BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LXI No. 3/2009	1 - 12	Seria Tehnică

Annotated Logic Program EVALPSN Based Process Order Control

Kazumi Nakamatsu

University of Hyogo, Himeji 670-0092, Japan
e-mail: nakamatu@shse.u-hyogo.ac.jp

Abstract

We have already proposed a paraconsistent annotated logic program called EVALPSN (Extended Vector Annotated Logic Program with Strong Negation) and applied it to various control based on safety verification. In this paper, a new EVALPSN called bf (before-after)-EVALPSN that can deal with before-after relation between two processes is introduced, and its application to real-time process order control based on safety verification is presented with simple examples. Moreover, some useful features of bf-EVALPSN for real-time control are introduced.

Key words: *annotated logic program, process order control, safety verification, EVALPSN, bf-EVALSN.*

Introduction

The safety verification for process order is an important issue in various kinds of process control. For example, two different kinds of liquid such as nitric acid and caustic soda are used for cleaning pipelines in a brewery plant. Usually there must be a water process between those liquid processes in order to avoid dangerous explosion caused by mixing. Such a process order must be verified to avoid dangerous accidents. In this paper, we introduce a novel intelligent tool based on a paraconsistent annotated logic program for verifying the safety for process order control in real-time.

We have already developed a paraconsistent annotated logic program called EVALPSN (Extended Vector Annotated Logic Program with Strong Negation) [4], and also applied it to various kinds of intelligent control such as pipeline valve control [6]. Moreover, a new EVALPSN called bf(before-after)-EVALPSN has been developed for dealing with before-after relation between two time intervals(processes) dynamically [8]. In bf-EVALP- SN, before-after relations between processes are represented in 2-dimensional vector annotations whose first and second components represent before and after degrees respectively, and vector annotations are determined in real-time according to start/finish information of processes. We show how the bf-EVALPSN safety verification based real-time process order control is carried out with simple examples.

Although the bf-EVALPSN real-time process order control can be performed by simple integer computation and it can contribute to reduce the computation cost of the real-time control, if all before-after relations between all processes have to be computed, it would cost much computation. For example, if there are ten processes, forty five before-after relations have to be considered. Therefore, we introduce useful unique reasoning in bf-EVALPSN that can be

applied for reducing process order control time. We have transitive inference rules to derive the before-after relation between processes Prl and Pr3 from the bf-relations between processes Prl and Pr2 and between processes Pr2 and Pr3 in bf-EVALPSN, which are called bf-relation inference rules (bf-inf rules for short). If we use these rules, forty five before-after relations to be computed from process start/finish information can be reduced nine before-after relations between adjoining processes. We introduce how to derive some of bf-inf rules and how to apply them to the real-time process order control.

This paper is organized as follows: EVALPSN is briefly reviewed in Section 1; bf-EVALPSN and its implementation for determining before-after relation are introduced in Section 1; it is shown that how to apply bf-EVALPSN to the real-time process order control with examples in Section 1; and practicai inference rules in bf-EVALPSN for real-time processing are briefly introduced in Section 1; last, concluding remarks are provided.

EVALPSN

Generally, a truth value called an *annotation* is explicitly attached to each literal in annotated logic program [2]. For example, let p be a literal, μ an annotation, then $p:\mu$, is called an *annotated literal*. The set of annotations constitutes a complete lattice. An annotation in EVALPSN has a form of $[(i, j), \mu]$ called an *extended vector annotation*. The first component (i, j) is called a *vector annotation* and the set of vector annotations constitutes a complete lattice,

$$\tau_v(n) = \{(x, y) \mid 0 \le x \le n, 0 \le y \le n, x, y \, and \, n \, are \, integers\}.$$

The ordering $(\preceq_v)$ of lattice $\tau_v(n)$ is defined as: let (x_1, y_1), $(x_2, y_2) \in \tau_v(n)$,

$$(x_1, y_1) \preceq_v (x_2, y_2) \Leftrightarrow x_1 \le x_2 \, and \, y_1 \le y_2 .$$

For each extended vector annotated literal $p:[(i, j), \mu]$, the integer i denotes the amount of positive information to support the literal p and the integer j denotes that of negative information. The second component μ is an index of fact and deontic notions such as obligation, and the set of the second annotations constitutes the complete lattice,

$$\tau_d = \{\bot, \alpha, \beta, \gamma, *_1, *_2, *_3, \mathrm{T}\}.$$

The ordering $(\preceq_d)$ of lattice τ_d is described by the Hasse's diagram in Figure 1. Then, the complete lattice $\tau_e(n)$ of extended vector annotations is defined as the product $\tau_v(n) \times \tau_d$ The order $(\preceq_e)$ of lattice $\tau_e(n)$ is defined as: *let* $[(i_1, j_1), \mu_1]$ and $[(i_2, j_2), \mu_2]$ be extended vector annotations,

$$[(i_1, j_1), \mu_1] \preceq_e [(i_2, j_2), \mu_2] \Leftrightarrow (i_1, j_1) \preceq_v (i_2, j_2) \text{ and } \mu_1 \preceq_d \mu_2 .$$

The intuitive meaning of each member of lattice τ_d is $\bot$ (unknown), α (fact), β (obligation), γ (non-obligation), $*_1$ (fact and obligation), $*_2$ (obligation and non-obligation), $*_3$ (fact and non-obligation), and T (inconsistency).

There are two kinds of epistemic negation $\neg_1$ and $\neg_2$ in EVALPSN, which are defined as mappings over lattices $\tau_v(n)$ and τ_d, respectively.

Definition 1 (epistemic negations $\neg_1$ and $\neg_2$ in EVALPSN)

$$\forall \mu \in \tau_d : \neg_1([(i, j), \mu]) = [(j, i), \mu]$$

$$\neg_2([(i, j), \bot]) = [(i, j), \bot], \qquad \neg_2([(i, j), \alpha]) = [(i, j), \alpha],$$

$$\neg_2([(i, j), \beta]) = [(i, j), \gamma], \qquad \neg_2([(i, j), \gamma]) = [(i, j), \beta],$$

$$\neg_2([(i, j), *_1]) = [(i, j), *_3], \qquad \neg_2([(i, j), *_2]) = [(i, j), *_2],$$

$$\neg_2([(i, j), *_3]) = [(i, j), *_1], \qquad \neg_2([(i, j), \mathrm{T}]) = [(i, j), \mathrm{T}].$$

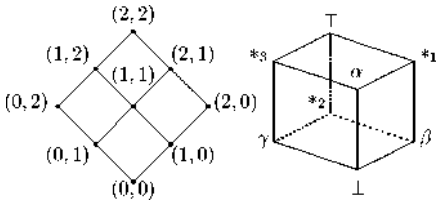


Fig. 1. Lattice $\tau_v(2)$ and Lattice τ_d.

Then, the epistemic negations can be eliminated by the syntactical operations in Definition 1. There also is ontological (strong) negation ($\sim$) in EVALPSN, which is defined by epistemic negations $\neg_1$ or $\neg_2$, and it works as classical negation [4].

Definition 2 (strong negation) [3] Let F be any formula and $\neg$ be $\neg_1$ or $\neg_2$.

$$\sim F =_{def} F \rightarrow ((F \rightarrow F) \wedge \neg(F \rightarrow F)).$$

Definition 3 (weva-literal) Let p be a literal. $p:[(i,0),\mu]$ and $p:[(0,j),\mu]$ are called *weva-literals*, where $i, j \in \{1,2,...\}$ and $\mu \in \{\alpha, \beta, \gamma\}$.

Definition 4 (EVALPSN) If $L_0,...,L_n$ are weva-literals,

$$L_1 \wedge ... \wedge L_i \wedge \sim L_{i+1} \wedge ... \wedge \sim L_n \rightarrow L_0$$

is called an *EVALPSN clause*. An *EVALPSN* is a finite set of EVALPSN clauses.

Deontic notions such as obligation and fact are represented by extended vector annotations as follows: let m be a positive integer;

"fact" is represented by an annotation $[(m,0),\alpha]$,
"obligation" is done by an annotation $[(m,0),\beta]$,
"forbiddance" is done by an annotation $[(0,m),\beta]$,
"permission" is done by an annotation $[(0,m),\gamma]$.

Before-after EVALPSN

First of all, we introduce a particular literal $R(pi, pj, t)$ whose vector annotation represents the before-after relation between processes $\mathrm{Pr}_i(pi)$ and $\mathrm{Pr}_j(pj)$ at time t, which is called a *bf-literal*[1].

[1] Hereafter, the word "before-after" is abbreviated as just "bf" in this paper.

Definition 5 (bf-EVALPSN) An extended vector annotated literal

$$R(p_i, p_j, t):[\mu_1, \mu_2]$$

is called a *bf-EVALP literal*, where μ_1 is a vector annotation and $\mu_2 \in \{\alpha, \beta, \gamma\}$. If an EVALPSN clause contains a bf-EVALP literal, it is called a *bf-EVALPSN clause* or just a *bf-EVALP clause* if it contains no strong negation. A *bf-EVALPSN* is a finite set of bf-EVALPSN clauses.

In order to represent bf-relations between processes, we provide a paraconsistent before-after interpretation for vector annotations of bf-literals, which are called *bf-annotations*. Exactly speaking, bf-relations between processes are classified into meaningful fifteen kinds according to start/finish times of two processes in bf-EVALPSN though [8], we consider ten kinds of bf-relations for simplicity.

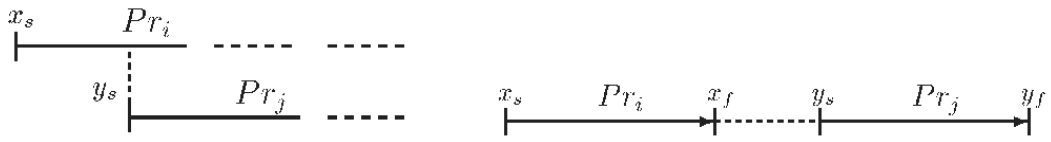


Fig. 2. Before / After Disjoint Before / After.

Fig. 3. Immediate Before / After Join Before / After.

Suppose that there are two processes, Pr_i with its start /finish time x_s / x_f, and Pr_j with its start/finish time y_s / y_f.

Before (be)/After (af) First of all, we define the most basic bf-relations *before/after* based on the before-after relation between each start time of two processes, which are represented by bf-annotations **be/af**, respectively. If one process has started before/after another one, then the bf-relations between them are defined as "before(**be**)/ after(**af**)", respectively. They are described in the left process chart in Figure 2 with the condition that Pr_i has started before Pr_j.

Disjoint Before (db)/After (da) Bf-relations *disjoint before/after* between Pr_i and Pr_j are represented by bf-annotations **db/da,** respectively. The expression "disjoint before/after" implies that there is a timelag between the earlier process finish time and the later one start time. They are described in the right process chart in Figure 2 with the condition that Pr_i has finished before Pr_j starts.

Immediate Before (mb)/After (ma) Bf-relations *immediate before/after* between Pr_i and Pr_j are represented by bf-annotations **mb/ma,** respectively. The expression "immediate before/after" implies that there is no timelag between the earlier process finish time and the later one start time. They are described in the left process chart in Figure 3 with the condition that Pr_i has finished immediately before Pr_j starts.

Joint Before (jb)/After (ja) Bf-relations *joint before/after* between Pr_i and Pr_j are represented by bf-annotations **jb/ja,** respectively. The expression "joint before/after" implies that the two processes overlap and the earlier process has finished before the later one finishes. The bf-relations are described in the right process chart in Figure 3 with the condition that Pr_i has started before Pr_j starts and Pr_i has finished before Pr_j finishes.

Included Before (ib)/After (ia) Bf-relations *included before/after* between Pr_i and Pr_j are represented by bf-annotations **ib/ia,** respectively. The expression "included before/after" implies that one process has started/finished before/after another one starts/ finishes, respectively. They are described in the process chart in Figure 4 with the condition that Pr_i has started before Pr_j starts and finished after Pr_j finishes.

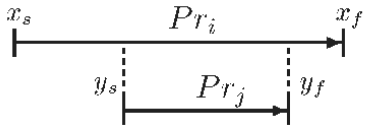
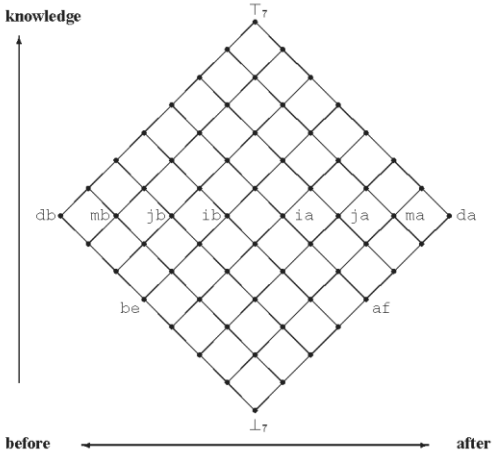


Fig. 4. Included Before /After.

If we take the before-after measure over the ten bf-annotations as the horizontal order and the before-after knowledge amount of them as the vertical one, we obtain the complete bi-lattice $\tau_v(7)_{bf}$ of bf-annotations in Figure 5.



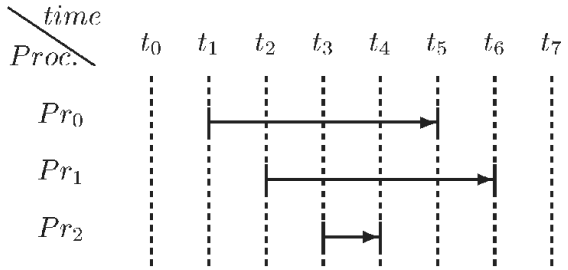


Fig. 5. Lattice $\tau_v(7)_{bf}$ Process Timing Chart.

Then, there is the following correspondence between bf-annotations and vector annotations:

$$be(0,4)/af(4,0), db(0,7)/da(7,0), mb(1,6)/ma(6,1), jb(2,5)/ja(5,2), ib(3,4)/ia(4,3)\,.$$

Definition 6 (negation $\neg_1$ bf-EVALPSN)

Obviously the epistemic negation $\neg_1$ that maps bf-annotations to themselves is defined as follows:

$$\neg_1(af) = be,\ \neg_1(be) = af,\ \neg_1(da) = db,\ \neg_1(db) = da,\ \neg_1(ma) = mb,\ \neg_1(mb) = ma,$$

$$\neg_1(ja) = jb,\ \neg_1(jb) = ja,\ \neg_1(ia) = ib,\ \neg_1(ib) = ia,\ \neg_1(\bot_7) = \bot_7,\ \neg_1(\mathrm{T}_7) = \mathrm{T}_7.$$

Real-time Process Order Control/bf-EVALPSN

In this section, we show how the start/finish time of processes can be treated dynamically in bf-EVALPSN with a simple example in Figure 5, and introduce a real-time process order control in bf-EVALPSN with the brewery pipeline processes schedule in Figure 6. The details and basic ideas of the EVALPSN control based on safety verification has been presented in [5].

Example 1

Let us consider three processes $\mathrm{Pr}_0(id\ p0)$, $\mathrm{Pr}_1(id\ p1)$ and $\mathrm{Pr}_2(id\ p2)$, which are supposed to be processed according to the process schedule in Figure 5, and three bf-relations represented in the bf-EVALP clauses:

$$R(p0, p1, t):[(i_1, j_1), \alpha], \quad R(p1, p2, t):[(i_2, j_2), \alpha], \quad R(p2, p3, t):[(i_3, j_3), \alpha],$$

which are determined by each process start/finish information at time $t_0,...,t_7$. At time t_0, no process has started yet. Thus, we have no knowledge in terms of each bf-relation.

$$R(p0, p1, t_0):[(0,0), \alpha], \quad R(p1, p2, t_0):[(0,0), \alpha], \quad R(p2, p3, t_0):[(0,0), \alpha].$$

At time t_1, only Pr_0 has started before Pr_1 starts, and $R(p0, p1, t_1)$ has bf-annotation be(0,4). Because since one of the bf-relations, $db(0,7), mb(1,6), jb(2,5)\ and\ ib(3,4)$, could be the bf-relation between Pr_1 and Pr_2, thus, the greatest lower bound (0,4) of the set { (0,7), (1,6), (2,5), (3,4) } becomes the bf-annotation of $R(p0, p1, t_1)$. Other bf-literals have the bottom vector annotation $(0,0)$.

$$R(p0, p1, t_1):[(0,4), \alpha], \quad R(p1, p2, t_1):[(0,0), \alpha], \quad R(p2, p3, t_1):[(0,0), \alpha].$$

At time t_2, the second process Pr_1 also has started before Pr_0 finishes. Then, bf-relations jb(2,5) or ib(3,4) could be the bf-relation between Pr_0 and Pr_1. Thus, the greatest lower bound (2,4) of the set {(2, 5), (3,4)} has to be the vector annotation of $R(p0, p1, t_2)$. In addition, $R(p1, p2, t_2)$ has vector annotation (0,4) as well as $R(p0, p1, t_1)$ since Pr_1 has also started before Pr_2 starts. $R(p2, p3, t_2)$ has the bottom vector annotation (0,0) since Pr_2 has not started yet.

$$R(p0, p1, t_2):[(2,4), \alpha], \quad R(p1, p2, t_2):[(0,4), \alpha], \quad R(p2, p3, t_2):[(0,0), \alpha].$$

At time t_3, Pr_2 has started before both Pr_0 and Pr_1 finish. Then, $R(p0, p1, t_3)$ and $R(p1, p2, t_3)$ have the same vector annotation (2,4) as well as $R(p0, p1, t_2)$. Moreover, $R(p1, p2, t_3)$ has vector annotation (0,4) as well as $R(p0, p1, t1)$.

$$R(p0, p1, t_3):[(2,4), \alpha], \quad R(p1, p2, t_3):[(2,4), \alpha], \quad R(p2, p3, t_3):[(0,4), \alpha].$$

At time t_4, Pr_2 has finished before both Pr_0 and Pr_1 finish. Then, $R(p0, p1, t_4)$ still should have the same vector annotation (2,4) as well as the previous time t_3. In addition, $R(p1, p2, t_4)$ has its bf-annotation ib(3,4). There are still two alternatives for the bf-relation between Pr_2 and Pr_3:

(i) if Pr_3 starts immediately after Pr_2 finishes, $R(p2, p3, t4)$ has bf-annotation $mb(1,6)$;

(ii) if Pr_3 does not start immediately after Pr_2 finishes, $R(p2, p3, t_4)$ has bf-annotation $db(0,7)$.

Either way, we have only information that Pr_2 has just finished at time t_4, which can be represented in vector annotation (0, 6) that is the greatest lower bound of the set {(1,6), (0,7)}.

$$R(p0, p1, t_4):[(2,4),\alpha], \quad R(p1, p2, t_4):[(3,4),\alpha], \quad R(p2, p3, t_4):[(0,6),\alpha].$$

At time t_5, Pr_0 has finished before Pr_1 finishes. Then, $R(p0, p1, t_5)$ has bf-annotation $jb(2,5)$, and $R(p2, p3, t_5)$ also has bf-annotation $db(0,7)$ because Pr_3 has not started yet.

$$R(p0, p1, t_5):[(2,4),\alpha], \quad R(p1, p2, t_5):[(3,4),\alpha], \quad R(p2, p3, t_5):[(0,7),\alpha].$$

We note that all the three bf-relations can be determined at time t_5 before Pr_1 finishes and Pr_3 starts.

Example 2

We consider two brewery pipeline processes, brewery and pipeline cleaning ones that are scheduled in Figure 6. Each process $\mathrm{Pr}_i\,(i \in \{0,1,2,3\})$ has its process order safety property $SPR-i$ to be assured.

SPR-0 Pr_0 must start before any other processes,

SPR-1 Pr_1 must start immediately after Pr_0 starts,

SPR-2 Pr_2 must start immediately after Pr_1 finishes,

SPR-3 Pr_3 must start immediately after Pr_0 and Pr_2 finish.

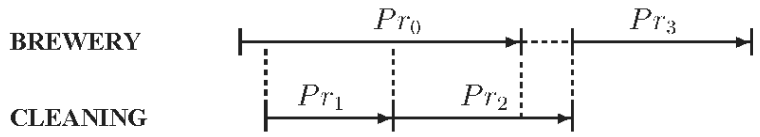


Fig. 6. Brewery Pipeline Process Schedule.

All the safety properties $SPR-0,1,2,3$ are translated into

$$\begin{aligned} &\sim R(p0, p1, t):[(4,0),\alpha] \wedge \sim R(p0, p2, t):[(4,0),\alpha] \wedge \sim R(p0, p3, t):[(4,0),\alpha] \\ &\rightarrow S(p0, t):[(0,1),\gamma], \\ &\sim S(p0, t):[(0,1),\gamma] \rightarrow S(p0, t):[(0,1),\beta], \end{aligned} \tag{1}$$

$$\begin{aligned} &R(p1, p0, t):[(4,0),\alpha] \rightarrow S(p1, t):[(0,1),\gamma], \\ &\sim S(p1, t):[(0,1),\gamma] \rightarrow S(p1, t):[(0,1),\beta], \end{aligned} \tag{2}$$

$$\begin{aligned} &R(p2, p1, t):[(6,0),\alpha] \wedge \sim R(p2, p1, t):[(7,0),\alpha] \rightarrow S(p2, t):[(0,1),\gamma], \\ &\sim St(p2, t):[(0,1),\gamma] \rightarrow S(p2, t):[(0,1),\beta], \end{aligned} \tag{3}$$

$$\begin{aligned} &R(p3, p0, t):[(6,0),\alpha] \wedge R(p3, p2, t):[(6,0),\alpha] \wedge \sim R(p3, p2, t):[(7,0),\alpha] \\ &\rightarrow S(p3, t):[(0,1),\gamma], \\ &\sim St(p3, t):[(0,1),\gamma] \rightarrow S(p3, t):[(0,1),\beta], \end{aligned} \tag{4}$$

where each EVALP literal $S(pi, t):[(0,1),\gamma/\beta]\ (i \in \{0,1,2,3\})$ denotes that it is permitted/forbidden to start the process Pr_i at time t.

Now, we show how the process order control is carried out in real-time by bf-EVALPSN programming. We consider variation of the vector annotations of bf-literals, $R(p0, p1, t)$, $R(p0, p2, t)$, $R(p0, p3, t)$, $R(p2, p1, t)$ and $R(p3, p2, t)$.

Stage 0 since no process has started at time t_0, the vector annotation of each bf-literals is (0,0). Then, by bf-EVALPSN clauses (1), (2), (3) and (4), we obtain $S(p0,t_0):[(0,1),\gamma], S(p1,t_0):[(0,1),\beta], S(p2,t_0):[(0,1),\beta]$ and $S(p3,t_0):[(0,1),\beta]$. Therefore, only Pr_0 is permitted for starting.

Stage 1 if both Pr_0 and Pr_1 have started but neither of them have finished yet at time t_1, then we obtain

$$R(p0,p1,t_1):[(2,4),\alpha], \quad R(p2,p1,t_1):[(4,0),\alpha],$$
$$R(p0,p2,t_1):[(0,4),\alpha], \quad R(p3,p2,t_1):[(0,0),\alpha],$$
$$R(p0,p3,t_1):[(0,4),\alpha].$$

Then, by (1), (2), (3) and (4), we obtain $S(p2,t_1):[(0,1),\beta]$ and $S(p3,t_1):[(0,1),\beta]$. Therefore, neither Pr_2 nor Pr_3 are permitted for starting.

Stage 2 if Pr_1 has just finished, Pr_2 has not started yet, and Pr_0 has not finished yet at time t_2, then we obtain

$$R(p0,p1,t_2):[(3,4),\alpha], \quad R(p2,p1,t_2):[(6,0),\alpha],$$
$$R(p0,p2,t_2):[(0,4),\alpha], \quad R(p3,p2,t_2):[(0,0),\alpha],$$
$$R(p0,p3,t_2):[(0,4),\alpha].$$

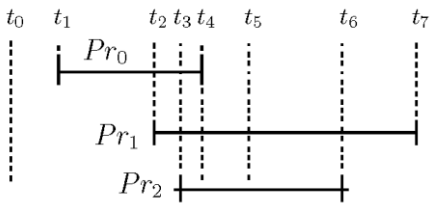


Fig. 7. Process Timing Chart 1.

Then, by (1), (2), (3) and (4), we obtain $S(p2,t_0):[(0,1),\gamma]$ and $S(p3,t_0):[(0,1),\beta]$. Therefore, Pr_2 has been permitted for starting, however Pr_3 is still for forbidden from starting.

Stage 3 if Pr_0 has just finished, Pr_3 has not started yet, and Pr_2 has not finished yet at time t_3, then we obtain

$$R(p0,p1,t_3):[(3,4),\alpha], \quad R(p2,p1,t_3):[(6,1),\alpha],$$
$$R(p0,p2,t_3):[(2,5),\alpha], \quad R(p3,p2,t_3):[(4,0),\alpha],$$
$$R(p0,p3,t_3):[(0,6),\alpha].$$

Then, by (1), (2), (3) and (4), we obtain $S(p3,t_0):[(0,1),\beta]$. Therefore, Pr_3 is still forbidden from starting because Pr_2 has not finished yet.

Stage 4 if Pr_2 has just finished and Pr_3 has not started continuously yet at time t_4, then we obtain

$$R(p0,p1,t_4):[(3,4),\alpha], \quad R(p2,p1,t_4):[(6,1),\alpha],$$
$$R(p0,p2,t_4):[(2,5),\alpha], \quad R(p3,p2,t_4):[(6,0),\alpha],$$
$$R(p0,p3,t_4):[(0,7),\alpha].$$

which represent the bf-relations between the processes. Then, by (1), (2), (3) and (4), we have $S(p3,t_0):[(0,1),\gamma]$. Therefore, Pr_3 is permitted for starting because both Pr_0 and Pr2 have finished.

Transitive Inference Rules in Bf-EVALPSN

We have introduced that safety verification based real-time process order control can be performed by simple integer computation in vector annotations by bf-EVALPSN programming. However, if we need to control all bf-relations between any two processes in real-time, it takes much computation time. In this section, we introduce the transitive reasoning of bf-relations in bf-EVALPSN, which can reduce process order control time. If we use such a reasoning system, the computation of 45 bf-relations can be reduced to that of just 9 neighbor bf-relations between Pr_0 and Pr_1, Pr_1 and Pr_2,…, Pr_8 and Pr_9. We introduce some transitive inference rules to reason the bf-annotation of $R(pi,pk,t)$ from the bf-annotations of $R(pi,pj,t)$ and $R(pj,pk,t)$ in real-time, which are called *bf-relation inference rules (bf-inf rules* for short), and show how to apply them to real-time process order control with a simple example.

Suppose that three processes, Pr_0, Pr_1 and Pr_2 are processed according to the three process time charts in Figure 7, 8, 9 in which only the start time of Pr_2 varies time t_3 to t_5 and no bf-relation varies. The vector annotations of $R(p0,p1,t)$, $R(p1,p2,t)$ and $R(p0,p2,t)$ at each time t_i are shown by the three timing charts in Table 1. If we consider the annotations at time $t_{1,2}$,

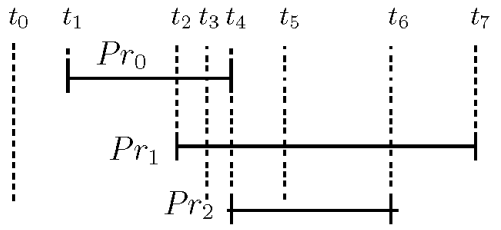


Fig. 8. Process Timing Chart 2.

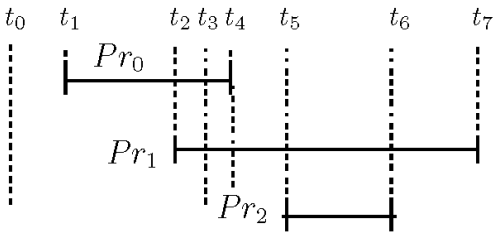


Fig. 9. Process Timing Chart 3.

we obtain the following bf-inf rule:

$$rule-1 \qquad R(p0,p1,t):[(0,4),\alpha] \rightarrow R(p0,p2,t):[(0,4),\alpha]. \tag{5}$$

Furthermore, if we also focus on the vector annotations at time $t_{3,4}$ in Table 1, we obtain the following bf-inf rules:

$$rule-2 \qquad R(p0,p1,t):[(2,4),\alpha] \wedge R(p1,p2,t):[(2,4),\alpha] \rightarrow R(p0,p2,t):[(2,4),\alpha], \tag{6}$$

$$rule-3 \quad R(p0, p1, t):[(2,5),\alpha] \wedge R(p1, p2, t):[(2,4),\alpha] \rightarrow R(p0, p2, t):[(2,5),\alpha]. \tag{7}$$

As well as *rule-2* and *rule-3,* we also obtain the following bf-inf rules by taking the vector annotations at time t_4 into account.

$$rule-4 \quad R(p0, p1, t):[(2,5),\alpha] \wedge R(p1, p2, t):[(2,4),\alpha] \rightarrow R(p0, p2, t):[(1,6),\alpha]. \tag{8}$$

$$rule-5 \quad R(p0, p1, t):[(2,5),\alpha] \wedge \sim R(p1, p2, t):[(2,4),\alpha] \wedge R(p1, p2, t):[(0,4),\alpha] \rightarrow R(p0, p2, t):[(0,7),\alpha]. \tag{9}$$

Among the above bf-inf rules, *rule-3(7)* and *rule-4(8)* have the same precedent (program clause body),

$$R(p0, p1, t):[(2,5),\alpha] \wedge R(p1, p2, t):[(2,4),\alpha],$$

and different consequents(program clause heads)

$$R(p0, p2, t):[(2,5),\alpha] \text{ and } R(p0, p2, t):[(1,6),\alpha].$$

Having the same precedent may cause duplicate application of *rule-3(7)* and *rule-4(8).* Obviously they cannot be uniquely applied without extra information. In order to avoid such duplicate application of bf-inf rules, we consider applicable orders of bf-inf rules.

order-1 *rule-1* $\rightarrow$ *rule-2* $\rightarrow$ *rule-3*

order-2 *rule-1* $\rightarrow$ *rule-4*

order-3 *rule-1* $\rightarrow$ *rule-5*

Table 1. Vector Annotations of Process Time Chart 1,2,3

chart 1	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
$R(p0,p1,t)$	$(0,0)$	$(0,4)$	$(2,4)$	$(2,4)$	$(2,5)$	$(2,5)$	$(2,5)$	$(2,5)$
$R(p1,p2,t)$	$(0,0)$	$(0,0)$	$(0,4)$	$(2,4)$	$(2,4)$	$(3,4)$	$(3,4)$	$(3,4)$
$R(p0,p2,t)$	$(0,0)$	$(0,4)$	$(0,4)$	$(2,4)$	$(2,5)$	$(2,5)$	$(2,5)$	$(2,5)$

chart 2	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
$R(p0,p1,t)$	$(0,0)$	$(0,4)$	$(2,4)$	$(2,4)$	$(2,5)$	$(2,5)$	$(2,5)$	$(2,5)$
$R(p1,p2,t)$	$(0,0)$	$(0,0)$	$(0,4)$	$(0,4)$	$(2,4)$	$(2,4)$	$(3,4)$	$(3,4)$
$R(p0,p2,t)$	$(0,0)$	$(0,4)$	$(0,4)$	$(0,4)$	$(1,6)$	$(1,6)$	$(1,6)$	$(1,6)$

chart 3	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
$R(p0,p1,t)$	$(0,0)$	$(0,4)$	$(2,4)$	$(2,4)$	$(2,5)$	$(2,5)$	$(2,5)$	$(2,5)$
$R(p1,p2,t)$	$(0,0)$	$(0,0)$	$(0,4)$	$(0,4)$	$(0,4)$	$(2,4)$	$(3,4)$	$(3,4)$
$R(p0,p2,t)$	$(0,0)$	$(0,4)$	$(0,4)$	$(0,4)$	$(0,7)$	$(0,7)$	$(0,7)$	$(0,7)$

As indicated in the above orders, *rule-3(7)* should be applied immediately after *rule-2(6),* on the other hand, *rule-4(8)* should be done immediately after *rule-1(5).* Thus, if we take the applicable orders *order-1,2,3,* into account, such confusions may be avoided. Actually, bf-inf rules are not complete, that is to say there exist some other cases in which bf-relations cannot be uniquely determined by bf-inf rules, although we will not address such topics in this paper.

We show a real-time application of *order-1,2,3* based on the chart 2 in Figure 8.

At time t_1,

only *rule-1(5)* can be applied, thus, $R(p0, p2, t_1):[(0,4),\alpha]$ is obtained.

At time $t_{2,3}$,

no bf-inf rule can be applied.

At time t_4,

both *rule-3(7)* and *rule-4(8)* can be applied, however, since there is no order such that "*rule-1* $\rightarrow$ *rule-3* $\rightarrow$ ..." in *order-1,2,3,* only *rule-4(8)* can be uniquely applied according to *order-2,* and $R(p0, p2, t_4):[(0,7),\alpha]$ is obtained.

Here we introduce another bf-inf rule. For example, suppose that three processes Pr_0, Pr_1 and Pr_2 have started sequentially, and only Pr_1 has finished at time t as shown in Figure 10. Then, two bf-relations between Pr_0, Pr_1 and Pr_1, Pr_2 have already determined, which are represented by the following bf-EVALP clauses with complete bf-annotations,

$$R(p0, p1, t):[(3,4),\alpha] \; and \; R(p1, p2, t):[(1,6),\alpha]. \tag{10}$$

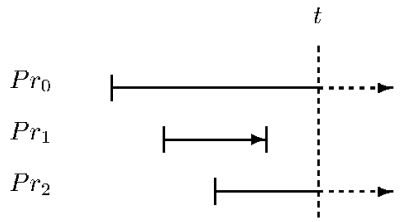


Fig. 10. Anticipation of bf-relation.

On the other hand, the bf-relation between Pr_0 and Pr_2 cannot be determined then. However, if we use the following bf-inf rule:

$$rule-6 \quad R(p0, p1, t):[(3,4),\alpha] \wedge R(p1, p2, t):[(2,5),\alpha] \rightarrow R(p0, p2, t):[(2,4),\alpha]. \tag{11}$$

the vector annotation (2,4) of $R(p0, p2, t)$ is derived. Moreover, since the vector annotation (2,4) is the greatest lower bound of the set, it is reasoned that the bf-relation between Pr_0 and Pr_2 must be either jb(2,5) or ib(3,4). As mentioned here, bf-relations can be reasoned from incomplete bf-annotations in bf-EVALPSN. Such anticipatory reasoning in bf-EVALPSN could be applied to safety verification or control.

Concluding remarks

In this paper, we have introduced bf-EVALPSN that can easily deal with before-after relations between processes dynamically, and shown how bf-EVALPSN can be applied to real-time process order control with simple examples. We have also briefly introduced that bf-EVALPSN has a dynamic method to reason bf-relations transitively.

As related original works, an interval temporal logic has been proposed by Allen et al. for knowledge representation of properties, actions and events[l]. In Allen's temporal logic, six predicates representing primitive bf-relations are used. Although Allen's temporal logic is a logically sophisticated tool to develop planning, natural language understanding and so on, it does not seem to be so suitable for real-time processing because bf-relations cannot be

determined until both two processes finish. On the other hand, in bf-EVALPSN process order control, vector annotations to represent bf-relations can be determined in real-time according to start/finish information of processes even if only one process has started. Moreover, it has been shown in [7] that EVALPSN can be implemented on a microchip as electronic circuits. Therefore, bf-EVALPSN is a more practical and suitable intelligent tool for real-time process order control.

As our future works, we are considering more practicai case studies to improve the bf-EVALPSN based process order control, especially applications of transitive reasoning by bf-inf rules[2].

References

1. Allen, J.F., Ferguson, G. – *Actions and Events in Interval Temporal Logic*. J.Logic and Computation vol.4, pp.531-579, 1994.
2. Blair,H.A., Subrahmanian,V.S. – *Paraconsistent Logic Programming*. Theoretical Computer Science vol.68, pp.135-154, 1989.
3. da Costa, N.C.A., Subrahmanian, V.S., Vago,C – *The Paraconsistent Logics PT*. Zeitschrift fur Mathematische Logic und Grundlangen der Mathematik vol.37, pp.139-148, 1991.
4. Nakamatsu,K., Abe,J.M., Suzuki,A.– *Annotated Semantics for Defeasible Deontic Reasoning*. Lecture Notes on Artificial Intelligence voi.2005, pp.432-440, Springer, 2001.
5. Nakamatsu,K. – *Intelligent Information Systems Based on Paraconsistent Logic*. In Innovations in Intelligent Systems and Applications Chap.11. Studies in Fuzziness and Soft Computing, vol. 140, pp.257-283, Springer-Verlag, 2004.
6. Nakamatsu,K.– *Pipeline Valve Control Based on EVALPSN Safety Verification*. J. Advanced Computational Intelligence and Intelligent Informatics, vol.10, pp.647-656, 2006.
7. Nakamatsu,K., Mita,Y., Shibata,T. – *An Intelligent Action Control System Based on Extended Vector Annotated Logic Program and its Hardware Implementation*. J. Intelligent Automation and Soft Computing, vol.13, pp.289-304, 2007.
8. Nakamatsu,K., Abe.J.M., Akama,S.– *Paraconsistent Before-after Relation Reasoning Based on EVALPSN. In New Directions in Intelligent Interactive Multimedia*. Studies in Computational Intelligence, vol.142, pp.265-274, Springer, 2008.

Program logic cu adnotare EVALPSN pentru procesele de reglare

Rezumat

Am propus deja cu alte ocazii un program logic paraconsistent denumit EVALPSN (Program Logic tip Vector Extins cu Negatie Puternica) și l-am aplicat către diferite sarcini de reglare, bazate pe verificarea siguranței. În această lucrare, un nou EVALPSN, denumit bf (înainte-după)-EVALPSN care poate trata relațiile tip înainte-după între două procese, este introdus. Iar aplicațiile lui pentru procesele de reglare desfășurate în timp real bazate pe verificarea siguranței sunt descrise împreună cu exemple simple. Mai mult, sunt prezentate anumite funcții utile ale bf-EVALPSN pentru reglarea în timp-real.

[2] This research is financially supported by Japanese Scientific Research Grant (C) No.20500074.