Review Paper: Paraconsistent Process Order Control

Kazumi Nakamatsu, Jair Minoro Abe and Seiki Akama

Abstract We have already proposed the paraconsistent process order control method based on an annotated logic program bf-EVALPSN. Bf-EVALPSN can deal with before-after relations between two processes (time intervals) in its annotations, and its reasoning system consists of two kinds of inference rules called the basic bf-inference rule and the transitive bf-inference rule. In this paper, we review how bf-EVALPSN can be applied to process order control with a simple example.

Keywords Paraconsistent annotated logic program · Bf-EVALPSN · Process order control

1 Introduction

We have already proposed the paraconsistent process order control method based on an annotated logic program bf-EVALPSN [4–6]. Bf-EVALPSN can deal with before-after relations between two processes(time intervals) in its annotations, and its reasoning system consists of two kinds of inference rules called the basic bf-inference rule and the transitive bf-inference rule. In this paper, we review how bf-EVALPSN can be applied to process order control with a simple example.

K. Nakamatsu (✉)
University of Hyogo, Himeji, Japan
e-mail: nakamatu@shse.u-hyogo.ac.jp

J.M. Abe
Paulista University, Sao Paulo, Brazil
e-mail: jairabe@uol.com.br

S. Akama
C-republic, Kawasaki, Japan
e-mail: akama@jcom.home.ne.jp

I. Czarnowski et al. (eds.), *Intelligent Decision Technologies 2016*,
Smart Innovation, Systems and Technologies 57,
DOI 10.1007/978-3-319-39627-9_20

2 Annotated Logic Program bf-EVALPSN

In this section a special version of EVALPSN, bf-EVALPSN [4, 5] that can deal with before-after relation between two processes are reviewed briefly. The details of EVALPSN can be found in [3, 4].

In bf-EVALPSN, a special annotated literal $R(p_m, p_n, t) : [(i, j), \mu]$ called *bf-literal* whose non-negative integer vector annotation (i, j) represents the before-after relation between processes Pr_m and Pr_n at time t is introduced. The integer components i and j of the vector annotation (i, j) represent the after and before degrees between processes $Pr_m(p_m)$ and $Pr_n(p_n)$, respectively, and before-after relations are represented paraconsistently in vector annotations.

Definition 1 An extended vector annotated literal, $R(p_i, p_j, t) : [(i, j), \mu]$ is called a bf-EVALP literal or a bf-literal for short, where (i, j) is a vector annotation and $\mu \in \{\alpha, \beta, \gamma\}$. If an EVALPSN clause contains bf-EVALP literals, 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.

We provide a paraconsistent before-after interpretation for vector annotations representing bf-relations in bf-EVALPSN, and such a vector annotation is called a *bf-annotation*. Exactly speaking, there are fifteen kinds of bf-relation according to before-after order between four start/finish times of two processes.

Before (`be`)/**After** (`af`) are defined according to the bf-relation between each start time of two processes. If one process has started before/after another one starts, then the bf-relations between them are defined as "before/after", which are represented as the left figure in Fig. 1.

Other kinds of bf-relations are shown as follows.

Disjoint Before (`db`)/**After** (`da`) are defined as there is a time lag between the earlier process finish time and the later one start time, which are described as the right figure in Fig. 1.

Immediate Before (`mb`)/**After** (`ma`) are defined as there is no time lag between the earlier process finish time and the later one start time, which are described as the left figure in Fig. 2.

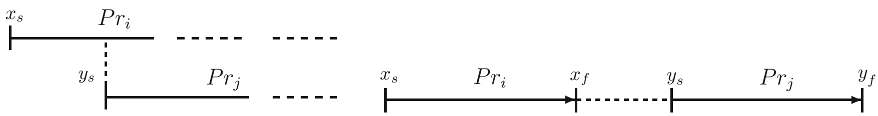
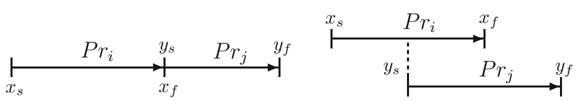


Fig. 1 Before (`be`)/After (`af`) and Disjoint Before (`db`)/After (`da`)

Fig. 2 Immediate Before (`mb`)/After (`ma`) and Joint Before (`jb`)/After (`ja`)



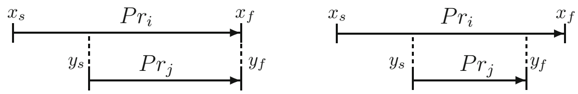


Fig. 3 S-included Before (sb)/After (sa) and Included Before (ib)/After (ia)

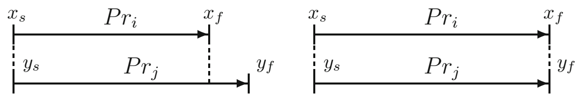


Fig. 4 F-included Before (fb)/After (fa) and Paraconsistent Before-after (pba)

Joint Before (jb)/**After** (ja) are defined as two processes overlap and the earlier process had finished before the later one finished, which are described as the right figure in Fig. 2.

S-included Before (sb), **S-included After** (sa) are defined as one process had started before another one started and they have finished at the same time, which are described as the left figure in Fig. 3.

Included Before (ib)/**After** (ia) are defined as one process had started/finished before/after another one started/finished, which are described as the right figure in Fig. 3.

F-included Before (fb)/**After** (fa) are defined as the two processes have started at the same time and one process had finished before another one finished, which are described as the left figure in Fig. 4.

Paraconsistent Before-after (pba) is defined as two processes have started at the same time and also finished at the same time, which is described as the right figure in Fig. 4.

The epistemic negation over bf-annotations, be, af, db, da, mb, ma, jb, ja, ib, ia, sb, sa, fb, fa, pba is defined and the complete lattice of bf-annotations is shown in Fig. 5.

Definition 2 The epistemic negation $\neg_1$ over the bf-annotations is obviously defined as the following mappings:

$$\neg_1(\texttt{af}) = \texttt{be}, \quad \neg_1(\texttt{be}) = \texttt{af}, \quad \neg_1(\texttt{da}) = \texttt{db}, \quad \neg_1(\texttt{db}) = \texttt{da}, \quad \neg_1(\texttt{ma}) = \texttt{mb},$$
$$\neg_1(\texttt{mb}) = \texttt{ma}, \quad \neg_1(\texttt{ja}) = \texttt{jb}, \quad \neg_1(\texttt{jb}) = \texttt{ja}, \quad \neg_1(\texttt{sa}) = \texttt{sb}, \quad \neg_1(\texttt{sb}) = \texttt{sa},$$
$$\neg_1(\texttt{ia}) = \texttt{ib}, \quad \neg_1(\texttt{ib}) = \texttt{ia}, \quad \neg_1(\texttt{fa}) = \texttt{fb}, \quad \neg_1(\texttt{fb}) = \texttt{fa}, \quad \neg_1(\texttt{pba}) = \texttt{pba}.$$

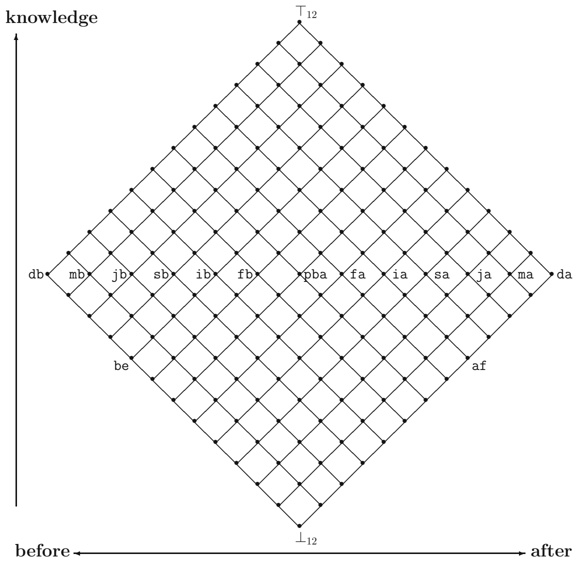


Fig. 5 The complete lattice $\mathcal{T}_v(12)_{bf}$ for Bf-annotations

3 Reasoning Systems in bf-EVALPSN

In order to represent the *basic bf-inference rule* two literals are introduced: $st(p_i, t)$: "process Pr_i starts at time t", and $fi(p_i, t)$: "process Pr_i finishes at time t". Those literals are used for expressing process start/finish information and may have one of the vector annotations, $\{\perp(0,0), \mathtt{t}(1,0), \mathtt{f}(0,1), \top(1,1)\}$.

A group of basic bf-inference rules named $(0,0)$-*rules* to be applied at the initial stage (time t_0) are introduced.

(0, 0)-rules Suppose that no process has started yet and the vector annotation of bf-literal $R(p_i, p_j, t)$ is $(0,0)$, which shows that there is no knowledge in terms of the bf-relation between processes Pr_i and Pr_j, then the following two basic bf-inference rules are applied at the initial stage.

(0, 0)-rule-1 If process Pr_i started before process Pr_j starts, then the vector annotation $(0,0)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{be}(0,8)$, which is the greatest lower bound of $\{\mathtt{db}(0,12), \mathtt{mb}(1,11), \mathtt{jb}(2,10), \mathtt{sb}(3,9), \mathtt{ib}(4,8)\}$.

(0, 0)-rule-2 If both processes Pr_i and Pr_j have started at the same time, then it is reasonably anticipated that the bf-relation between processes Pr_i and Pr_j will be one of the bf-annotations, $\{\mathtt{fb}(5,7), \mathtt{pba}(6,6), \mathtt{fa}(7,5)\}$ whose greatest lower bound is $(5,5)$ (refer to Fig. 5). Therefore, the vector annotation $(0,0)$ of bf-literal $R(p_i, p_j, t)$ should turn to $(5,5)$.

$(0,0)$-rule-1 and $(0,0)$-rule-2 are translated into the bf-EVALPSN,

$$R(p_i,p_j,t):[(0,0),\alpha] \wedge st(p_i,t):[\mathtt{t},\alpha] \wedge \sim st(p_j,t):[\mathtt{t},\alpha] \rightarrow R(p_i,p_j,t):[(0,8),\alpha]$$
$$R(p_i,p_j,t):[(0,0),\alpha] \wedge st(p_i,t):[\mathtt{t},\alpha] \wedge st(p_j,t):[\mathtt{t},\alpha] \rightarrow R(p_i,p_j,t):[(5,5),\alpha]$$

Suppose that $(0,0)$-rule-1 or 2 has been applied, then the vector annotation of bf-literal $R(p_i,p_j,t)$ should be one of $(0,8)$ or $(5,5)$. Therefore, we need to consider two groups of basic bf-inference rules to be applied for following $(0,0)$-rule-1 and 2, which are named $(0,8)$-*rules* and $(5,5)$-*rules*, respectively.

(0, 8)-rules Suppose that process Pr_i has started before process Pr_j, then the vector annotation of bf-literal $R(p_i,p_j,t)$ should be $(0,8)$. We obtain the following inference rules to be applied for $(0,0)$-rule-1.

(0, 8)-rule-1 If process Pr_i has finished before process Pr_j starts, and process Pr_j starts immediately after process Pr_i finished, then the vector annotation $(0,8)$ of bf-literal $R(p_i,p_j,t)$ should turn to $\mathtt{mb}(1,11)$.

(0, 8)-rule-2 If process Pr_i has finished before process Pr_j starts, and process Pr_j has not started immediately after process Pr_i finished, then the vector annotation $(0,8)$ of bf-literal $R(p_i,p_j,t)$ should turn to $\mathtt{db}(0,12)$.

(0, 8)-rule-3 If process Pr_j starts before process Pr_i finishes, then the vector annotation $(0,8)$ of bf-literal $R(p_i,p_j,t)$ should turn to $(2,8)$ that is the greatest lower bound of the set, $\{\mathtt{jb}(2,10),\ \mathtt{sb}(3,9),\ \mathtt{ib}(4,8)\}$.

$(0,8)$-rule-1, 2 and 3 are translated into the bf-EVALPSN,

$$R(p_i,p_j,t):[(0,8),\alpha] \wedge fi(p_i,t):[\mathtt{t},\alpha] \wedge st(p_j,t):[\mathtt{t},\alpha] \rightarrow R(p_i,p_j,t):[(1,11),\alpha]$$
$$R(p_i,p_j,t):[(0,8),\alpha] \wedge fi(p_i,t):[\mathtt{t},\alpha] \wedge \sim st(p_j,t):[\mathtt{t},\alpha] \rightarrow R(p_i,p_j,t):[(0,12),\alpha]$$
$$R(p_i,p_j,t):[(0,8),\alpha] \wedge \sim fi(p_i,t):[\mathtt{t},\alpha] \wedge st(p_j,t):[\mathtt{t},\alpha] \rightarrow R(p_i,p_j,t):[(2,8),\alpha]$$

(5, 5)-rules Suppose that both processes Pr_i and Pr_j have already started at the same time, then the vector annotation of bf-literal $R(p_i,p_j,t)$ should be $(5,5)$. We have the following inference rules to be applied for following $(0,0)$-rule-2.

(5, 5)-rule-1 If process Pr_i has finished before process Pr_j finishes, then the vector annotation $(5,5)$ of bf-literal $R(p_i,p_j,t)$ should turn to $\mathtt{sb}(5,7)$.

(5, 5)-rule-2 If both processes Pr_i and Pr_j have finished at the same time, then the vector annotation $(5,5)$ of bf-literal $R(p_i,p_j,t)$ should turn to $\mathtt{pba}(6,6)$.

(5, 5)-rule-3 If process Pr_j has finished before process Pr_i finishes, then the vector annotation $(5,5)$ of bf-literal $R(p_i,p_j,t)$ should turn to $\mathtt{sa}(7,5)$.

Basic bf-inference rules $(5,5)$-rule-1, 2 and 3 are translated into the following bf-EVALPSN,

$$R(p_i, p_j, t) : [(5,5), \alpha] \wedge fi(p_i, t) : [\mathtt{t}, \alpha] \wedge \sim fi(p_j, t) : [\mathtt{t}, \alpha] \rightarrow R(p_i, p_j, t) : [(5,7), \alpha]$$
$$R(p_i, p_j, t) : [(5,5), \alpha] \wedge fi(p_i, t) : [\mathtt{t}, \alpha] \wedge fi(p_j, t) : [\mathtt{t}, \alpha] \rightarrow R(p_i, p_j, t) : [(6,6), \alpha]$$
$$R(p_i, p_j, t) : [(5,5), \alpha] \wedge \sim fi(p_i, t) : [\mathtt{t}, \alpha] \wedge fi(p_j, t) : [\mathtt{t}, \alpha] \rightarrow R(p_i, p_j, t) : [(7,5), \alpha]$$

If one of $(0,8)$-rule-1, 2, $(5,5)$-rule-1, 2 and 3 has been applied, a final bf-annotation such as $\mathtt{jb}(2,10)$ between two processes should be derived. However, even if $(0,8)$-rule-3 has been applied, no bf-annotation could be derived. Therefore, a group of basic bf-inference rules named $(2,8)$-rules should be considered for following $(0,8)$-rule-3.

(2, 8)-rules Suppose that process Pr_i has started before process Pr_j starts and process Pr_j has started before process Pr_i finishes, then the vector annotation of bf-literal $R(p_i, p_j, t)$ should be $(2,8)$ and the following three rules should be considered.

(2, 8)-rule-1 If process Pr_i finished before process Pr_j finishes, then the vector annotation $(2,8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{jb}(2,10)$.

(2, 8)-rule-2 If both processes Pr_i and Pr_j have finished at the same time, then the vector annotation $(2,8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{fb}(3,9)$.

(2, 8)-rule-3 If process Pr_j has finished before Pr_i finishes, then the vector annotation $(2,8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{ib}(4,8)$.

Basic bf-inference rules $(2,8)$-rule-1, 2 and 3 are translated into the bf-EVALPSN,

$$R(p_i, p_j, t) : [(2,8), \alpha] \wedge fi(p_i, t) : [\mathtt{t}, \alpha] \wedge \sim fi(p_j, t) : [\mathtt{t}, \alpha] \rightarrow R(p_i, p_j, t) : [(2,10), \alpha]$$
$$R(p_i, p_j, t) : [(2,8), \alpha] \wedge fi(p_i, t) : [\mathtt{t}, \alpha] \wedge fi(p_j, t) : [\mathtt{t}, \alpha] \rightarrow R(p_i, p_j, t) : [(3,9), \alpha]$$
$$R(p_i, p_j, t) : [(2,8), \alpha] \wedge \sim fi(p_i, t) : [\mathtt{t}, \alpha] \wedge fi(p_j, t) : [\mathtt{t}, \alpha] \rightarrow R(p_i, p_j, t) : [(4,8), \alpha]$$

The application orders of all basic bf-inference rules are summarized in Table 1.

Suppose that there are three processes Pr_i,Pr_j and Pr_k starting sequentially, then we consider to derive the vector annotation of bf-literal $R(p_i, p_k, t)$ from those of

Table 1 Application orders of basic Bf-inference rules

Vector annotation	Rule	Vector annotation	Rule	Vector annotation	Rule	Vector annotation
$(0,0)$	rule-1	$(0,8)$	rule-1	$(0,12)$		
			rule-2	$(1,11)$		
			rule-3	$(2,8)$	rule-1	$(2,10)$
					rule-2	$(3,9)$
					rule-3	$(4,8)$
	rule-2	$(5,5)$	rule-1	$(5,7)$		
			rule-2	$(6,6)$		
			rule-3	$(7,5)$		

bf-literals $R(p_i, p_j, t)$ and $R(p_j, p_k, t)$ transitively. We describe the rules by vector annotations.

Transitive Bf-inference Rules

$\mathbf{TR0}\quad (0,0) \wedge (0,0) \rightarrow (0,0)$

$\mathbf{TR1}\quad (0,8) \wedge (0,0) \rightarrow (0,8)$

$\mathbf{TR1-1}\quad (0,12) \wedge (0,0) \rightarrow (0,12)$

$\mathbf{TR1-2}\quad (1,11) \wedge (0,8) \rightarrow (0,12)$

$\mathbf{TR1-3}\quad (1,11) \wedge (5,5) \rightarrow (1,11)$

$\mathbf{TR1-4}\quad (2,8) \wedge (0,8) \rightarrow (0,8)$

$\mathbf{TR1-4-1}\quad (2,10) \wedge (0,8) \rightarrow (0,12)$

$\mathbf{TR1-4-2}\quad (4,8) \wedge (0,12) \rightarrow (0,8)$

$\mathbf{TR1-4-3}\quad (2,8) \wedge (2,8) \rightarrow (2,8)$

$\mathbf{TR1-4-3-1}\quad (2,10) \wedge (2,8) \rightarrow (2,10)$

$\mathbf{TR1-4-3-2}\quad (4,8) \wedge (2,10) \rightarrow (2,8)$

$\mathbf{TR1-4-3-3}\quad (2,8) \wedge (4,8) \rightarrow (4,8)$

$\mathbf{TR1-4-3-4}\quad (3,9) \wedge (2,10) \rightarrow (2,10)$

$\mathbf{TR1-4-3-5}\quad (2,10) \wedge (4,8) \rightarrow (3,9)$

$\mathbf{TR1-4-3-6}\quad (4,8) \wedge (3,9) \rightarrow (4,8)$

$\mathbf{TR1-4-3-7}\quad (3,9) \wedge (3,9) \rightarrow (3,9)$

$\mathbf{TR1-4-4}\quad (3,9) \wedge (0,12) \rightarrow (0,12)$

$\mathbf{TR1-4-5}\quad (2,10) \wedge (2,8) \rightarrow (1,11)$

$\mathbf{TR1-4-6}\quad (4,8) \wedge (1,11) \rightarrow (2,8)$

$\mathbf{TR1-4-7}\quad (3,9) \wedge (1,11) \rightarrow (1,11)$

$\mathbf{TR1-5}\quad (2,8) \wedge (5,5) \rightarrow (2,8)$

$\mathbf{TR1-5-1}\quad (4,8) \wedge (5,7) \rightarrow (2,8)$

$\mathbf{TR1-5-2}\quad (2,8) \wedge (7,5) \rightarrow (4,8)$

$\mathbf{TR1-5-3}\quad (3,9) \wedge (5,7) \rightarrow (2,10)$

$\mathbf{TR1-5-4}\quad (2,10) \wedge (7,5) \rightarrow (3,9)$

$\mathbf{TR2}\quad (5,5) \wedge (0,8) \rightarrow (0,8)$

$\mathbf{TR2-1}\quad (5,7) \wedge (0,8) \rightarrow (0,12)$

$\mathbf{TR2-2}\quad (7,5) \wedge (0,12) \rightarrow (0,8)$

$\mathbf{TR2-3}\quad (5,5) \wedge (2,8) \rightarrow (2,8)$

$\mathbf{TR2-3-1}\quad (5,7) \wedge (2,8) \rightarrow (2,10)$

$\mathbf{TR2-3-2}\quad (7,5) \wedge (2,10) \rightarrow (2,8)$

$\mathbf{TR2-3-3}\quad (5,5) \wedge (4,8) \rightarrow (4,8)$

$\mathbf{TR2-3-4}\quad (7,5) \wedge (3,9) \rightarrow (4,8)$

$\mathbf{TR2-4}\quad (5,7) \wedge (2,8) \rightarrow (1,11)$

$\mathbf{TR2-5}\quad (7,5) \wedge (1,11) \rightarrow (2,8)$

$\mathbf{TR3}\quad (5,5) \wedge (5,5) \rightarrow (5,5)$

$\mathbf{TR3-1}\quad (7,5) \wedge (5,7) \rightarrow (5,5)$

$\mathbf{TR3-2}\quad (5,7) \wedge (7,5) \rightarrow (6,6)$

4 Process Order Control in Bf-EVALPSN

In this section, a simple example of the process order control is shown. The process order control method has the following steps: **step 1**, translate the safety properties of the process order control system into bf-EVALPSN; **step 2**, verify if permission for starting the process can be derived from the bf-EVALPSN in **step1** by the basic bf-inference rule and the transitive bf-inference rule or not.

We assume a pipeline system consists of two pipelines, PIPELINE-1 and 2, which deal with pipeline processes Pr_0, Pr_1, Pr_2 and Pr_3. The process schedule of those processes are shown in Fig. 6. Moreover, we assume that the pipeline system has four safety properties $SPR - i(i = 0, 1, 2, 3)$.

SPR−0 process Pr_0 must start before any other processes, and process Pr_0 must finish before process Pr_2 finishes,
SPR−1 process Pr_1 must start after process Pr_0 starts,
SPR−2 process Pr_2 must start immediately after process Pr_1 finishes,
SPR−3 process Pr_3 must start immediately after processes Pr_0 and Pr_2 finish.

Step 1. All safety properties $SPR - i(i = 0, 1, 2, 3)$ can be translated into the following bf-EVALPSN clauses.

$$SPR - 1$$

$$\sim R(p_0, p_1, t) : [(0, 8), \alpha] \rightarrow st(p_1, t) : [\mathtt{f}, \beta], \tag{1}$$

$$\sim R(p_0, p_2, t) : [(0, 8), \alpha] \rightarrow st(p_2, t) : [\mathtt{f}, \beta], \tag{2}$$

$$\sim R(p_0, p_3, t) : [(0, 8), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \tag{3}$$

$$st(p_1, t) : [\mathtt{f}, \beta] \wedge st(p_2, t) : [\mathtt{f}, \beta] \wedge st(p_3, t) : [\mathtt{f}, \beta] \rightarrow st(p_0, t) : [\mathtt{f}, \gamma], \tag{4}$$

$$\sim fi(p_0, t) : [\mathtt{f}, \beta] \rightarrow fi(p_0, t) : [\mathtt{f}, \gamma]. \tag{5}$$

$$SPR - 1$$

$$\sim st(p_1, t) : [\mathtt{f}, \beta] \rightarrow st(p_1, t) : [\mathtt{f}, \gamma], \tag{6}$$

$$\sim fi(p_1, t) : [\mathtt{f}, \beta] \rightarrow fi(p_1, t) : [\mathtt{f}, \gamma]. \tag{7}$$

$$SPR - 2$$

$$\sim R(p_2, p_1, t) : [(11, 0), \alpha] \rightarrow st(p_2, t) : [\mathtt{f}, \beta], \tag{8}$$

$$\sim st(p_2, t) : [\mathtt{f}, \beta] \rightarrow st(p_2, t) : [\mathtt{f}, \gamma], \tag{9}$$

$$\sim R(p_2, p_0, t) : [(10, 2), \alpha] \rightarrow fi(p_2, t) : [\mathtt{f}, \beta], \tag{10}$$

$$\sim fi(p_2, t) : [\mathtt{f}, \beta] \rightarrow fi(p_2, t) : [\mathtt{f}, \gamma]. \tag{11}$$

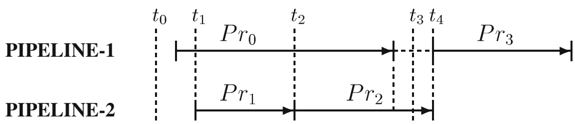


Fig. 6 Pipeline process schedule

$$SPR - 3$$

$$\sim R(p_3, p_0, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \quad (12)$$

$$\sim R(p_3, p_1, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \quad (13)$$

$$\sim R(p_3, p_2, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \quad (14)$$

$$\sim st(p_3, t) : [\mathtt{f}, \beta] \rightarrow st(p_3, t) : [\mathtt{f}, \gamma], \quad (15)$$

$$\sim fi(p_3, t) : [\mathtt{f}, \beta] \rightarrow fi(p_3, t) : [\mathtt{f}, \gamma]. \quad (16)$$

Step 2. Here, we show how the bf-EVALPSN process order safety verification is carried out at five time points, t_0, t_1, t_2 and t_3 in the process schedule (Fig. 6). We consider five bf-relations between processes Pr_0, Pr_1, Pr_2 and Pr_3 represented by the vector annotations of bf-literals,

$$R(p_0, p_1, t), \quad R(p_0, p_2, t), \quad R(p_0, p_3, t), \quad R(p_1, p_2, t), \quad R(p_2, p_3, t)$$

which should be verified based on safety properties $SPR - 0$, 1, 2 and 3 in real-time.

Initial Stage (at time t_0) no process has started at time t_0, thus, the bf-EVALP clauses,

$$R(p_i, p_j, t_0) : [(0, 0), \alpha], \quad \text{where } i = 0, 1, 2, \; j = 1, 2, 3 \quad (17)$$

are obtained by transitive bf-inference rule **TR0**; then, bf-EVALP clauses (17) satisfy each body of bf-EVALPSN clauses (1), (2) and (3), respectively, therefore, the forbiddance,

$$st(p_1, t_0) : [\mathtt{f}, \beta], \quad (18)$$

from starting each process $Pr_i (i = 1, 2, 3)$ is derived; moreover, since bf-EVALP clauses (18) satisfy the body of bf-EVALPSN clause (4), the permission for starting process Pr_0, $st(p_0, t_0) : [\mathtt{f}, \gamma]$ is derived; therefore, process Pr_0 is permitted for starting at time t_0.

2nd Stage (at time t_1) process Pr_0 has already started but all other processes $Pr_i (i = 1, 2, 3)$ have not started yet; then the bf-EVALP clauses,

$$R(p_0, p_1, t_1) : [(0, 8), \alpha], \quad (19)$$

are obtained, where the bf-EVALP clause (19) is derived by basic bf-inference rule $(0, 0)$-rule-1; moreover, the bf-EVALP clauses,

$$R(p_0, p_2, t_1) : [(0, 8), \alpha], \qquad R(p_0, p_3, t_1) : [(0, 8), \alpha] \quad (20)$$

are obtained by transitive bf-inference rule TR1; as bf-EVALP clause (19) does not satisfy the body of bf-EVALPSN clause (1), the forbiddance from starting process Pr_1, $st(p_1, t_1) : [\mathtt{f}, \beta]$ cannot be derived; then, since there does not exist the forbiddance, the body of bf-EVALPSN clause (6) is satisfied, and the permission for starting process Pr_1, $st(p_1, t_1) : [\mathtt{f}, \gamma]$ is derived; on the other hand, since bf-EVALP clauses (20) satisfy the body of bf-EVALPSN clauses (8) and (12) respectively, the forbiddance from starting both processes Pr_2 and Pr_3,

$$st(p_2, t_1) : [\mathtt{f}, \beta], \quad st(p_3, t_1) : [\mathtt{f}, \beta] \tag{21}$$

are derived; therefore, process Pr_1 is permitted for starting at time t_1.

3rd Stage (at time t_2) process Pr_1 has just finished and process Pr_0 has not finished yet; then, the bf-EVALP clauses,

$$R(p_0, p_1, t_2) : [(4, 8), \alpha], \quad R(p_1, p_2, t_2) : [(1, 11), \alpha], \quad R(p_2, p_3, t_2) : [(0, 8), \alpha] \tag{22}$$

are derived by basic bf-inference rules $(2, 8)$-rule-3, $(0, 8)$-rule-2 and $(0, 0)$-rule-1, respectively; moreover, the bf-EVALP clauses,

$$R(p_0, p_2, t_2) : [(2, 8), \alpha], \qquad R(p_0, p_3, t_2) : [(0, 12), \alpha] \tag{23}$$

are obtained by transitive bf-inference rules **TR1-4-6** and **TR1-2**, respectively; then, since bf-EVALP clause (22) does not satisfy the body of bf-EVALPSN clause (8), the forbiddance from starting process Pr_2, $st(p_2, t_2) : [\mathtt{f}, \beta]$ cannot be derived; since there does not exist the forbiddance, the body of bf-EVALPSN clause (9) is satisfied, and the permission for starting process Pr_2, $st(p_2, t_2) : [\mathtt{f}, \gamma]$ is derived; on the other hand, since bf-EVALP clauses (23) satisfy the body of bf-EVALPSN clause (12), the forbiddance from starting process Pr_3, $st(p_3, t_2) : [\mathtt{f}, \beta]$ is derived; therefore, process Pr_2 is permitted for starting, however process Pr_3 is still forbidden from starting at time t_2.

4th Stage (at the t_3) process Pr_0 has finished, process Pr_2 has not finished yet, and process Pr_3 has not started yet; then, the bf-EVALP clauses,

$$R(p_0, p_1, t_3) : [(4, 8), \alpha], \quad R(p_1, p_2, t_3) : [(1, 11), \alpha], \quad R(p_2, p_3, t_3) : [(0, 8), \alpha]$$

have been already reasoned at the previous stage; moreover, the bf-EVALP clauses,

$$R(p_0, p_2, t_3) : [(2, 10), \alpha], \qquad R(p_0, p_3, t_3) : [(0, 12), \alpha] \tag{24}$$

are obtained by basic bf-inference rule $(2, 8)$-rule-1; then, bf-EVALP clause (24) satisfies the body of bf-EVALP clause (14), and the forbiddance from starting process Pr_3, $S(p_3, t_3) : [\mathtt{f}, \beta]$ is derived; therefore, process Pr_3 is still forbidden to start because process Pr_2 has not finished yet at time t_3.

5 Concluding Remarks

In this paper, we have reviewed the process order control method based on a paraconsistent annotated logic program bf-EVALPSN, which can deal with before-after relation between processes with a small process order safety verification example.

References

1. Blair, H.A., Subrahmanian, V.S.: Paraconsistent logic programming. Theoret. Comput. Sci. **68**, 135–154 (1989)
2. da Costa, N.C.A., et al.: The Paraconsistent logics P$\mathcal{T}$. Zeitschrift für Mathematische Logic und Grundlangen der Mathematik **37**, 139–148 (1989)
3. Nakamatsu, K., et al.: Annotated Semantics for Defeasible Deontic Reasoning. LNAI, vol. **2005**, pp. 432–440. Springer (2001)
4. Nakamatsu, K., Abe, J.M.: The development of paraconsistent annotated logic program. Int. J. Reasoning-Based Intell. Syst. **1**, 92–112 (2009)
5. Nakamatsu, K., Abe, J.M., Akama, S.: A logical reasoning system of process before-after relation based on a paraconsistent annotated logic program bf-EVALPSN. J. Knowl. Based Intell. Eng. Syst. **15**, 145–163 (2011)
6. Nakamatsu, K., Abe, J.M.: The paraconsistent process order control method. Vietnam J. Comput. Sci. **1**, 29–37 (2014)