Article

Not peer-reviewed version

Analysis of Concurrent Systems Based on Interval Order

Yang Xu *, YE Chen *, Chen Yi Jun *

Posted Date: 2 May 2023

doi: 10.20944/preprints202305.0049.v1

Keywords: concurrent systems; Mazurkiewicz traces; interval order; Petri net with Data

Article

Analysis of Concurrent Systems Based on Interval Order

Xu Yang [1,2,*,†], Chen Ye [1,2,†] and CHEN Yijun [3,†]

[1] Department of Computer Science and Technology, College of Electronic and Information, Tongji University, Shanghai 201804
[2] The Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 201804
[3] National Maglev Transportation Engineering R&D Center, Tongji University, Shanghai,201804, China
* Correspondence: 1610482@ tongji.edu.cn, yechen@tongji.edu.cn, chenyijun@tongji.edu.cn; Tel.: +86-186-1681-5297 (X.Y.)
† These authors contributed equally to this work.

Abstract: One way to verify the concurrent system is to sort the read and write events and control events in the concurrent system according to the sequence relationship, either in a partial order, or in a strict order. Due to the concurrency of the events and the timing of the written data events, there is an overlap between the events, which requires an interval order for analysis. The order structure of the entire interval order is represented by the start and end sequences, and the order is observed for all equivalent interval orders. Mazurkiewicz traces, Comtraces are hierarchical traces, but neither can describe the interval trace. The petri interval trace with data is the technique to solve the problem in this paper, DPN, the Petri net with data, to describe the interval trace of read and write data. The method used is a BE (Beginnings and Endings) sequence to represent a class of runs based on the interval sequence. The case multi-threaded system represents the interval of writing events, and according to the unfolding of Petri net, the analysis of the running trace involved by read and write data events is successfully analyzed.

Keywords: concurrent systems; mazurkiewicz traces; interval order; style; Petri net with data

0. Introduction

In concurrent systems, most behavioral semantics are defined according to total order or stratified step order. If the causal relationship is considered, concurrent history is partial order [1–3] or Mazurkiewicz trace [4]. The Mazurkiewicz trace allows an equivalent single sequence, to represent the entire partial order. Traces provide a simple link between the observational and process semantics of concurrency systems. Another trace Comtraces represents the hierarchical sequence structure by a single step sequence, and its related observations can be obtained as a hierarchical or interval extension of the appropriate partial order [5,6]. A sequence of events that contains a causal relationship needs to be represented by a step-by-step sequence, or by a structure using a hierarchical order. An interval order structure is required when modeling the relationship, or when the observation is an interval order. Other relevant observations can be obtained as stratified or interval expansion of appropriate partial order. All observations of causality [7,8] are step sequences, which use stratified order structures. When modeling with the "no later than" relationship or all observations are interval order [9], the interval order structure should be employed to describe the observations.

When studying the behavior patterns of the concurrent system, the read and write data events in the concurrent system are sorted according to the sequence relationship, either in partial orders [10] or in total orders. Because of the concurrency of events [11–13] and the time required to writing events, there will be overlap between events, which requires interval order analysis. For most concurrency models, it is problematic to directly generate interval order. A very good way to represent sequences with feasible interval order is to use the beginnings and endings of events involved.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

On a monoid-based model, the model will allow a sequence containing the beginning and end of written data events to represent the entire hierarchical order structure, and all equivalent observable interval orders. This completes the write data events by introducing the concept of an interval trace that combines the idea of Mazurkiewicz traces and comtrace, and proves that converting the write events into each interval trace at the beginning and end, uniquely identifies the interval order structure.

The impact of data on control flow cannot be described in ordinary Petri nets [14] or Petri nets with read arcs [10] or inhibitor arcs [15] for concurrent systems. This paper uses Petri nets with data, namely DPN, to describe the data participation in the modeling and simulation of concurrent systems [16–20]. This paper will also show how interval write data traces describe the interval orders semantics of Petri nets with data, so as to analyze the operation of concurrent systems. Through the analysis of the results, the interval trace is applied to the concurrent system with data Petri net description.

The main contributions are as follows:

1.Using Petri nets with data, namely DPN, to describe the data participation in the modeling and simulation of concurrent systems. 2.A BE (Beginnings and Endings) sequence representing an equivalent class of runs, or targets. The goal is to have a BE sequence to represent the entire hierarchical order structure, that is, all equivalent observable interval orders. 3.How interval write data traces describe the interval orders semantics of Petri nets with data, so as to analyze the operation of concurrent systems. Through the analysis of the results, the interval trace is applied to the concurrent system with data Petri net description.

The rest of this paper is organized as follows. Session II to III discuss the relevant concepts and modeling model work. Session II provides some basic mathematical knowledge about partial order, complete order, hierarchical order and interval order, as well as Mazurkiewicz traces and Comtraces. Then, the Petri net and the Petri network with the data are introduced in Session III. In Session Iv, the addition of beginnings and endings of writing events in DPN for description, as well as experiments performed in netDraw. In Session V, the application of the interval trajectory is discussed, and the interval trajectory is analyzed to effectively represent the abstract interval order semantics of DPN. In Session VI, we present the running results of the interval of the case, indicating that the proposed interval sequence correctly describes the analysis of the control process based on the interval order.

1. Mathematical Foundations

We will introduce some relevant mathematical symbols and concepts such as partial order, hierarchical order, interval order, Mazurkiewicz trace and Comtraces and etc.

1.1. Partial orders,Stratified order and Interval order

Sort description of events running in a concurrent system, partial order is one of the tools to describe the relationship between events in a concurrent system. Will be used as a complete representation of events running in the concurrent system and a partial representation of concurrent events. There are also hierarchical sequences, complete sequences, and interval sequences, which can all sequence the events. Figure 1 presents the time-based orders. The definitions of different orders are presented as follows.

Definition 1 (Partial Orders)**.** *The relation* $<\subseteq E \times E$ *is a (strict) partial order if it is transitive and irreflexive, i.e., for all* e_1,e_2,$e_3 \in E$, $e_1 < e_2 < e_3 \Rightarrow e_1 < e_3$. *and:*

1) $e_1 \frown\!< e_2 \overset{df}{\Leftrightarrow} \neg(e_1 < e_2) \wedge \neg(e_2 < e_1) \wedge e_1 \neq e_2$,

2) $e_1 <\!\frown e_2 \overset{df}{\Leftrightarrow} e_1 < e_2 \vee e_1 \frown\!< e_2$

Note that $e_1 \frown\!< e_2$ *means* e_1 *and* e_2 *are incomparable (w.r.t.*$<$*) elements E.*

Definition 2 (Total order)**.** *For a relation* $R_1 \subseteq E \times E$, *any relation* $R_2 \subseteq E \times E$ *is an extension of* R_1 *if* $R_1 \subseteq R_2$. *we define Total* $(<) \overset{df}{\Leftrightarrow} \{\lhd \subseteq E \times E\}$, *where* $\lhd$ *is a total order and* $< \subseteq \lhd$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

Definition 3 (Stratified orders). *Stratified orders are often defined in an alternative way,i.e., a partial order* $<$ *on* E *is stratified iff there exists a total order* $\lhd$ *on some n and a mapping* $O : m \to n$ *such that* $\forall m, n \in E$. $m < n \Leftrightarrow \phi(m) \lhd \phi(n)$. *This definition is illustrated in Figure 1, where* $\phi(1) = \{1\}$, $\phi(2) = \phi\{3\} = \{2,3\}$ *and* $\phi(4) = \{4\}$.

Theorem 1. *(Fishburn [1970]) A partial order* $<$ *on E is interval if and only if there exists a total order* $\lhd$ *on some W, and two mappings* $B, E : y \to W$ *such that for all* $a, b \in E$:

1) $B(a) < E(b)$,

2) $a < b \Leftrightarrow E(a) \lhd B(b)$.

Usually $B(a)$ *is interpreted as the beginning and* $E(a)$ *as the end of an interval a (Such events/actions consume time). The intuition of Fishburn's theorem is illustrated in Figure 1 with* $<_3$ *and* $\lhd_3$. *For all* $a, b \in \{1,2,3,4\}$, *we have* $B(a) \lhd_3 E(a)$ *and* $a <_3 b \Leftrightarrow E(a) \lhd_3 B(b)$. *For better readability in the future we will skip parentheses in* $B(a)$ *and* $E(a)$.

In summary, we can conclude that:

1) $<$ is total if $\frown < = O$, In other words, for all $e_1, e_2 \in X, e_1 > e_2 \vee e_2 < e_1 \vee e_1 = e_2$. For clarity, we will reserve the symbol $\lhd$ to denote total orders;

2) $<$ is stratified if $e_1 - < e_2 \frown < e_3 \Rightarrow e_1 \frown < e_3 \vee e_1 = e_3$, i.e., the relation $\frown < \bigcup id_X$ is an equivalence relation on X;

3) $<$ is interval if for all $a, b, c, d \in E, e_1 < e_3 \wedge e_2 < e_4 \Rightarrow e_1 < e_4 \vee e_2 < e_3$.

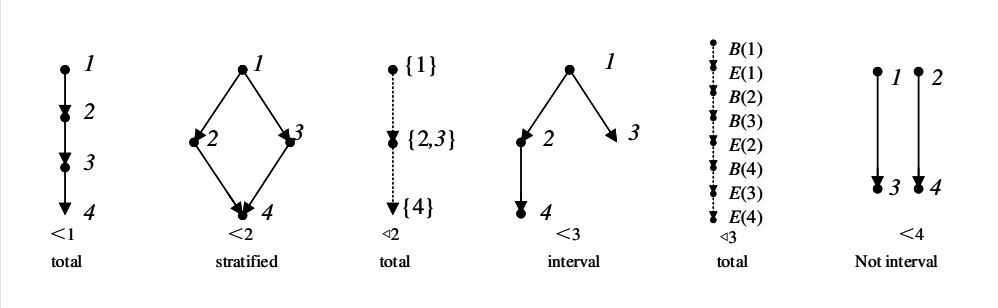

Figure 1. Time-based orders. [5]

1.2. Mazurkiewicz Traces

In 1970, Anthony Mazurkiewicz first proposed the trace theory [21–27] due to the generation, which was also driven by Petri Nets [28–37] and the formal language with automata. The aim is to circumvent some of the problems in the theory of concurrent computation, initially, the most popular way to deal with concurrency is interweaving. Includes interleaved and non-deterministic selection issues regarding process computational refinement. Mazurkiewicz Traces, often abbreviated as traces, correspond to a sequence of atomic actions of concurrent systems and are used to model the execution. The concurrent system can then be described by the execution set, called the language in the framework. Currently, the concept of a trace [38] is usually used to describe the non-sequential behavior of a concurrent system through its sequential observation. In this approach, concurrency is replaced by non-deterministic, where concurrent execution of actions is considered as an uncertain choice of the order of execution of these actions. The trace always represents a concurrent process in a sequence associated with a string.

For a concurrent system, the different actions executed are not formed in a linear order, but the events are arranged in partial order. Therefore, the sequential relationship of the execution events of the concurrent system is defined as the partial order.

Definition 4 (Partia1 order set, poset). *Let* $(Ev, \leq, \lambda)$ *be a poset where* Ev *is countable.*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

For $e \in$ Ev, we define $\downarrow e = \{e_1 \in Ev \mid e_1 \leq e\}$ and $\uparrow e = \{e_1 \in Ev \mid e \leq e_1\}$. We call $\downarrow e$ the history of the event e and $\uparrow e$ the future of the event e.

Let $\lessdot$ be the covering relation given by $e_1 < e_2$ if $e_1 \leq e_2$, $e_1 \neq e_2$, and for all $e_3 \in Ev$, $e_1 \leq e_3 \leq e_2$ implies $e_1 = e_3$ or $e_3 = e_2$.

Moreover, let the concurrency relation (co) be defined as e_1 *co* e_2 *iff $e_1 \not\leq e_2$ and $e_2 \not\leq e_1$.*

The definition of the Mazurkiewicz trace is given below

Definition 5 (Mazurkiewicz trace)**.** *A Mazurkiewicz trace over the independence alphabet $(\Sigma x, In)$ is a Σ-labeled poset $T = (Ev, \leq, \lambda)$ satisfying:*

1) For $\forall\, e_1 \in Ev$, $\downarrow e_1$ is a finite set.

2) For $\forall e_1, e_2 \in Ev, e_1 \lessdot e_2$ implies $\lambda(e_1) \preceq \lambda(e_2)$, where $\preceq$ refers to dependence relation.

3) For $\forall e_1, e_2 \in Ev, \lambda(e_1) \preceq \lambda(e_2)$ implies $e_1 \leq e_2$ or $e_2 \leq e_1$.

The monoid of Mazurkiewicz traces are on the sequence, equivalent monoid. The application of traces in concurrency theory stems from the fact that traces are sequential representations of partial order. The theory of the traces has been used to solve problems from a number of different fields, which enables the traces to model the "true concurrency" semantics. Including combinatorial theory, graph theory, algebra theory, logic, and especially concurrency theory.

2. The proposed model

Multi-threaded concurrent system is multi-threaded in a multi-core processor, in a certain storage mode, based on the shared memory, under the operation process. Before analyzing the operation of multi-threaded system, give the semantic rules of multi-threaded system, the storage rules of running storage mode, the memory access rules and data transformation rules.

2.1. Multithreaded system

Here presents the syntax of concurrent programming language.

$a, b \in$ Values $oX \in$ Mod$X$ where $X \in \{R, W\}$ Access modes

$r, s, t \in$ Loc $\subseteq \{\mathrm{r}, \mathrm{s}, \ldots\}$ Locations sPr $\in$ SProg $\triangleq \{0, 1, \ldots, N\} \rightarrow$ Inst Sequential programs

$x \in$ Reg $\subseteq \{x, y, \ldots\}$ Registers Pr : Tid $\rightarrow$ SProg (Concurrent) programs

$\tau, \pi \in$ Tid $\subseteq \{T_1, T_2, \ldots\}$ Thread identifiers

$e ::= r|v|e + e|e = e|e \neq e \mid \ldots$

Inst $\ni$ inst $::= r := e \mid$ if e goto pc1, $\ldots$, pc$n \mid$ assert(e)

$|x.store(e, o\ \mathrm{W})|r := x \cdot \mathrm{load}(o\mathrm{R})$

A finite set Loc of memory location; a finite set Reg of (local) register; a finite set Val of value; a finite set TId of thread identifier;

Owicki Gries inference logic is used for concurrency. OG inference extends Hoare's proof rule logic and rules to infer the concurrent program form $C_1 \,||\, C_2$, allowing the combination of verified programs C_1 and C_2 into a verified concurrent program, provided that the two C1 and C2 proofs do not interfere with each other:

READ:

$$\frac{P_1 \Rightarrow P_2\,[x_\mathrm{v}/\mathrm{a}]}{\langle \{P_1, P_2\}, \varnothing \rangle \vdash \{P_1\}\mathrm{a} := x\{P_1\}.}$$

WRITE:

$$\frac{P_1 \Rightarrow P_1\,[\mathrm{e}/x_\mathrm{v}]}{\langle \{P_1, Q_1\}, \{\langle x_\mathrm{v}, \mathrm{e}, P_1 \rangle\} \rangle \vdash \{P_1\}x := \mathrm{e}\{P_1\}}$$

SEQ:

$$\frac{\langle \mathcal{R}_2; \mathcal{G}_2 \rangle \vdash \{R\}\mathrm{c}_2\{Q\}}{\langle \mathcal{R}_1 \cup \mathcal{R}_2; \mathcal{G}_1 \cup \mathcal{G}_2 \rangle \vdash \{P\}\mathrm{c}_1; \mathrm{c}_2\{Q\}}$$

ITE:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

$$\frac{\langle \mathcal{R}; \mathcal{G} \rangle \vdash \{P \wedge \mathrm{e} = 0\} \mathrm{c}_2 \{Q\}}{\langle \mathcal{R} \cup \{P\}; \mathcal{G} \rangle \vdash \{P\} \text{ if (e) then } \mathrm{c}_1 \text{ else } \mathrm{c}_2 \{Q\}}$$

$$P \wedge \mathrm{e} = 0 \Rightarrow Q \quad P \Rightarrow P \quad \mathcal{R}' \subseteq \mathcal{R} \quad \mathcal{G}' \subseteq \mathcal{G} \quad Q' \Rightarrow Q$$

WHILE:

$$\frac{\langle \mathcal{R}; \mathcal{G} \rangle \vdash \{P \wedge \mathrm{e} \neq 0\} \mathrm{c} \{P\}}{\langle \mathcal{R} \cup \{Q\}; \mathcal{G} \rangle \vdash \{P\} \text{ while } (e) c \{Q\}.}$$

CONSEQ:

$$\frac{\langle \mathcal{R}'; \mathcal{G}' \rangle \vdash \{P'\} \mathrm{C} \{Q\}}{\langle \mathcal{R} \cup \{P, Q\}; \mathcal{G} \rangle \vdash \{P\} C \{Q\}}$$

With the advent of non-volatile memory, the presence of permanent memory in the storage system requires new storage modes to verify multithreaded concurrent systems. volatile memory storage mode of random access memory (RAM), and non-volatile permanent storage mode(NVM) have emerged in the storage system, as shown in Figures 2 and 3.

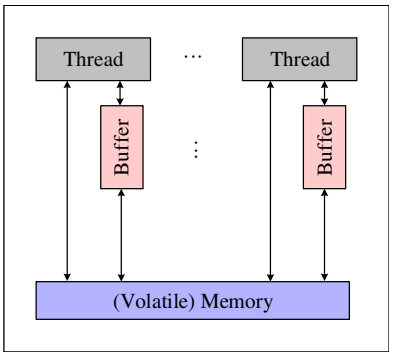

Figure 2. Volatile Memory.

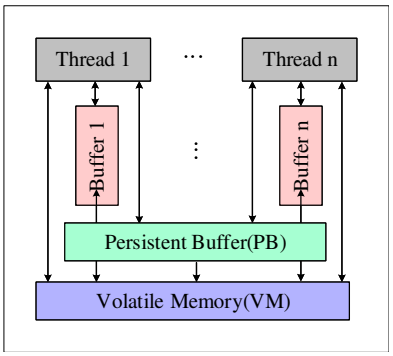

Figure 3. Non-Volatile Memory.

In persistent memory model [39–42], Multithreaded system verification [39,43] is very important, since it is difficult to verify the correctness of a Multithreaded system [44,45] sharing variables [46,47], [47], [48–52]. Prior work fails to well consider the impact of data on control flow, which makes verification more difficult [17–20]. Figure 4 is the two multi-thread systems sharing x and y in the persistent memory mode, r_1, r_2 are local variables, where x, y are shared variables, thread 1 and thread 2 run concurrently, thread 1 and thread 2 assign 2 and 1 respectively to x, and thread 1 reads the value of x, which may be 1 or 2. If the read value of x is 2, 1 is written to the y. thread 2 may read the value of y as 1 or 0 (initial), and judge the value of r_2. If the read value r_2 of y is 0, Number 3 is assigned as the value of x.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

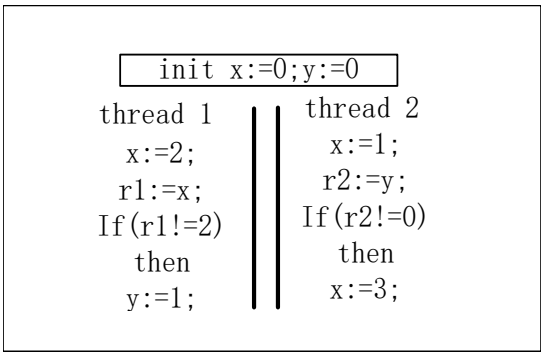

Figure 4. Concurrent code.

2.2. Petri net with read-write data

Definition 6 (Petri net). *Petri net [53,54] is a triplet PN_i=(Pl, Tr, Fl), where Pl and Tr are respectively finite sets of places and transitions* $\mathrm{Pl} \cap \mathrm{Tr} = \varnothing$ *and* $\mathrm{Pl} \cup \mathrm{Tr} \neq \varnothing$). *while* $\mathrm{Fl} \subseteq (\mathrm{Pl} \times \mathrm{Tr}) \cup (\mathrm{Tr} \times \mathrm{Pl})$ *is a set of arcs (flow relationships).* ${}^\bullet x = \{y \in Pl \cup T \mid (y,x) \in Fl\}$; $x^\bullet = \{y \in Pl \cup Tr \mid (x,y) \in Fl\}$. $\forall x \in Pl \cup Tr$: ${}^\bullet x \cup x^\bullet \neq \varnothing$.

Figure 5 is a simple example of Petri net, where $Tr = \{t_1, t_2, t_3\}$; $Pl = \{p_1, p_2, p_3, p_4\}$; $In(t_1, p_1) = 2, In(t_1, p_i) = 0$ for $i = 2,3,4; Oc(t_1, p_2) = 2, Oc(t_1, p_3) = 1, Oc(t_1, p_i) = 0 for i = 1,4$; $In(t_2, p_2) = 1, In(t_2, p_i) = 0$ for i = 1, 3, 4; $Oc(t_2, p_4) = 1, Oc(t_2, p_i) = 0 for i = 1,2,3; In(t_3, p_3) = 1, In(t_3, p_i) = 0 for i = 1,2,4; Oc(t_3, p_4) = 1, Oc(t_3, p_i) = 0 for i = 1,2,3; M_0 = (2,0,0,0)$.

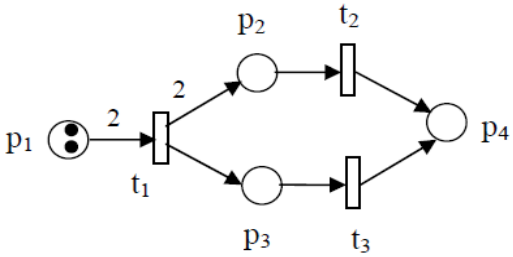

Figure 5. A simple Petri net.

Definition 7 (PN transition rules). *Let PN_i=(N, M) be a simple Petri net and the transition rules :*

(1) For transition $t_1 \in T_i$, *if* $\forall p_i \in {}^\bullet t_i : M_i(p_i) \geq 1$, *it is said that transition* t_i *is enabled under the marking M_i, which is recorded as $M_i[t_i\rangle$;*

(2) If $M_i[t_i\rangle$, the transition t_i can take place under the marking M_i', and a new marking M_i' can be obtained from the transition t_i caused by the marking M_i, which is marked as $M_i[t_i\rangle M_i'$ for $\forall p_i \in P$,

The whole state space of PN_i=(N, M_0) is determined by its network N and initial marking M_0. Figure 6 shows a transition t_1 firing based on Figure 5, which presents new tokens distribution of this Petri net.

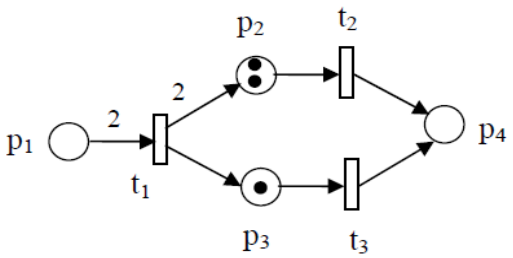

Figure 6. Firing of transition t_1.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

We can find that under marking, M_0 = (2, 0, 0, 0). Firing of t_1 results in a new marking, i.e., M_2. = (0, 2, 1, 0). Again , It follows the firing rule that M_1 In marking M_1= (0, 2, 1, 0), both transitions oft_3 and t_2 are enabled. If t_3 fires, the new marking, i.e., M_3 = (0, 2, 0, 1).If t_2 fires, the new marking, i.e., M_2 = (0, 1, 1, 1).

Definition 8 (Petri net with data: DPN). *An 8-tuple $DPN_i = (Pc, Pd, Tc, Td, Fc, Fd, Pc_0, Pd_0)$ is called Data Petri net if it meets:*

(1) (Pc, Tc, Fc, Pc_0) is a Petri net;

(2) Pd= pd_1=[D_1, val_1], pd_2= [D_2, val_2] . . . pd_n= [D_n, val_n]. [D_i, val_i] is the data place, Pd is a finite set of elements D_i, where val_i is the value of D_i, and the token of the initial pd_i=1, val_i=0.

(3) Fd : Td×Pd ∪ Pd×Td. Fd includes read data (Rd) and write data (Wr) arcs, namely Rd : Pd×Td and Wr: Td×Pd;

(4) Wr writing data transition: write the value to D_i. Rd read data transition: read the value val_i from the data place D_i.

(5) Td: Td×Pd ∪ Pd×Td. Write data transition Td $\xrightarrow{Fd}$Pd, read data transition: Pd$\xrightarrow{Fd}$ Td.

(6) Configuration: Pc→ $\{0, 1, 2 \ldots\}$ and Pd→ $\{1\}$ represent the configuration or state, where $c = (m, \sigma)$, M is the control marking, σ is the data status. Pc_0 is the initial configuration of the control place, and Pd_0 is the initial configuration of the data place, $pd_i = 1, val_i = 0$.

For the data place pd_i∈Pd in DPN, its value is obtained by using the function getValue(pd). For convenience, a pair of data transition functions on conversion are also provided, namely Read: getValue(D_i), Write: setValue(pd_i)=value.

Definition 9 (DPN transition rule). *and set DPN=(DN,M,Σ) It is a Petri net and has the following transition rules:*

(1) For transition $t \in Tc$, if $\forall p \in {}^\bullet t : M(p) \geq 1$, control transition t is said to be enabled under the marking M_1, which is recorded as $M_1[t\rangle$;

(2) If $M_1[t\rangle$, the control transition t can occur under the marking M_1, and the transition t caused by the marking M_1 can get a new marking M_1', which is recorded as $M_1[t\rangle M_1'$,and for $\forall\, p \in Pc \cup Pd$.

$$\left(\mathrm{M}', \Sigma'\right) = \begin{cases} (M(p) - 1, \Sigma) & if \quad p \in {}^\bullet t - t^\bullet \\ (M(p) + 1, \Sigma') & if \quad p \in t^\bullet - {}^\bullet t \\ M(p) & else \end{cases} \tag{1}$$

For the transition $t \in Tc$, if $\forall p \in {}^\bullet t : M(p) \geq 1$, the data transition t in marking (M_i, Σ) is enabled, which is recorded as $(M_i, \Sigma)[t\rangle(M_i', \Sigma)$;

(3) In the marking (M_i, Σ), if (M_i, Σ)[td⟩, data transition $td \in Td$ is enabled and triggered to get a new marking (M_i', Σ'), i.e., $(M_i, \Sigma)\ [td > \left(M_i', \Sigma'\right)$.

$$\left(M', \Sigma'\right) = \begin{cases} (M(p) - 1, \Sigma) & if \quad p \in {}^\bullet t - t^\bullet \\ (M(p) + 1, \Sigma) & if \quad p \in t^\bullet - {}^\bullet t \wedge t \in Pc \\ (M(p) + 1, \Sigma') & if \quad p \in t^\bullet - {}^\bullet t \wedge t \in Pd \\ M(p) \in else & \end{cases} \tag{2}$$

Wherein, Σ[Fd $\xrightarrow{(Di,value)}$ Pd > Σ'.

(4) Finally, the entire state space of DPN_i=(DN,(M, Σ)) is determined by its network DPN_i and initial marking M_0.

In Figure 7, T_1 is the write data transition, which realizes the write data value 2 to the data place x (there is a green arrow line between the write transition and the data place office).t_8 and t_9 are control transitions. The first step is to obtain the write permission in x, i.e., the token of x. After obtaining the write permission in x, write the value 2 to x, and then return the write permission of the token to x; T_2

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

is the change of write data. Write the data value 1 to the data place x, and the process is the same as the change of write data t_1. T_3 is read data transition. Read the value of data place x (there is a blue line between the data place object and the change of read data, and the end of the read data place object is a solid circle). First, judge whether data place object x has read permission, i.e., whether there is a token in x, if hold, it can be read. Read the value of x through the reading arc, if there is no token in x, x cannot be read. The semantics of read data transition t_6 are the same as that of read data transition t_3.

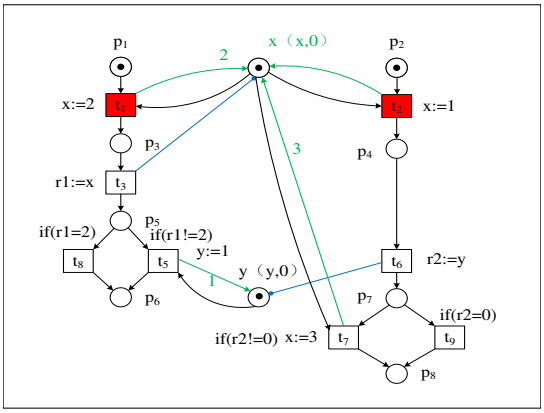

Figure 7. Concurrent code of Figure 4 described in DPN.

In Figure 7, if P_1 has a token and the data place Dx has a token, t_1 can occur. After the occurrence, the value of $x(x, 2)$ in the data place change to number 2, and the status is shown in Figure 8; If there is a token in P_2, and there is a token in data place $x(x, 2)$, t_2 can occur. After t_2 transition occur, the value in data place $x(x, 1)$ change to number 1; When P_3 has a token, and the data place has a token in $x(x, 1)$, then t_3 transition can occur. Read the value in $x(x, 1)$ of the data place, and the token in $x(x, 1)$ of the data place remains unchanged, and read the value of data place x is number 1.

In Figure 8, if t_3 is triggered again (i.e., t_1 is triggered first, then t_3 is triggered), the state is shown in Figure 9.

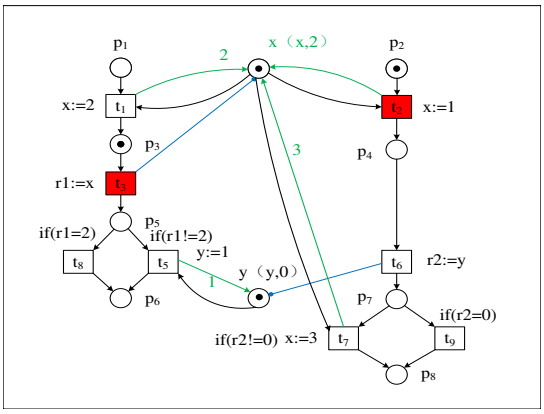

Figure 8. Concurrent code of Figure 7 after t_1 is fired.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

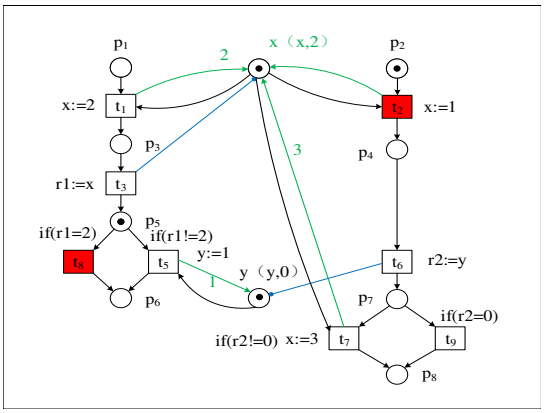

Figure 9. Concurrent code of Figure 7 after t_1 and t_3 is fired.

3. Convert the writing event to the Beginning and Ending event

Writing events takes time in theory, so when we study the sequence of reading and writing events, we consider the time of writing events and convert the writing events into two events: starting and ending [55,56]. BE for a given writing event Σ = {Beg a|a$\in$ Σ} $\bigcup$ {End a|a $\in$ Σ}. The write data event is converted to the Beginning and Ending events according to Figure 10. For each writing event in Σ, it is transferred into beginning and ending event following by BEΣ={Ba|a$\in\Sigma$}$\bigcup${Ea|a$\in\Sigma$}, where B refers to the beginning event and E refers to the ending one.

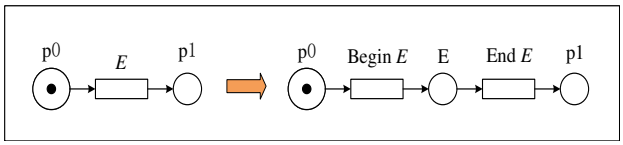

Figure 10. Conversion from writing event to beginning and ending one.

Definition 10. *Let Petri PN_i = (P, T, TR, TW, D, m_0) be a Petri net with data.*

(1) For each $t_i \in TW$,now,we define $Beg - t_i$ the beginning of t_i and $Endt - t_i$ the end of t_i, and the set $BegT = \{Begt \mid t \in Tw\} \cup \{Endt \mid t \in Tw\}$. The elements of TW are called BE-transitions.

(2) For each $t_i \in TW$, we define: (a) ${}^\bullet Bt = {}^\bullet t$, (b) $Bt^\bullet = \{t\}$, (c) ${}^\bullet Et = \{t\}$, (d) $Et^\bullet = t^\bullet$.

(3) We say that a set $m \subseteq P \cup T$ is an extended marking if $m \cap ({}^\bullet m \cup m^\bullet) = \varnothing$.

(4) An enabled BE-transition same enabling capacity as Petri net

(5) An extended firing sequence same enabling capacity as Data Petri net

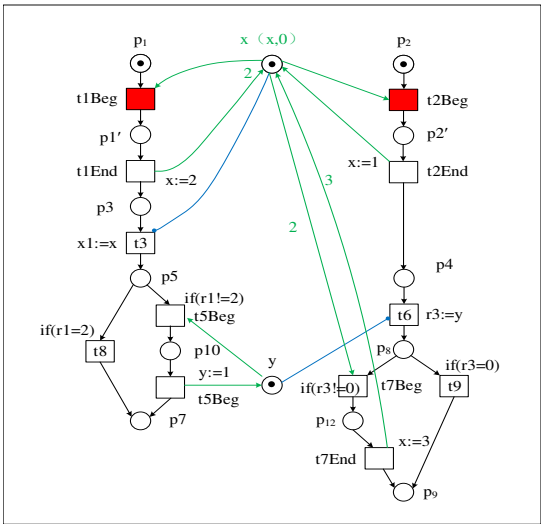

Figure 11. The writing events of concurrent code in Figure 7 to beginning and ending ones.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

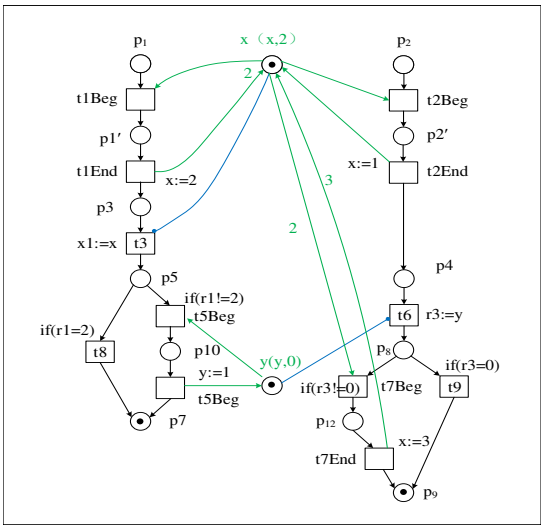

Figure 12. result[x=2,y=0].

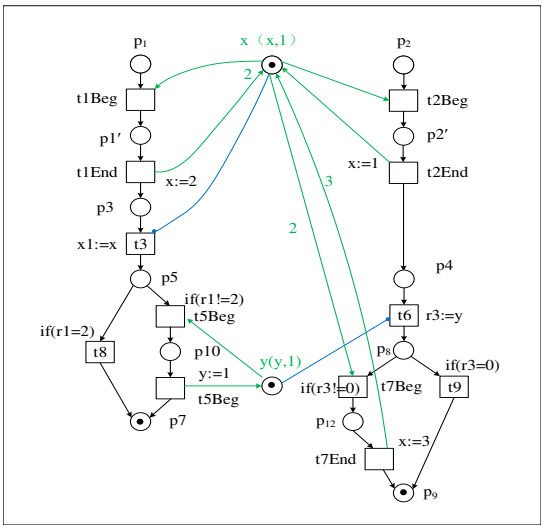

Figure 13. result[x=1,y=1].

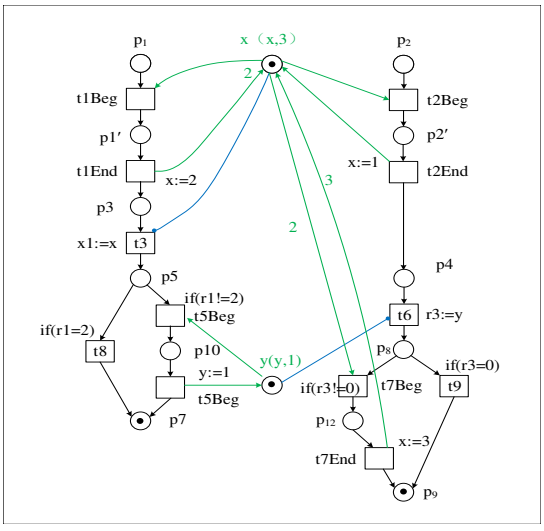

Figure 14. result[x=3,y=1].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2023 doi:10.20944/preprints202305.0049.v1

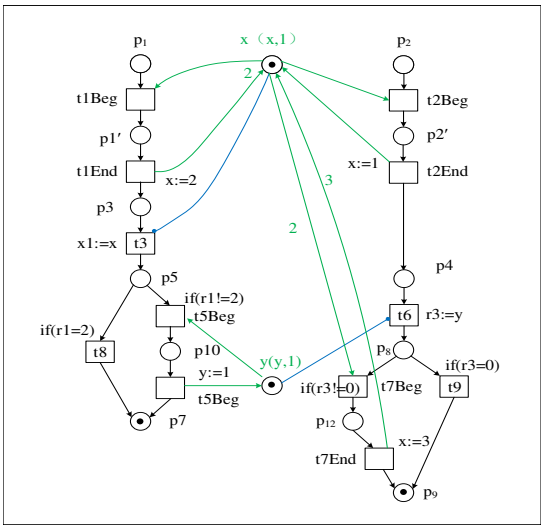

Figure 15. result[x=3,y=0].

Table 1. The firing trace of transition and the final results of shared variables x and y.

Firing trace of transition	x, y
$T1: Bt1, Et1, Bt2, Et2, t3, Bt5, Et5, t6, Bt7, Et7$	3, 0
$T2: Bt1, Et1, Bt2, Et2, t3, t6, t8, Bt5, Et5$	1, 1
$T3: Bt1, Et1, t3, t4, Bt2, Et2, t6, t8$	1, 1
$T5: Bt2, Et2, t6, Bt1, Et1, t8, t3, Bt5, Et5$	1, 1
$T4: Bt2, Et2, t6, Bt1, Et1, t3, Bt5, Et5, Bt7, Et7$	3, 1
$T7: Bt2, Et2, Bt1, Et1, t3, Bt5, Et5, t6, Bt7, Et7$	3, 1
$T6: Bt2, Et2, t6, t8, Bt1, Et1, t3, t4$	2, 0

4. Conclusion

The verification of concurrent system has always been a hot topic of academic research. The verification of concurrent system based on trace realizes the detection of stateless model, which can well reduce the state space and memory consumption. The concept of trace itself is based on state equivalence classification, and again based on the equivalence of numerical traces, classification will further reduce the number of traces.

This paper first introduces the sequence of events describing the operation of a multithreaded concurrent system. The sequence of events has order, bias order, complete order, and hierarchical sequence, interval sequence. In a concurrent multi-threaded system, the events of writing the data need some time, and an interval sequence can be formed between the events of writing the written data. In order to better analyze the running of multi-threading of the concurrent system, the events of read and write data are presented as interval sequence, presented in the form of trace, describing the results of multi-threading running. With a multi-threaded case as the motivation, describe multi-threaded system as Petri net with data, in writing data event x to B (x) and E (x), the event interval sequence, gives the trajectory of the case, and present the state of the data, perfect trajectory with data.

The research [57] in this paper only realizes the interval sequence [4]of events in multi-threaded systems, and analyzed the operation results, which has achieved initial success. However, the data consistency analysis still needs further analysis, and there are still many research work to be further conducted. One is to add the time label to analyze the amount of time. Second, the new permanent storage mode brings a new storage access mode to the concurrent system, which gives new challenges to the model detection of the concurrent system. Further research will be done from the above two aspects.

The equivalence of traces is also the focus of recent research. Based on numerical equivalence trace, some studies have shown that based on numerical trace number less than M trace, especially based on read equivalent trace, less than the number of M trace, this paper has realized the numerical involved in the trace of the trace, the next step, whether can be based on read equivalence, or based on numerical equivalence applied to the Petri net trace, reduce the number of traces, is also a good research idea.

Funding: The APC was funded by National Natural Science Foundation of China (62172166).

References

1. Alglave, J.; Deacon, W.; Grisenthwaite, R.; Hacquard, A.; Maranget, L. Armed Cats: Formal Concurrency Modelling at Arm. *ACM Trans. Program. Lang. Syst.* **2021**, *43*, 8:1–8:54. doi:10.1145/3458926.
2. Alglave, J.; Kroening, D.; Nimal, V.; Poetzl, D. Don't Sit on the Fence: A Static Analysis Approach to Automatic Fence Insertion. *ACM Trans. Program. Lang. Syst.* **2017**, *39*, 6:1–6:38. doi:10.1145/2994593.
3. Alglave, J.; Cousot, P. Ogre and Pythia: an invariance proof method for weak consistency models. Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017; Castagna, G.; Gordon, A.D., Eds. ACM, 2017, pp. 3–18. doi:10.1145/3009837.3009883.
4. Janicki, R.; Kleijn, J.; Koutny, M.; Mikulski, L. *Paradigms of Concurrency - Observations, Behaviours, and Systems - a Petri Net View*; Vol. 1020, *Studies in Computational Intelligence*, Springer, 2022. doi:10.1007/978-3-662-64821-6.
5. Janicki, R.; Koutny, M. Operational Semantics, Interval Orders and Sequences of Antichains. *Fundam. Informaticae* **2019**, *169*, 31–55. doi:10.3233/FI-2019-1838.
6. Horn, A.; Alglave, J. Concurrent Kleene Algebra of Partial Strings. *CoRR* **2014**, *abs/1407.0385*, [1407.0385].
7. Lutz-Ley, A.; López-Mellado, E. Stability Analysis of Discrete Event Systems Modeled by Petri Nets Using Unfoldings. *IEEE Trans Autom. Sci. Eng.* **2018**, *15*, 1964–1971. doi:10.1109/TASE.2018.2830385.
8. de Visme, M.; Winskel, G. Causal Unfoldings. 8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019, June 3-6, 2019, London, United Kingdom; Roggenbach, M.; Sokolova, A., Eds. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, Vol. 139, *LIPIcs*, pp. 9:1–9:18. doi:10.4230/LIPIcs.CALCO.2019.9.
9. Janicki, R.; Kleijn, J.; Koutny, M.; Mikulski, L. Relational structures for concurrent behaviours. *Theor. Comput. Sci.* **2021**, *862*, 174–192. doi:10.1016/j.tcs.2020.10.019.
10. Rodríguez, C. Verification based on unfoldings of Petri nets with read arcs. (Vérification à l'aide de dépliages de réseaux de Petri étendus avec des arcs de lecture). PhD thesis, École normale supérieure de Cachan, Paris, France, 2013.
11. Baldan, P.; Bruni, R.; Corradini, A.; Gadducci, F.; Melgratti, H.C.; Montanari, U. Event Structures for Petri nets with Persistence. *Log. Methods Comput. Sci.* **2018**, *14*. doi:10.23638/LMCS-14(3:25)2018.
12. Baldan, P.; Gorla, D.; Padoan, T.; Salvo, I. Characterising spectra of equivalences for event structures, logically. *Inf. Comput.* **2022**, *285*, 104887. doi:10.1016/j.ic.2022.104887.
13. Baldan, P.; Corradini, A.; Gadducci, F. Concurrent semantics for fusions: Weak prime domains and connected event structures. *Inf. Comput.* **2021**, *281*, 104770. doi:10.1016/j.ic.2021.104770.
14. Kähkönen, K.; Heljanko, K. Testing Multithreaded Programs with Contextual Unfoldings and Dynamic Symbolic Execution. 14th International Conference on Application of Concurrency to System Design, ACSD 2014, Tunis La Marsa, Tunisia, June 23-27, 2014. IEEE Computer Society, 2014, pp. 142–151. doi:10.1109/ACSD.2014.20.
15. Janicki, R. On Interval Semantics of Inhibitor and Activator Nets. Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings; Donatelli, S.; Haar, S., Eds. Springer, 2019, Vol. 11522, *Lecture Notes in Computer Science*, pp. 192–212. doi:10.1007/978-3-030-21571-2_12.
16. Gondelman, L.; Gregersen, S.O.; Nieto, A.; Timany, A.; Birkedal, L. Distributed causal memory: modular specification and verification in higher-order distributed separation logic. *Proc. ACM Program. Lang.* **2021**, *5*, 1–29. doi:10.1145/3434323.
17. Coti, C.; Petrucci, L.; Rodríguez, C.; Sousa, M. Quasi-optimal partial order reduction. *Formal Methods Syst. Des.* **2021**, *57*, 3–33. doi:10.1007/s10703-020-00350-4.

18. Schemmel, D.; Büning, J.; Rodríguez, C.; Laprell, D.; Wehrle, K. Symbolic Partial-Order Execution for Testing Multi-Threaded Programs. Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I; Lahiri, S.K.; Wang, C., Eds. Springer, 2020, Vol. 12224, *Lecture Notes in Computer Science*, pp. 376–400. doi:10.1007/978-3-030-53288-8_18.
19. Nguyen, H.T.T.; Rodríguez, C.; Sousa, M.; Coti, C.; Petrucci, L. Quasi-Optimal Partial Order Reduction. Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II; Chockler, H.; Weissenbacher, G., Eds. Springer, 2018, Vol. 10982, *Lecture Notes in Computer Science*, pp. 354–371. doi:10.1007/978-3-319-96142-2_22.
20. Sousa, M.; Rodríguez, C.; D'Silva, V.V.; Kroening, D. Abstract Interpretation with Unfoldings. Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II; Majumdar, R.; Kuncak, V., Eds. Springer, 2017, Vol. 10427, *Lecture Notes in Computer Science*, pp. 197–216. doi:10.1007/978-3-319-63390-9_11.
21. Xiang, D.; Liu, G.; Yan, C.; Jiang, C. Detecting data-flow errors based on Petri nets with data operations. *IEEE/CAA Journal of Automatica Sinica* **2018**, *5*, 251–260. doi:10.1109/JAS.2017.7510766.
22. Zhao, F.; Xiang, D.; Liu, G.; Jiang, C. A New Method for Measuring the Behavioral Consistency Degree of WF-Net Systems. *IEEE Transactions on Computational Social Systems* **2022**, *9*, 480–493. doi:10.1109/TCSS.2021.3099475.
23. Xiang, D.; Liu, G.; Yan, C.; Jiang, C. Checking the Inconsistent Data in Concurrent Systems by Petri Nets with Data Operations. 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), 2016, pp. 501–508. doi:10.1109/ICPADS.2016.0073.
24. Huang, Z.; Xu, X.; Zhu, H.; Zhou, M. An Efficient Group Recommendation Model With Multiattention-Based Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* **2020**, *31*, 4461–4474. doi:10.1109/TNNLS.2019.2955567.
25. Zhu, H.; Liu, G.; Zhou, M.; Xie, Y.; Kang, Q. Dandelion Algorithm With Probability-Based Mutation. *IEEE Access* **2019**, *7*, 97974–97985. doi:10.1109/ACCESS.2019.2927846.
26. He, L.; Liu, G.; Zhou, M. Petri-Net-Based Model Checking for Privacy-Critical Multiagent Systems. *IEEE Transactions on Computational Social Systems* **2022**, pp. 1–14. doi:10.1109/TCSS.2022.3164052.
27. He, L.; Liu, G. Model Checking CTLK Based on Knowledge-Oriented Petri Nets. 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019, pp. 1139–1146. doi:10.1109/HPCC/SmartCity/DSS.2019.00161.
28. Tao, X.; Liu, G.; Yang, B.; Yan, C.; Jiang, C. Workflow Nets With Tables and Their Soundness. *IEEE Transactions on Industrial Informatics* **2020**, *16*, 1503–1515. doi:10.1109/TII.2019.2949591.
29. Yu, W.; Yan, C.G.; Ding, Z.; Jiang, C.; Zhou, M. Modeling and Verification of Online Shopping Business Processes by Considering Malicious Behavior Patterns. *IEEE Transactions on Automation Science and Engineering* **2016**, *13*, 647–662. doi:10.1109/TASE.2014.2362819.
30. Yu, W.; Yan, C.; Ding, Z.; Jiang, C.; Zhou, M. Analyzing E-Commerce Business Process Nets via Incidence Matrix and Reduction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **2018**, *48*, 130–141. doi:10.1109/TSMC.2016.2598287.
31. Wang, M.; Ding, Z.; Zhao, P.; Yu, W.; Jiang, C. A Dynamic Data Slice Approach to the Vulnerability Analysis of E-Commerce Systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **2020**, *50*, 3598–3612. doi:10.1109/TSMC.2018.2862387.
32. Ding, Z.; Wang, S.; Jiang, C. Kubernetes-Oriented Microservice Placement with Dynamic Resource Allocation. *IEEE Transactions on Cloud Computing* **2022**, pp. 1–1. doi:10.1109/TCC.2022.3161900.
33. Wang, C.; Wang, C.; Zhu, H.; Cui, J. LAW: Learning Automatic Windows for Online Payment Fraud Detection. *IEEE Transactions on Dependable and Secure Computing* **2021**, *18*, 2122–2135. doi:10.1109/TDSC.2020.3037784.
34. Cui, J.; Yan, C.; Wang, C. ReMEMBeR: Ranking Metric Embedding-Based Multicontextual Behavior Profiling for Online Banking Fraud Detection. *IEEE Transactions on Computational Social Systems* **2021**, *8*, 643–654. doi:10.1109/TCSS.2021.3052950.
35. Chen, X.; Wang, C.; Cui, J.; Yang, Q.; Teng, H.; Jiang, C. Incorporating Prior Knowledge in Local Differentially Private Data Collection for Frequency Estimation. *IEEE Transactions on Big Data* **2022**, pp. 1–13. doi:10.1109/TBDATA.2022.3190033.

36. Wang, C.; Yang, B.; Cui, J.; Wang, C. Fusing Behavioral Projection Models for Identity Theft Detection in Online Social Networks. *IEEE Transactions on Computational Social Systems* **2019**, *6*, 637–648. doi:10.1109/TCSS.2019.2917003.
37. Liang, Y.; Li, M.; Jiang, C.; Liu, G. CEModule: A Computation Efficient Module for Lightweight Convolutional Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* **2021**, pp. 1–12. doi:10.1109/TNNLS.2021.3133127.
38. Janicki, R.; Mikulski, L. Algebraic Structure of Step Traces and Interval Traces. *Fundam. Informaticae* **2020**, *175*, 253–280. doi:10.3233/FI-2020-1956.
39. Lahav, O.; Boker, U. Decidable verification under a causally consistent shared memory. Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020; Donaldson, A.F.; Torlak, E., Eds. ACM, 2020, pp. 211–226. doi:10.1145/3385412.3385966.
40. Lahav, O.; Boker, U. What's Decidable About Causally Consistent Shared Memory? *ACM Trans. Program. Lang. Syst.* **2022**, *44*, 8:1–8:55. doi:10.1145/3505273.
41. Lahav, O.; Namakonov, E.; Oberhauser, J.; Podkopaev, A.; Vafeiadis, V. Making weak memory models fair. *Proc. ACM Program. Lang.* **2021**, *5*, 1–27. doi:10.1145/3485475.
42. Khyzha, A.; Lahav, O. Taming x86-TSO persistency. *Proc. ACM Program. Lang.* **2021**, *5*, 1–29. doi:10.1145/3434328.
43. Cho, M.; Lee, S.; Hur, C.; Lahav, O. Modular data-race-freedom guarantees in the promising semantics. PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021; Freund, S.N.; Yahav, E., Eds. ACM, 2021, pp. 867–882. doi:10.1145/3453483.3454082.
44. Wang, P.; Ding, Z.; Jiang, C.; Zhou, M. Constraint-Aware Approach to Web Service Composition. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **2014**, *44*, 770–784. doi:10.1109/TSMC.2013.2280559.
45. Wang, P.; Ding, Z.; Jiang, C.; Zhou, M.; Zheng, Y. Automatic Web Service Composition Based on Uncertainty Execution Effects. *IEEE Transactions on Services Computing* **2016**, *9*, 551–565. doi:10.1109/TSC.2015.2412943.
46. Kähkönen, K.; Heljanko, K. Testing Multithreaded Programs with Contextual Unfoldings and Dynamic Symbolic Execution. 2014 14th International Conference on Application of Concurrency to System Design, 2014, pp. 142–151. doi:10.1109/ACSD.2014.20.
47. Lutz-Ley, A.; López-Mellado, E. Stability Analysis of Discrete Event Systems Modeled by Petri Nets Using Unfoldings. *IEEE Transactions on Automation Science and Engineering* **2018**, *15*, 1964–1971. doi:10.1109/TASE.2018.2830385.
48. André, E.; Chatain, T.; Rodríguez, C. Preserving Partial-Order Runs in Parametric Time Petri Nets. *ACM Trans. Embed. Comput. Syst.* **2016**, *16*. doi:10.1145/3012283.
49. Xie, Y.; Liu, G.; Yan, C.; Jiang, C.; Zhou, M.; Li, M. Learning Transactional Behavioral Representations for Credit Card Fraud Detection. *IEEE Transactions on Neural Networks and Learning Systems* **2022**, pp. 1–14. doi:10.1109/TNNLS.2022.3208967.
50. Xie, Y.; Liu, G.; Yan, C.; Jiang, C.; Zhou, M. Time-Aware Attention-Based Gated Network for Credit Card Fraud Detection by Extracting Transactional Behaviors. *IEEE Transactions on Computational Social Systems* **2022**, pp. 1–13. doi:10.1109/TCSS.2022.3158318.
51. Xiang, D.; Liu, G.; Yan, C.; Jiang, C. A Guard-Driven Analysis Approach of Workflow Net with Data. *IEEE Transactions on Services Computing* **2021**, *14*, 1650–1661. doi:10.1109/TSC.2019.2899086.
52. Xiang, D.; Lin, S.; Wang, X.; Liu, G. Checking Missing-Data Errors in Cyber-Physical Systems Based on the Merged Process of Petri Nets. *IEEE Transactions on Industrial Informatics* **2022**, pp. 1–10. doi:10.1109/TII.2022.3181669.
53. Kähkönen, K.; Heljanko, K. Testing Programs with Contextual Unfoldings. *ACM Trans. Embed. Comput. Syst.* **2018**, *17*, 23:1–23:25. doi:10.1145/2810000.
54. Kähkönen, K.; Saarikivi, O.; Heljanko, K. Unfolding based automated testing of multithreaded programs. *Autom. Softw. Eng.* **2015**, *22*, 475–515. doi:10.1007/s10515-014-0150-6.
55. Bui, T.L.; Chatterjee, K.; Gautam, T.; Pavlogiannis, A.; Toman, V. The reads-from equivalence for the TSO and PSO memory models. *Proc. ACM Program. Lang.* **2021**, *5*, 1–30. doi:10.1145/3485541.
56. Agarwal, P.; Chatterjee, K.; Pathak, S.; Pavlogiannis, A.; Toman, V. Stateless Model Checking Under a Reads-Value-From Equivalence. Computer Aided Verification - 33rd International Conference, CAV 2021,

Virtual Event, July 20-23, 2021, Proceedings, Part I; Silva, A.; Leino, K.R.M., Eds. Springer, 2021, Vol. 12759, *Lecture Notes in Computer Science*, pp. 341–366. doi:10.1007/978-3-030-81685-8_16.

57. He, L.; Liu, G. Verifying Computation Tree Logic of Knowledge via the Similar Reachability Graphs of Knowledge-oriented Petri Nets. 2020 39th Chinese Control Conference (CCC), 2020, pp. 5026–5031. doi:10.23919/CCC50068.2020.9188719.