Now, $Token$ can be simplified to be $\{\uparrow_e, \downarrow_e, \_\}$ for $e \in \mathcal{N}_0$. A marking M is $P \nrightarrow 2^{Token}$.

Given a marking M, a transition t is enabled if its all input places have tokens other than $\_$ (*none*). Thus, a set of enabled transition is

$$\mathcal{T}(M) = \{\ t \in T \mid \bigwedge_{p \in {}^\bullet t} M(p) \cap \{\uparrow_e, \downarrow_e\} \neq \emptyset\ \}$$



(a) CLD Example (b) Translated CLN

Fig. 4 CDL and CLN

$\mathscr{T}(M)$ may have more than one transition, which ensures that CLN is a formal framework allowing *true concurrency*.

5.2 Dynamic Behavior

Dynamic behavior of a CLN is captured by an associated marking graph, which changes in markings ($M \stackrel{\mathscr{T}(M)}{\longrightarrow} M'$) define. Given a set of enabled transitions $\mathscr{T}(M)$, $M'(p)$ for all places p in P is defined below, $\bigoplus$ denoting *overriding*.

$$M'(p) \;=\; M(p) \bigoplus (\bigcup_{t\in\mathscr{T}(M)} {}^{\bullet}\Delta(p,t) \;\cup\; \bigcup_{t\in\mathscr{T}(M)} \Delta^{\bullet}(t,p))$$

where ${}^{\bullet}\Delta$ and $\Delta^{\bullet}$ are of $F \longrightarrow 2^{Token}$. Recall that $F \subset (P\times T)\cup(T\times P)$.

${}^{\bullet}\Delta(p,t)$ removes all the tokens in $M(p)$ and leaves delay tokens whose residual time is decremented by one.

$$ {}^{\bullet}\Delta(p,t) \;=\; \bigcup_{x\in M(p)} Decr(x)$$

$Decr(x)$ makes use of the relation that $\uparrow = \uparrow_0$ and $\downarrow = \downarrow_0$.

$$Decr(x_e):\; Token \longrightarrow 2^{Token} = \begin{cases} \{\; x_{e-1} \;\} & \textbf{if} \quad e > 0 \\ \emptyset & \textbf{if} \quad e{=}0 \end{cases}$$

$\Delta^{\bullet}(t,p)$ adds tokens calculated according to the polarity $\tau(t)$, where $M({}^{\bullet}t)$ is an abbreviation of $\bigcup_{p'\in{}^{\bullet}t} M(p')$.

$$\Delta^{\bullet}(t,p) \;=\; \bigcup_{x\in M({}^{\bullet}t)} \mathrm{Xfer}(x)$$

$\mathrm{Xfer}(x)$ assumes, for simplicity, that $\nu(t) = 0$ if a causal link is not marked *delay*.

$$\mathrm{Xfer}(x):\; Token \longrightarrow 2^{Token} = \begin{cases} \{\; x_{\nu(t)} \;\} & \textbf{if } \tau(t) = + \\ \{\; rev(x_{\nu(t)}) \;\} & \textbf{if } \tau(t) = - \end{cases}$$

The transition relationship $M \stackrel{\mathscr{T}(M)}{\longrightarrow} M'$ may lead to a situation where, for a certain p, $\{p \mapsto \uparrow,\; p \mapsto \downarrow\} \in M'$ occurs. Such a marking is *inconsistent* in that it assigns two tokens $\uparrow$ and $\downarrow$ to a place p at the same time. P_{NG} is a subset of P to contain those inconsistent places; $P_{NG} \stackrel{def}{=} \{\; p \mid \{\uparrow,\downarrow\} \subseteq M(p) \;\}$.

The inconsistency demonstrates a situation where both cases, p with $\uparrow$ and p with $\downarrow$ are possible for $\forall p{\in}P_{NG}$. Therefore, two cases are to be explored non-deterministically[3]. An auxiliary function Nr constructs consistent markings by dividing an inconsistent marking into two cases.

[3] This makes it difficult to define matrix-equations for CLN.

$$Nr(M) \;=\; \begin{cases} Nr(M[p \mapsto D(p,\uparrow)]) \cup\; Nr(M[p \mapsto D(p,\downarrow)]) & \textbf{if}\;\; p \in P_{NG} \\ \{\, M \,\} & \textbf{if}\;\; P_{NG} \;=\; \emptyset \end{cases}$$

where $D(p,x) = (M(p) \cap \{\uparrow_d, \downarrow_d\}) \cup \{x\} (d{>}0)$. Now that $Post(M)$ is a set of consistent markings reached directly from M when inconsistent markings are considered.

$$Post(M) \;=\; \{\, M' \mid M \stackrel{\mathscr{T}(M)}{\Longrightarrow} M'' \;\wedge\; M' \in Nr(M'') \}$$

Given a CLD D, a marking graph $\mathscr{G}^D$ of a CLN is a tuple $\langle Node, n_0, Edge \rangle$, where n_0 is an initial node representing the initial marking M_0, $Edge$ is a set of transition relations $M \Longrightarrow M'$ for $M' \in Post(M)$, and $Node$ consists of all markings reached from M_0 via $Edge$. $\mathscr{G}^D$ is finite in particular.

6 Formal Analysis

Once a marking graph is obtained for a given CLN or CLD equivalently, we can conduct formal analyses to study the CLN from various ways.

6.1 Verification Problems

As explained before, a place of CLN represents a variable in CLD. Since a node n in $\mathscr{G}^D$ is a marking, $[\![n.v]\!]$, a value of variable v at node n, is $M(v)$ and $[\![n.v]\!] \in \hat{\mathrm{Q}}$. Delay tokens, $\uparrow_d$ or $\downarrow_d$, are identified with _ (*none*) as values, because they are transient and do not any definite values.

Let $Trace^D$ be a set of all possible transition sequences σ_i generated by $\mathscr{G}^D$ starting from n_0; $Trace^D = \{\; \sigma_i \;\}$. $\sigma_i(r)$ is an r-th node in σ_i, and $\sigma_i(r..s)$ is a transition sequence starting with $\sigma_i(r)$ and ending with $\sigma_i(s)$. In particular, $\sigma_i(0)$ is n_0 for any sequence.

A verification problem is to check whether at least one transition sequence exists in $Trace^D$ to satisfy a given property Φ taking into account simulation relations $\sim$.

$$\exists \sigma_i \in Trace^D :\; \Phi(\sigma_i) \;\;\textbf{mod}\;\; \sim$$

The simulation relation $\sim$ is either $\sim_1$ or $\sim_2$ below.

$$\begin{array}{ll} & \sim_1 : \hat{\mathrm{Q}}^N \longleftrightarrow Q \;\text{ is }\; q(q|none)^{N-1} \sim_1 q \text{ for } q \in Q, \\ \text{and} & \sim_2 : \hat{\mathrm{Q}}^N \longleftrightarrow \hat{\mathrm{Q}} \;\text{ is }\; q^N \sim_2 q \qquad\qquad \text{for } q \in \hat{\mathrm{Q}}. \end{array}$$

Adapting simulation relations makes it easy to write properties to be checked. Transition sequences σ_i are of $seq(\hat{\mathrm{Q}})$. Because $q \in \hat{\mathrm{Q}}$ does not refer to a quantitative, concrete value, identifying N consecutive q (q^N) with just a q does not make much

effects on qualitative characteristics of sequences. Namely, q can be a summary of q^N. Simulation relation of the first type ($\sim_1$) provides a way to use Q as a basis for this summary, while the second ($\sim_2$) is using $\hat{\mathrm{Q}}$ instead. We will see how two simulation relations are effective in Section 7.

6.2 Some Query Patterns

We introduce a few typical properties to be checked, all of which take a form of queries on $Trace^D$. A finite mapping f of $[0,k-1] \longrightarrow Q$ for a $k \in \mathcal{N}$ is called *a summary function* when f describes desired sequences of a CLD variable. A predicate $\varphi_0(i,r,v,f)$ becomes *true* if there is at least one transition sequence σ_i in $Trace^D$ whose subsequence starting with r and ending with s satisfies the condition that a sequence of values generated by a variable v in D is simulated by a given summary function f.

$$\varphi_0(i,r,v,f) = \bigwedge_{j=0}^{k-1} \exists s_{j+1}.\ (\sigma_i(s_j \,..\, s_{j+1}).v \sim f(j))$$

where $k = |f|$ (length of f), $r = s_0$ and $s = s_k$.

Some properties employ $\varphi_0(i,r,v,f)$ to define themselves. Firstly, $\varphi_1(v,f)$ returns *true* if there is a transition sequence σ_i to have a subsequence of a certain length in which the specified variable v is simulated by a given summary function f.

$$\varphi_1(v,f) \ = \ \exists i,r:\ \varphi_0(i,r,v,f)$$

Secondly, $\Phi_3(\varphi_1(v^m,f),v^\ell,g^\ell)$ is *true* if a summary function of a variable v^ℓ ($m \neq \ell$) is simulated by g^ℓ in a transition sequence that $\varphi_1(v^m,f)$ satisfies;

$$g^\ell(X) \ = \ \exists s.\ \sigma_i(r \,..\, s).v^\ell.$$

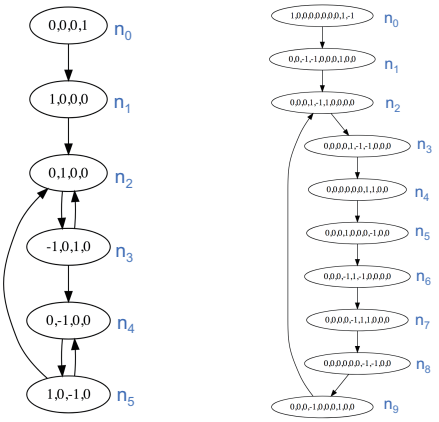
Intuitively, Φ_3 extracts a sequence of v^ℓ values along a transition sequence that v^m satisfies f. Therefore, we can compare how two variables, v^m and v^ℓ, change their values in an obtained interval of r and s.

7 Examples

Figure 5 shows two marking graph examples generated with a Scala-based PoC tool using Graphviz[4] for preparing graphical images. Figure 5(a) is the one for the CLD in Figure 4. Figure 5(b) is simple because of no inconsistent marking ($P_{NG} = \emptyset$). In these graphs, `1`, `0` and `-1` stands for $\uparrow$, _ and $\downarrow$ respectively.

Each node in the graph of Figure 5(a) refers to variable values in a form of $\langle$`Var1`, `Var2`, `Var3`, `Var0`$\rangle$. The graph is the one starting with an initial marking of $M_0 \ = \ \{\texttt{Var0} \mapsto \uparrow\}$. The prototype PoC tool accepts a CLD in a textual form. The CLD in Figure 4 is, for example, entered as below.

[4] http://viz-js.com



(a) CLD in Figure 4 (b) CLD in Figure 2

Fig. 5 Marking Graphs

```
var netF4 = newclpn ++ (
  4 |>+ 1,
  1 |>+ 2,
  2 |>- 1,
  2 |>+ 3,
  3 |>+ 2
) initMark((4, Set(1))
```

The numbers are representing CLD variables. `4` stands for `Var0` whose initial value is `1` (namely ↑) as specified by `initMark((4, Set(1))`.

We check whether $\mathscr{G}^D$ satisfies $\varphi_1(\texttt{Var2}, f^{\text{Var2}})$ modulo $\sim_1$ where $f^{\text{Var2}} = (\uparrow\uparrow)$. For this, $\varphi_1(\texttt{Var2}, f^{\text{Var2}})$ is $false$ because the graph does not have any sequence with consecutive ↑s. Alternatively, if we choose f^{Var2} to be $(\downarrow\uparrow)$, we actually find transition sequences $\sigma_i(5\,..)$, some of which are shown here. `Var3` changes its value along each transition sequence.

$$\begin{array}{ll}
\sigma_1(5\,..\,8) = n_4\ n_5\ n_2\ n_3 & (\sigma_1(5..8)).v^{\text{Var3}} = _\downarrow_\uparrow \\
\sigma_2(5\,..\,10) = n_4\ n_5\ n_4\ n_5\ n_2\ n_3 & (\sigma_2(5..10)).v^{\text{Var3}} = _\downarrow_\downarrow_\uparrow \\
\sigma_3(5\,..\,10) = n_4\ n_5\ n_2\ n_3\ n_2\ n_3 & (\sigma_3(5..10)).v^{\text{Var3}} = _\downarrow_\uparrow_\uparrow \\
\sigma_4(5\,..\,12) = n_4\ n_5\ n_4\ n_5\ n_2\ n_3\ n_2\ n_3 & (\sigma_4(5..12)).v^{\text{Var3}} = _\downarrow_\downarrow_\uparrow_\uparrow
\end{array}$$

A summary function g^{Var3} modulo $\sim_1$ for all the above will be $(\downarrow\uparrow)$, which provides little information. On the other hand, if we use $\sim_2$, the summary function g^{Var3} becomes $((\_\downarrow)^{n1}(\_\uparrow)^{n2})$ for $n1,n2\in[1,\,2]$. This scenario illustrates that the $\sim_1$ is appropriate when we extract transition sequences by means of f^{Var2}, and that the $\sim_2$ is useful in searching for summary functions of g^{Var3} because the obtained sequences has detailed information. We may choose g^{Var3} as $(\_\downarrow)^{n1}(\_\uparrow)^{n2}$ for $\Phi_3(\varphi_1(\texttt{Var2}, (\downarrow\uparrow)), \texttt{Var3}, g^{\text{Var3}})$ to be satisfied modulo $\sim_2$.

8 Concluding Remarks

Petri-nets have a large body of work [6], from introducing high level Petri-nets such as CP-net, to studying subclasses of PT-nets in view of behavioral or structural characteristics. Causal Loop Net (CLN) is unique in that our approach is introducing qualitative abstractions into a Petri-net family of formal frameworks. CLN provides a formal basis of Causal Loop Diagrams (CLD) to enable formal analysis in terms of marking graphs. Query patterns, although not general enough, are useful in tool-assisted inspections as demonstrated with our PoC tool. The proposed CLN enables adapting a tool-supported iterative process of building CLDs.

Our future plan includes developing a robust tool that can work on large CLDs, studying pros and cons of the proposed abstraction method for delay, and adapting logic model checking methods [1] for formal analyses.

Acknowledgements

The first author conducted the reported work at NII under NII-Internship Program 2017-1 call. This work is a result of project "SmartEGOV/NORTE-01-0145-FEDER-000037", supported by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (EFDR). Additional support is provided by the PT-FLAD Chair on Smart Cities & Smart Governance. The second author is partially supported by JSPS KAKENHI Grant Number JP17H01726.

References

1. E.M. Clarke, O. Grumberg, and D.A. Peled : *Model Checking*, The MIT Press, 1999.
2. G. Cledou and L. Barbosa : An Ontology for Licensing Public Transport Services, In *Proc. 9th ICEGOV*, pp.230-239, 2016.
3. D. Jackson and J. Wing : Lightweight Formal Methods, *IEEE Computer*, 29(4), pp.21-22, 1996.
4. K. Jensen : *Coloured Petri Net*, Springer 1996.
5. N.G. Leveson : *Engineering a Safer World: Systems Thinking Applied to Safety*, The MIT Press 2011.
6. T. Murata : Petri Nets: Properties, Analysis and Applications, In *Proc. IEEE*, 77(4), pp.541-580, 1989.
7. S. Nakajima : Model Checking of Energy Consumption Behavior, In *Proc. 1st CSD&M-Asia*, pp.3-14, 2014.
8. C. Perrow : *Normal Accidents: Living with High-Risk Technologies*, Princeton University Press 1999.
9. S.P. Shepherd : A Review of System Dynamics Models Applied in Transportation, *Transportmetrica B: Transport Dynamics*, 2(2), pp.83-105, 2014.
10. J.D. Sterman : *Business Dynamics: Systems Thinking and Modeling for a Complex World*, Irwin McGraw-Hill 2000.