FACTA UNIVERSITATIS
Series: **Elec. Energ.** Vol. 26, No 1, April 2013, pp. 11 – 30
DOI: 10.2298/FUEE1301011N

APPLICATION OF PARACONSISTENT ANNOTATED LOGIC PROGRAM BF-EVALPSN TO INTELLIGENT CONTROL

Kazumi Nakamatsu

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

Abstract. A paraconsistent annotated logic program called EVALPSN has been developed for dealing with defeasible deontic reasoning and plausible reasoning, and applied to various kinds of intelligent control and safety verification. Moreover, in order to deal with before-after relation between processes(time intervals), bf(before-after)-EVALPSN has also been developed recently. In this paper, we review the reasoning system for before-after relation between processes based on bf-EVALPSN. The system consists of two groups of inference rules in bf-EVALPSN called basic and transitive bf-inference rules. The application of the reasoning system to real-time process order control is introduced with simple examples.

Key words: before-after relation, EVALPSN, bf-EVALPSN, annotated logic program, reasoning system.

I. INTRODUCTION

It has already passed over two decades since paraconsistent annotated logic and its logic programming have been developed [3], [4]. Based on the original annotated logic program we have developed four kinds of paraconsistent annotated logic program, ALPSN (Annotated Logic Program with Strong Negation) that can deal with some non-monotonic reasonings such as default reasoning [5], VALPSN (Vector ALPSN) that can deal with defeasible and plausible reasonings [6], EVALPSN (Extended VALPSN) that can deal with defeasible deontic and plausible reasonings [7], [17], and bf(before-after)-EVALPSN that can deal with before-after relation between processes(time intervals) recently [16], [18]. Those annotated logic programs have been applied to various kinds of intelligent control and safety verification, railway interlocking safety verification [9], air traffic safety verification [8], traffic signal control [10], discrete event control [11], robot action control [15], pipeline valve control [14], real-time process order control [13], and so on. Moreover, it has been shown that EVALPSN can be implemented on microchips as electronic circuits, which implies that EVALPSN is suitable for real-time control [15].

In this paper, we review the reasoning system for process before-after relations aiming efficient real-time process order control and safety verification in bf-EVALPSN [18]. The proposed before-after relation reasoning system consists of two groups of inference rules called *basic bf-inference rule* and *transitive bf-inference rule*, both of which can be represented in bf-EVALPSN.

In bf-EVALPSN, a special annotated literal $R(p_m, p_n, t) : [(i, j), \mu]$ called *bf-literal* whose non-negative integer vector

Accepted November 1, 2012

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 and Pr_n, and before-after relations are represented in vector annotations paraconsistently.

In the proposed reasoning system, the basic bf-inference rules are used for determining the vector annotation of a bf-literal in real-time according to the start/finish time information of two processes; on the other hand, the transitive bf-inference rules are used for determining the vector annotation of a bf-literal in real-time based on the vector annotations of two related bf-literals as follows. Suppose that there are three processes, Pr_0, Pr_1 and Pr_2 starting in sequence, then the before-after relation between processes Pr_0 and Pr_2 can be determined from the before-after relation between processes Pr_0 and Pr_1, and that between processes Pr_1 and Pr_2. Such process before-after relation reasoning is also formalized as transitive bf-inference rules in bf-EVALPSN. The transitive bf-inference system can contribute to the reduction of using times of basic bf-inference rules and it is a unique remarkable feature of the proposed system. Suppose that there is a bf-EVALPSN process order control system dealing with ten processes, Pr_0, Pr_1, $\cdots$ and Pr_9 starting in sequence. Without transitive bf-inference rules, the system has to deal with ${}_{10}C_2 = 45$ before-after relations independently by basic bf-inference rules. However, if we use transitive bf-inference rules, just nine before-after relations between processes Pr_i and Pr_{i+1} $(i = 0, 1, 2, \cdots, 8)$ should be determined by basic bf-inference rules, and the rest before-after relations could be determined based on the nine before-after relations by using transitive bf-inference rules.

This review paper is organized in the following manner: first, EVALPSN is reviewed briefly, and bf-EVALPSN is defined in details; next, it is shown how to reason before-after relations in bf-EVALPSN with a simple example of process order control, and basic bf-inference rules and transitive bf-inference rules are introduced; furthermore, a simple practical process order verification system is provided as an example; last, a related work of treating before-after relation of time intervals in a logical system and our future work are introduced as the conclusion.

II. EVALPSN

In this section, we review EVALPSN briefly [7]. Generally, a truth value called an *annotation* is explicitly attached to each literal in annotated logic programs [3]. 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 the complete lattice,

$$\mathcal{T}_v(n) = \{\ (x, y) | 0 \leq x \leq n, 0 \leq y \leq n, x, y, n \text{ are integers } \}$$

in Figure1. The ordering($\preceq_v$) of $\mathcal{T}_v(n)$ is defined as : let (x_1, y_1), $(x_2, y_2) \in \mathcal{T}_v(n)$,

$$(x_1, y_1) \preceq_v (x_2, y_2) \text{ iff } x_1 \leq x_2 \text{ and } y_1 \leq y_2.$$

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 one. The second component μ is an index of fact and deontic notions such as obligation, and the set of the second components constitutes the complete lattice,

$$\mathcal{T}_d = \{\bot, \alpha, \beta, \gamma, *_1, *_2, *_3, \top\}.$$

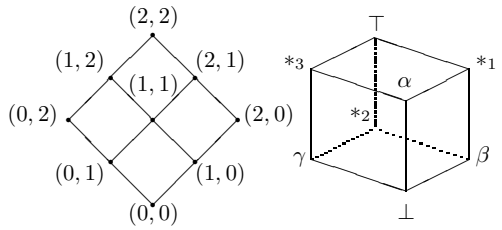

Fig. 1: Lattice $\mathcal{T}_v(2)$ and Lattice $\mathcal{T}_d$

The ordering($\preceq_d$) of $\mathcal{T}_d$ is described by the Hasse's diagram in Fig.1.

The intuitive meaning of each member of $\mathcal{T}_d$ is

$\bot$ (unknown), α (fact), β (obligation),
γ (non-obligation), $*_1$ (fact and obligation),
$*_2$ (obligation and non-obligation),
$*_3$ (fact and non-obligation), $\top$ (inconsistency).

Then the complete lattice $\mathcal{T}_e(n)$ of extended vector annotations is defined as the product $\mathcal{T}_v(n) \times \mathcal{T}_d$. The ordering($\preceq_e$) of $\mathcal{T}_e(n)$ is defined as : let $[(i_1, j_1), \mu_1]$ and $[(i_2, j_2), \mu_2] \in \mathcal{T}_e$,

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

There are two kinds of *epistemic negation* ($\neg_1$ and $\neg_2$) in EVALPSN, both of which are defined as mappings over $\mathcal{T}_v(n)$ and $\mathcal{T}_d$, respectively.

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

$$\neg_1([(i, j), \mu]) = [(j, i), \mu], \quad \forall \mu \in \mathcal{T}_d,$$
$$\neg_2([(i, j), \bot]) = [(i, j), \bot], \quad \neg_2([(i, j), \alpha]) = [(i, j), \alpha],$$
$$\neg_2([(i, j), \beta]) = [(i, j), \gamma], \quad \neg_2([(i, j), \gamma]) = [(i, j), \beta],$$
$$\neg_2([(i, j), *_1]) = [(i, j), *_3], \quad \neg_2([(i, j), *_2]) = [(i, j), *_2],$$
$$\neg_2([(i, j), *_3]) = [(i, j), *_1], \quad \neg_2([(i, j), \top]) = [(i, j), \top].$$

If we regard the epistemic negations as syntactical operations, the epistemic negations followed by literals can be eliminated by the syntactical operations. For example,

$$\neg_1(p : [(2, 0), \alpha]) = p : [(0, 2), \alpha] \quad \text{and}$$
$$\neg_2(q : [(1, 0), \beta]) = p : [(1, 0), \gamma].$$

There is another negation called *strong negation* ($\sim$) in EVALPSN, and it is treated as well as classical negation.

Definition 2(strong negation $\sim$) (see [4]) 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 (well extended vector annotated literal) Let p be a literal.

$$p{:}[(i,0),\mu] \quad \text{and} \quad p{:}[(0,j),\mu]$$

are called *well extended vector annotated literals*, where $i, j \in \{1, 2, \cdots, n\}$, and $\mu \in \{\ \alpha,\ \beta,\ \gamma\ \}$.

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

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

is called an *EVALPSN clause*. An *EVALPSN* is a finite set of EVALPSN clauses. Here we comment that if the annotations α and β represent fact and obligation, notions "fact", "obligation", "forbiddance" and "permission" can be represented by extended vector annotations, $[(m,0),\alpha]$, $[(m,0),\beta]$, $[(0,m),\beta]$, and $[(0,m),\gamma]$, respectively in EVALPSN, where m is a non-negative integer.

III. BEFORE-AFTER EVALPSN

In this section, we review bf-EVALPSN that can deal with before-after relations between two processes(time intervals). The reasoning system in bf-EVALPSN consists of two kinds of inference rules called *basic bf-inference rule* and *transitive bf-inference rule*, which will be introduced with some simple examples of real-time process order control in the following sections. 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 in vector annotations paraconsistently. In the reasoning system, the basic bf-inference rules are used for determining the vector annotation of a bf-literal in real-time according to the start/finish time information of two processes. On the other hand, the transitive bf-inference rule is used for determining the vector annotation of a bf-literal in real-time based on the vector annotations of two related bf-literals as follows. Suppose that there are three processes, Pr_0, Pr_1 and Pr_2 starting in sequence, then the before-after relation between processes Pr_0 and Pr_2 can be determined from two before-after relations between processes Pr_0 and Pr_1, and between processes Pr_1 and Pr_2. Such process before-after relation reasoning is also formalized as transitive bf-inference rules in bf-EVALPSN. The transitive bf-inference system can contribute to reduce using times of basic bf-inference rules and it is a unique remarkable feature of the reasoning system.

Suppose that there is a process order control system dealing with ten processes, Pr_0, Pr_1, $\cdots$ and Pr_9 starting in sequence. Without transitive bf-inference rules, the system has to deal with ${}_{10}C_2 = 45$ before-after relations independently by basic bf-inference rules. However, if we use transitive bf-inference rules, just nine before-after relations between processes Pr_i and Pr_{i+1} $(i = 0, 1, 2, \cdots, 8)$ need to be determined by basic bf-inference rules, and the rest before-after relations could be determined based on the nine before-after relations by using transitive bf-inference rules.

For example, the before-after relation of processes Pr_1 and Pr_4 is inferred from two before-after relations between processes Pr_1 and Pr_3, and between processes Pr_3 and Pr_4 by transitive bf-inference rules; moreover, the before-after relation between processes Pr_1 and Pr_3 is inferred

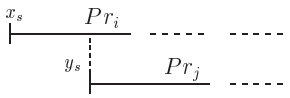

Fig. 2: Bf-relations Before(be)/After(af)

Fig. 3: Bf-relations Disjoint Before(db)/After(da)

from two before-after relations between processes Pr_1 and Pr_2, and between processes Pr_2 and Pr_3 by transitive bf-inference rules.

We introduce bf(before-after)-EVALPSN that can deal with before-after relations between two processes. Hereafter, the word "**b**efore-a**f**ter" is abbreviated as just "bf".

A particular literal $R(p_i, p_j, t)$ whose vector annotation represents the bf-relation between processes $Pr_i(p_i)$ and $Pr_j(p_j)$ is introduced, which declares the bf-relation between the processes

Definition 5(bf-EVALPSN)
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 *bf-annotations*. Exactly speaking, bf-relations are classified into fifteen meaningful kinds according to bf-relations between each start/finish time of two processes in bf-EVALPSN. First of all, we define the most basic bf-relations in bf-EVALPSN.

Before (be)/After (af)
Bf-relations *before*/*after* are defined according to the bf-relation between each start time of two processes, which are represented by bf-annotations be/af, respectively. Suppose that there are two processes, Pr_i with its start time x_s and finish time x_f, and Pr_j with its start time y_s and finish time y_f. If one process has started before/after another one starts, then the bf-relations between them are defined as "before(be)/after(af)", respectively. They are described by the process time chart in Fig.2 with the condition that process Pr_i has started before process Pr_j starts.

We introduce other kinds of bf-relations as well as before(be)/after(af). The original idea of the classification of process before-after relations has introduced in [1]

Disjoint Before (db) /After (da)
Bf-relations *disjoint before*/*after* between two processes are represented by bf-annotations db/da, respectively. The expressions "disjoint before/after" imply that there is a time lag between the earlier process finish time and the later one start time. They also are described by the process time chart in Fig.3.

Immediate Before (mb)/After (ma)
Bf-relations *immediate before*/*after* between two processes are represented by bf-annotations mb/ma, respectively. The expressions "immediate before/after" imply that there is no time lag between the earlier process finish time and the later one start time. The bf-relations are also described by the process time chart in Fig.4.

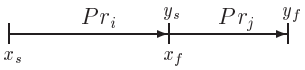

Fig. 4: Bf-relations Immediate Before(mb)/After(ma)

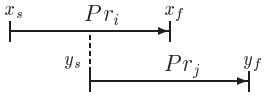

Fig. 5: Bf-relations, Joint Before/After

Joint Before (jb)/After (ja)
Bf-relations *joint before*/*after* between two processes are represented by bf-annotations jb/ja, respectively. The expressions "joint before/after" imply that the two processes overlap and the earlier process had finished before the later one finished. The bf-relations are also described by the process time chart in Fig.5.
S-included Before (sb), S-included After (sa)

Bf-relations *s-included before*/*after* between two processes are represented by bf-annotations sb/sa, respectively. The expressions "s-included before/after" imply that one process had started before another one started and they have finished at the same time. The bf-relations are also described by the process time chart in Fig.6.
Included Before (ib)/After (ia)
Bf-relations *included before*/*after* between processes Pr_i and Pr_j are represented by bf-annotations ib/ia, respectively. The expressions "included before/after" imply that one process had started/finished before/after another one started/finished, respectively. The bf-relations are also described by the process time chart in Fig.7.
F-included Before (fb)/After (fa)
bf-relations *f-include before*/*after* between two processes Pr_i and Pr_j are represented by bf-annotations fb/fa, respectively. The expressions "f-included before/after" imply that the two processes have started at the same time and one process had finished before another one finished.

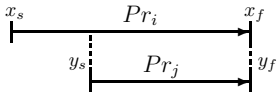

Fig. 6: Bf-relations S-included Before(sb)/After(sa)

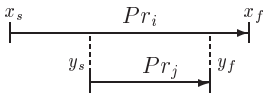

Fig. 7: Bf-relations Included Before(ib)/After(ia)

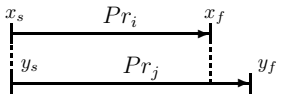

Fig. 8: Bf-relations F-included Before(fb)/After(fa)

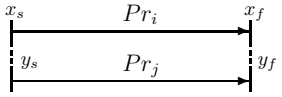

Fig. 9: Bf-relation, Paraconsistent Before-after

The bf-relations are also described by the process time chart in Fig.8.

Paraconsistent Before-after (pba)

Bf-relation *paraconsistent before-after* between two processes Pr_i and Pr_j is represented by bf-annotation pba. The expression "paraconsistent before-after" implies that two processes have started at the same time and also finished at the same time. The bf-relation is described by the process time chart in Fig.9.

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.10.

Definition 6 (Epistemic Negation $\neg_1$ for Bf-annotations) The epistemic negation $\neg_1$ over the bf-annotations

$$\{\texttt{be}, \texttt{af}, \texttt{da}, \texttt{db}, \texttt{ma}, \texttt{mb}, \texttt{ja}, \texttt{jb}, \texttt{sa}, \texttt{sb}, \texttt{ia}, \texttt{ib}, \texttt{fa}, \texttt{fb}, \texttt{pba}\}$$

is obviously defined as the following mappings :

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

We note that a bf-EVALP literal $R(pi, pj, t) : [\mu_1, \mu_2]$, where $\mu_1 \in \{$ mb, jb, sb, ib, fb, pba, fa, ia, sa, jb, ma $\}$ and $\mu_2 \in \{\alpha, \beta, \gamma\}$, would not be well annotated if $m \neq 0$ and $n \neq 0$, however, since the bf-literal is equivalent to the following two well annotated bf-literals:

$$R(p_i, p_j, t) : [(m, 0), \mu] \qquad \text{and} \qquad R(p_i, p_j, t) : [(0, n), \mu],$$

such a non-well annotated bf-EVALP literal can be dealt with as the conjunction of two well annotated bf-EVALP literals. For example, suppose a non-well annotated bf-EVALP clause

$$R(p_i, p_j, t_0) : [(m, n), \mu_0] \rightarrow R(p_i, p_j, t_1) : [(k, l), \mu_1],$$

where $k \neq 0,\ l \neq 0,\ m \neq 0$ and $n \neq 0$. It can be equivalently transformed into two well annotated

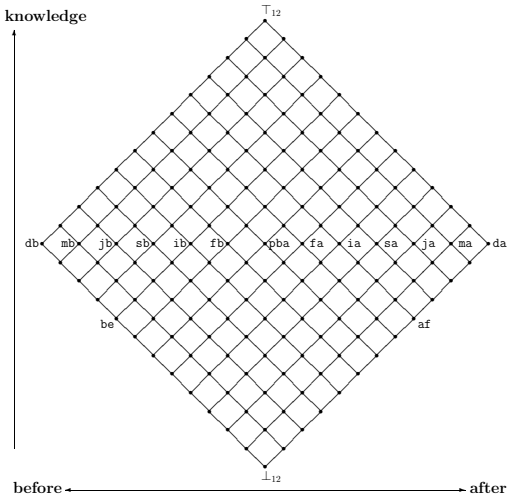

Fig. 10: The Complete Lattice $\mathcal{T}_v(12)_{bf}$ of Bf-annotations

bf-EVALP clauses,

$$R(p_i,p_j,t_0):[(m,0),\mu_0] \wedge R(p_i,p_j,t_0):[(0,n),\mu_0] \\ \quad \rightarrow R(p_i,p_j,t_1):[(k,0),\mu_1], \\ R(p_i,p_j,t_0):[(m,0),\mu_0] \wedge R(p_i,p_j,t_0):[(0,n),\mu_0] \\ \quad \rightarrow R(p_i,p_j,t_1):[(0,l),\mu_1].$$

IV. Reasoning system in bf-EVALPSN

In this section, we introduce the reasoning system in bf-EVALPSN, which consists of two kinds of inference rules, basic inference rules for reasoning bf-relations by process start/finish times and transitive inference rules for reasoning one bf-relation from two other bf-relations transitively.

A. Examples of bf-relation reasoning

In order to introduce the basic inference rule we show a simple example of bf-relation reasoning. Suppose that processes Pr_0, Pr_1 and Pr_2 are scheduled to be processed according to the time chart in Fig.11, then we show how the bf-relations between those processes are reasoned at each time $t_i (0 \leq i \leq 7)$.

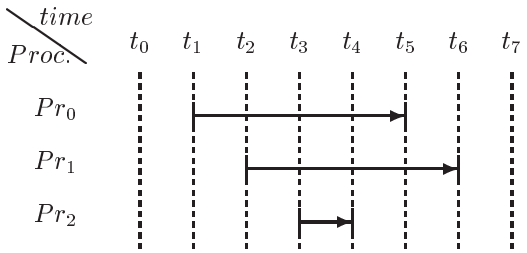

Fig. 11: Process Schedule Chart

At time t_0, no process has started, thus we have no knowledge in terms of the bf-relations between processes $Pr_0(p_0)$, $Pr_1(p_1)$ and $Pr_2(p_2)$. We have the bf-EVALP clauses,

$$R(p_0, p_1, t_0) : [(0, 0), \alpha], \quad R(p_1, p_2, t_0) : [(0, 0), \alpha],$$
$$R(p_0, p_2, t_0) : [(0, 0), \alpha].$$

At time t_1, only process Pr_0 has started, then it is obviously reasoned that the bf-relation between processes Pr_0 and Pr_1 will be one of the bf-annotations, $\mathtt{db}(0, 12)$, $\mathtt{mb}(1, 11)$, $\mathtt{jb}(2, 10)$, $\mathtt{sb}(3, 9)$, $\mathtt{ib}(4, 8)$ whose greatest lower bound is $(0, 8)$ (refer to Figure 10). On the other hand, we still have no knowledge in terms of the bf-relation between processes Pr_1 and Pr_2, thus the vector annotation of bf-literal $R(p_1, p_2, t_1)$ still remains $(0, 0)$. Obviously bf-literal $R(p_0, p_2, t_1)$ has the same vector annotation $\mathtt{be}(0, 8)$ as that of bf-literal $R(p_0, p_1, t_1)$, and we have the bf-EVALP clauses,

$$R(p_0, p_1, t_1) : [(0, 8), \alpha], \quad R(p_1, p_2, t_1) : [(0, 0), \alpha],$$
$$R(p_0, p_2, t_1) : [(0, 8), \alpha].$$

At time t_2, process Pr_1 has started before process Pr_0 finishes, then it is obviously reasoned that the bf-relation between processes Pr_0 and Pr_1 will be one of the bf-annotations, $\mathtt{jb}(2, 10)$, $\mathtt{sb}(3, 9)$, $\mathtt{ib}(4, 8)$ whose greatest lower bound is $(2, 8)$ (refer to Fig.10). As process Pr_2 has not started yet, the vector annotation of bf-literal $R(p_1, p_2, t_2)$ turns to $(0, 8)$ from $(0, 0)$ as well as that of bf-literal $R(p_0, p_2, t_1)$ and the vector annotation of bf-literal $R(p_0, p_2, t_2)$ still remains $(0, 8)$.

Therefore, we have the bf-EVALP clauses,

$$R(p_0, p_1, t_2) : [(2, 8), \alpha], \quad R(p_1, p_2, t_2) : [(0, 8), \alpha],$$
$$R(p_0, p_2, t_2) : [(0, 8), \alpha].$$

At time t_3, process Pr_2 has started before processes Pr_0 and Pr_1 finish, then the vector annotation of bf-literal $R(p_0, p_1, t_3)$ still remains $(2, 8)$, and the vector annotation $(0, 8)$ of both bf-literals $R(p_1, p_2, t_3)$ and $R(p_0, p_2, t_3)$ turns to $(2, 8)$ as well as that of bf-literal $R(p_0, p_1, t_2)$. We have the bf-EVALP clauses,

$$R(p_0, p_1, t_3) : [(2, 8), \alpha], \quad R(p_1, p_2, t_3) : [(2, 8), \alpha],$$
$$R(p_0, p_2, t_3) : [(2, 8), \alpha].$$

At time t_4, only process Pr_2 has finished before processes Pr_0 or Pr_1 finish, then bf-literals $R(p_1, p_2, t_4)$ and $R(p_0, p_2, t_4)$ have the same bf-annotation $\mathtt{ib}(4, 8)$. On the other hand, the vector annotation of bf-literal $R(p_1, p_2, t_4)$ still remains $(2, 8)$. We have the bf-EVALP clauses,

$$R(p_0, p_1, t_4) : [(2, 8), \alpha], \quad R(p_1, p_2, t_4) : [\mathtt{ib}(4, 8), \alpha],$$
$$R(p_0, p_2, t_4) : [\mathtt{ib}(4, 8), \alpha].$$

At time t_5, process Pr_0 has finished before process Pr_1 finishes, then bf-literal $R(p_0, p_1, t_5)$ has bf-annotation $\mathtt{jb}(2, 10)$. Even though process Pr_1 has not finished yet, all the bf-relations between processes Pr_0, Pr_1 and Pr_2 have been determined as follows:

$$R(p_0, p_1, t_5) : [\mathtt{jb}(2, 10), \alpha], \quad R(p_1, p_2, t_5) : [\mathtt{ib}(4, 8), \alpha],$$
$$R(p_0, p_2, t_5) : [\mathtt{ib}(4, 8), \alpha].$$

B. Basic before-after inference rule

Now we construct basic bf-relation inference rules called *basic bf-inference rules* with referring to the example in Section IV-A.

In order to represent the basic bf-inference rules in bf-EVALPSN, we newly introduce two more literals:

$st(p_i, t)$, which is intuitively interpreted that process Pr_i starts at time t, and
$fi(p_i, t)$, which is intuitively interpreted that process Pr_i finishes at time t,

which are used for expressing process start/finish information and may have one of the vector annotations, $(0, 0)$, $\mathtt{t}(1, 0)$, $\mathtt{f}(0, 1)$, $(1, 1)$, where annotations $\mathtt{t}$ and $\mathtt{f}$ can be intuitively interpreted as "true" and "false", respectively. Firstly, we show a group of basic bf-inference rules to be applied at the initial stage (time t_0) for bf-relation reasoning, which are called $(0, 0)$-*rules*.

(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 bf-annotation $\mathtt{be}(0, 8)$, which is the greatest lower bound of the bf-annotations, $\mathtt{db}(0, 12)$, $\mathtt{mb}(1, 11)$, $\mathtt{jb}(2, 10)$, $\mathtt{sb}(3, 9)$, $\mathtt{ib}(4, 8)$.

$(0, 0)$-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 $(5, 5)$ (see Fig.10). Therefore, the vector annotation $(0, 0)$ of bf-literal $R(p_i, p_j, t)$ should turn to $(5, 5)$.

Basic bf-inference rules $(0, 0)$-rule-1 and 2 may be translated into the bf-EVALPSN clauses,

$$R(p_i, p_j, t) : [(0, 0), \alpha] \wedge st(p_i, t) : [\mathtt{t}, \alpha] \wedge \sim st(p_j, t) : [\mathtt{t}, \alpha]$$
$$\rightarrow R(p_i, p_j, t) : [(0, 8), \alpha], \tag{1}$$
$$R(p_i, p_j, t) : [(0, 0), \alpha] \wedge st(p_i, t) : [\mathtt{t}, \alpha] \wedge st(p_j, t) : [\mathtt{t}, \alpha]$$
$$\rightarrow R(p_i, p_j, t) : [(5, 5), \alpha]. \tag{2}$$

Suppose that one of basic bf-inference rules $(0,0)$-rule-1 and 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 have two groups of basic bf-inference rules to be applied immediately after basic bf-inference rules $(0,0)$-rule-1 and 2, which are called $(0,8)$-*rules* and $(5,5)$-*rules* respectively. **(0,8)-rules**
Suppose that process Pr_i has started before process Pr_j starts, then the vector annotation of bf-literal $R(p_i, p_j, t)$ should be $(0,8)$. Then we have the following inference rules to be applied immediately after basic bf-inference rule $(0,0)$-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 bf-annotation $\mathtt{mb}(1,11)$.
$(0,8)$-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 bf-annotation $\mathtt{db}(0,12)$.
$(0,8)$-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 bf-annotations, $\mathtt{jb}(2,10)$, $\mathtt{sb}(3,9)$, $\mathtt{ib}(4,8)$. Basic bf-inference rules $(0,8)$-rule-1,2 and 3 may be translated into the bf-EVALPSN clauses,

$$R(p_i,p_j,t):[(0,8),\alpha] \wedge fi(p_i,t):[\mathtt{t},\alpha] \wedge st(p_j,t):[\mathtt{t}] \\ \rightarrow R(p_i,p_j,t):[(1,11),\alpha], \tag{3}$$

$$R(p_i,p_j,t):[(0,8),\alpha] \wedge fi(p_i,t):[\mathtt{t},\alpha] \wedge \sim st(p_j,t):[\mathtt{t}] \\ \rightarrow R(p_i,p_j,t):[(0,12),\alpha], \tag{4}$$

$$R(p_i,p_j,t):[(0,8),\alpha] \wedge \sim fi(p_i,t):[\mathtt{t},\alpha] \wedge st(p_j,t):[\mathtt{t},\alpha] \\ \rightarrow R(p_i,p_j,t):[(2,8),\alpha]. \tag{5}$$

(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 $(5,5)$. Then we have the following inference rules to be applied immediately after basic bf-inference rule $(0,0)$-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 bf-annotation $\mathtt{sb}(5,7)$.
$(5,5)$-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 bf-annotation $\mathtt{pba}(6,6)$.
$(5,5)$-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 bf-annotation $\mathtt{sa}(7,5)$. Basic bf-inference rules $(5,5)$-rules-1,2 and 3 may be translated into the bf-EVALPSN clauses,

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

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

$$R(p_i,p_j,t):[(5,5),\alpha] \wedge \sim fi(p_i,t):[\mathtt{t},\alpha] \wedge fi(p_j,t):[\mathtt{t},\alpha] \\ \rightarrow R(p_i,p_j,t):[(7,5),\alpha]. \tag{8}$$

If one of basic bf-inference rules, $(5,5)$-rule-1,2 and 3, and $(0,8)$-rule-1 and 2 has been applied, the final bf-relations represented by bf-annotations such as $\mathtt{jb}(2,10)/\mathtt{ja}(10,2)$ between two processes should be derived. On the other hand, even if basic bf-inference rule $(0,8)$-rule-3 has been applied, no bf-annotation could be derived. Therefore, a group of basic bf-inference rules called $(2,8)$-*rules* should be considered after applying basic bf-inference rule $(0,8)$-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 bf-annotation $\mathtt{jb}(2,10)$.

$(2,8)$-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 bf-annotation $\mathtt{fb}(3,9)$.

$(2,8)$-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 bf-annotation $\mathtt{ib}(4,8)$.

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

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

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

$$R(p_i,p_j,t):[(2,8),\alpha] \wedge \sim fi(p_i,t):[\mathtt{t},\alpha] \wedge fi(p_j,t):[\mathtt{t},\alpha] \\ \rightarrow R(p_i,p_j,t):[(4,8),\alpha]. \tag{11}$$

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

TABLE I: Application Orders of Basic Bf-inference Rules

vector	rule	vector	rule	vector	rule	vector
$(0,0)$	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	$(7,5)$		

C. Transitive before-after inference rule

Here we review another kind of bf-inference rules called *transitive bf-inference rules*, which can reason one vector annotation of bf-literal from two other vector annotations of bf-literals transitively. Bf-relations represented by bf-annotations could be reasoned efficiently by using the transitive bf-inference rules. Suppose that there are three processes Pr_i,Pr_j and Pr_k starting sequentially, then we consider to reason the vector annotation of bf-literal $R(p_i,p_k,t)$ from those

of bf-literals $R(p_i, p_j, t)$ and $R(p_j, p_k, t)$ transitively. We will show simple examples for forming some transitive bf-inference rules as introduction.

Example 1

Suppose that both processes Pr_i and Pr_j have already started at time t but process Pr_k has not started yet as shown in Fig.12, then we have obtained the vector annotation $(2, 8)$ of bf-literal $R(p_i, p_j, t)$ by basic bf-inference rule $(0, 8)$-rule-3 and the vector annotation $(0, 8)$ of bf-literal $R(p_j, p_k, t)$ by basic bf-inference rule $(0, 0)$-rule-1. Then, obviously the vector annotation of bf-literal $R(p_i, p_k, t)$ is reasoned as bf-annotation $\mathsf{be}(0, 8)$. Thus, we have the following bf-EVALP clause as a transitive bf-inference rule,

$$R(p_i, p_j, t) : [(2, 8), \mu] \wedge R(p_j, p_k, t) : [(0, 8), \mu] \\ \rightarrow R(p_i, p_k, t) : [(0, 8), \mu], \qquad \mu \in \{\alpha, \beta, \gamma\}.$$

Here we list all transitive bf-inference rules. The details of how to construct transitive bf-inference

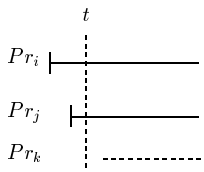

Fig. 12: Process Time Chart Ex-3

rules are in [18] For simplicity, we represent a transitive bf-inference rule,

$$R(p_i, p_j, t) : [(n_1, n_2), \alpha] \wedge R(p_j, p_k, t) : [(n_3, n_4), \alpha] \\ \rightarrow R(p_i, p_k, t) : [(n_5, n_6), \alpha]$$

by only vector annotations and logical connectives, $\wedge$ and $\rightarrow$, as follows: $(n_1, n_2) \wedge (n_3, n_4) \rightarrow (n_5, n_6)$ in the list of transitive bf-inference rules.

Transitive Bf-inference Rules

$$\begin{aligned}
&\mathbf{TR0} \quad (0,0) \wedge (0,0) \rightarrow (0,0) \\
&\mathbf{TR1} \quad (0,8) \wedge (0,0) \rightarrow (0,8) \\
&\mathbf{TR1-1} \quad (0,12) \wedge (0,0) \rightarrow (0,12) \\
&\mathbf{TR1-2} \quad (1,11) \wedge (0,8) \rightarrow (0,12) \\
&\mathbf{TR1-3} \quad (1,11) \wedge (5,5) \rightarrow (1,11) \\
&\mathbf{TR1-4} \quad (2,8) \wedge (0,8) \rightarrow (0,8)
\end{aligned}$$

$$\begin{aligned}
&\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) \qquad (12)\\
&\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) \qquad (13)\\
&\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) \qquad (14)\\
&\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) \qquad (15)\\
&\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) \qquad (16)\\
&\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) \qquad (17)\\
&\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) \qquad (18)\\
&\mathbf{TR3} \quad (5,5)\wedge(5,5)\rightarrow(5,5)\\
&\mathbf{TR3-1} \quad (7,5)\wedge(5,7)\rightarrow(5,5) \qquad (19)\\
&\mathbf{TR3-2} \quad (5,7)\wedge(7,5)\rightarrow(6,6)
\end{aligned}$$

Note : the bottom vector annotation $(0,0)$ in the list of transitive bf-inference rules implies that any bf-EVALP clause $R(p_j,p_k,t):[(n,m),\alpha]$ satisfies it.

Here we indicate two important points in terms of transitive bf-inference rules.

(I) Names of transitive bf-inference rules such as TR1-4-3 show their applicable orders. For example, if transitive bf-inference rule TR1 has been applied, one of transitive bf-inference rules

TR1-1,TR1-2,... or TR1-5 should be applied at the following stage; if transitive bf-inference rule TR1-4 has been applied after transitive bf-inference rule TR1, one of transitive bf-inference rules TR1-4-1,TR1-4-2, ... or TR1-4-7 should be applied at the following stage; on the other hand, if one of transitive bf-inference rules TR1-1, TR1-2 or TR1-3 has been applied after transitive bf-inference rule TR1, there is no transitive bf-inference rule to be applied at the following stage because bf-annotations $\mathtt{db}(0,12)$ or $\mathtt{mb}(1,11)$ between processes Pr_i and Pr_k have already been derived.

(II) The eight transitive bf-inference rules, TR1-4-2 (12), TR2-2 (16), TR1-4-3-2 (13), TR2-3-2 (17), TR1-4-6 (14), TR2-5 (18), TR1-5-1 (15), TR3-1 (19) have no following rule to be applied at the following stage, even though they cannot derive the final bf-relations between processes represented by bf-annotations such as $\mathtt{jb}(2,10)/\mathtt{ja}(10,2)$. For example, suppose that transitive bf-inference rule TR1-4-3-2 has been applied, then the vector annotation $(2,8)$ of the bf-literal (p_i,p_k,t) just implies that the final bf-relation between processes Pr_i and Pr_k is one of three bf-annotations, $\mathtt{jb}(2,10)$, $\mathtt{sb}(3,9)$ and $\mathtt{ib}(4,8)$. Therefore, if one of the eight transitive bf-inference rules has been applied, one of basic bf-inference rules $(0,8)$-rule, $(2,8)$-rule or $(5,5)$-rule should be applied for deriving the final bf-annotation at the following stage. For instance, if transitive bf-inference rule TR1-4-3-2 has been applied, basic bf-inference rule $(2,8)$-rule should be applied at the following stage.

D. Example of transitive bf-relation reasoning

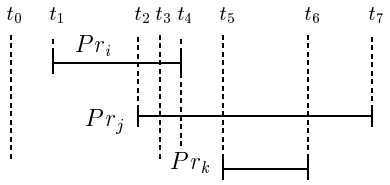

Fig. 13: Process Time Chart Ex-4-3

Now we show an example of bf-relation reasoning by transitive bf-inference rules taking the process time chart in Fig.13.

At time t_1, transitive bf-inference rule TR1 is applied and we have the bf-EVALP clause, $R(p_i,p_k,t_1):[(0,8),\alpha]$.

At time t_2, transitive bf-inference rule TR1-2 is applied, however bf-literal $R(p_i,p_k,t_2)$ has the same vector annotation $(0,8)$ as the previous time t_1. Therefore, we have the bf-EVALP clause, $R(p_i,p_k,t_2):[(0,8),\alpha]$.

At time t_3, no transitive bf-inference rule can be applied, since the vector annotations of bf-literals $R(p_i,p_j,t_3)$ and $R(p_j,p_k,t_3)$ are the same as the previous time t_2. Therefore, we still have the bf-EVALP clause having the same vector annotation, $R(p_i,p_k,t_3):[(0,8),\alpha]$.

At time t_4, transitive bf-inference rule TR1-2-1 is applied and we obtain the bf-EVALP clause having bf-annotation $\mathtt{db}(0,12)$, $R(p_i,p_k,t_4):[(0,12),\alpha]$.

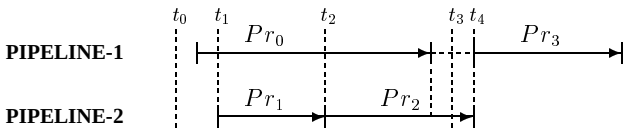

Fig. 14: Pipeline Process Schedule

V. Application of bf-EVALPSN to process order verification

In this section, we present a simple example for applying the bf-relation reasoning system in bf-EVALPSN to process order verification.

Suppose that there is a pipeline system consists of two pipelines PIPELINE-1 and PIPELINE-2, which is also supposed to deal with four pipeline processes Pr_0, Pr_1, Pr_2 and Pr_3 according to the process schedule in Fig.14. Moreover, the process order verification system has four safety properties $SPR-i$ for processes $Pr_i(i = 0, 1, 2, 3)$ to be strictly assured, respectively.

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.

All the safety properties can be translated into the following bf-EVALPSN clauses:

[SPR-0]

$$\sim R(p_0, p_1, t) : [(0, 8), \alpha] \rightarrow st(p_1, t) : [\mathtt{f}, \beta], \tag{20}$$

$$\sim R(p_0, p_2, t) : [(0, 8), \alpha] \rightarrow st(p_2, t) : [\mathtt{f}, \beta], \tag{21}$$

$$\sim R(p_0, p_3, t) : [(0, 8), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \tag{22}$$

$$st(p_1, t) : [\mathtt{f}, \beta] \wedge st(p_2, t) : [\mathtt{f}, \beta] \wedge st(p_3, t) : [\mathtt{f}, \beta]$$
$$\rightarrow st(p_0, t) : [\mathtt{f}, \gamma], \tag{23}$$

$$\sim fi(p_0, t) : [\mathtt{f}, \beta] \rightarrow fi(p_0, t) : [\mathtt{f}, \gamma], \tag{24}$$

where the bf-EVALPSN clauses (20),(21) and (22) declare that if process Pr_0 has not started before each process $Pr_i(i = 1, 2, 3)$ starts, it should be forbidden to start each process $Pr_i(i = 1, 2, 3)$, respectively; the bf-EVALPSN clause (23) declares that if each process $Pr_i(i = 1, 2, 3)$ is forbidden from starting, it should be permitted to start process Pr_0; and the bf-EVALPSN clause (24) declares that if there is no forbiddance from finishing process Pr_0, it should be permitted to finish process Pr_0.

[SPR-1]

$$\sim st(p_1, t) : [\mathtt{f}, \beta] \rightarrow st(p_1, t) : [\mathtt{f}, \gamma], \tag{25}$$

$$\sim fi(p_1, t) : [\mathtt{f}, \beta] \rightarrow fi(p_1, t) : [\mathtt{f}, \gamma], \tag{26}$$

where the bf-EVALPSN clauses (25)/(26) declare that if there is no forbiddance from starting/finishing

process Pr_1 respectively, it should be permitted to start/finish process Pr_1.

[SPR-2]

$$\sim R(p_2, p_1, t) : [(11, 0), \alpha] \rightarrow st(p_2, t) : [\mathtt{f}, \beta], \tag{27}$$

$$\sim st(p_2, t) : [\mathtt{f}, \beta] \rightarrow st(p_2, t) : [\mathtt{f}, \gamma], \tag{28}$$

$$\sim R(p_2, p_0, t) : [(10, 2), \alpha] \rightarrow fi(p_2, t) : [\mathtt{f}, \beta], \tag{29}$$

$$\sim fi(p_2, t) : [\mathtt{f}, \beta] \rightarrow fi(p_2, t) : [\mathtt{f}, \gamma], \tag{30}$$

where the bf-EVALPSN clause (27) declares that if process Pr_1 has not finished before process Pr_2 starts, it should be forbidden to start process Pr_2; the vector annotation $(11, 0)$ of bf-literal $R(p_2, p_1, t)$ is the greatest lower bound of the set $\{\mathtt{da}(12, 0), \mathtt{ma}(11, 1)\}$, which implies that process Pr_1 has finished before process Pr_2 starts in either way; the bf-EVALPSN clauses (28)/(30) declare that if there is no forbiddance from starting/finishing process Pr_2, it should be permitted to start/finish process Pr_2, respectively; and the bf-EVALPSN clauses (29) declares that if process Pr_0 has not finished before process Pr_2 finishes, it should be forbidden to finish process Pr_2.

[SPR-3]

$$\sim R(p_3, p_0, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \tag{31}$$

$$\sim R(p_3, p_1, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \tag{32}$$

$$\sim R(p_3, p_2, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \tag{33}$$

$$\sim st(p_3, t) : [\mathtt{f}, \beta] \rightarrow st(p_3, t) : [\mathtt{f}, \gamma], \tag{34}$$

$$\sim fi(p_3, t) : [\mathtt{f}, \beta] \rightarrow fi(p_3, t) : [\mathtt{f}, \gamma], \tag{35}$$

where the bf-EVALPSN clauses (31),(32) and (33) declare that if none of processes $Pr_i (i = 0, 1, 2)$ has not finished, it should be forbidden to start process Pr_3; and the bf-EVALPSN clauses (34)/(35) declares that if there is no forbiddance from starting/finishing process Pr_3, it should be permitted to start/finish process Pr_3, respectively. Now, we show how the process order verification in bf-EVALPSN is carried out in real-time by considering the process schedule in Fig.14. In the following example, five bf-relations between processes Pr_0, Pr_1, Pr_2 and Pr_3 represented by the vector annotations of bf-literals, $R(p_0, p_1, t)$, $R(p_0, p_2, t)$, $R(p_0, p_3, t)$, $R(p_2, p_1, t)$ and $R(p_3, p_2, t)$ are verified in real-time based on safety properties $SPR-0$, $SPR-1$, $SPR-2$ and $SPR-3$.

Stage 0(at time t_0) no process has started at time t_0, thus, the bf-EVALP clauses,

$$R(p_0, p_1, t_0) : [(0, 0), \alpha], \tag{36}$$

$$R(p_1, p_2, t_0) : [(0, 0), \alpha], \tag{37}$$

$$R(p_2, p_3, t_0) : [(0, 0), \alpha] \tag{38}$$

are obtained; also, the bf-EVALP clauses,

$$R(p_0, p_2, t_0) : [(0, 0), \alpha], \tag{39}$$

$$R(p_0, p_3, t_0) : [(0, 0), \alpha] \tag{40}$$

are obtained by transitive bf-inference rule TR0; then, the bf-EVALP clauses (36) and (39) satisfy each body of the bf-EVALPSN clauses (20),(21) and (22) respectively, therefore, the forbiddance from starting each process $Pr_i (i = 1, 2, 3)$,

$$st(p_1, t_0) : [\mathtt{f}, \beta], \quad st(p_2, t_0) : [\mathtt{f}, \beta], \text{ and}$$
$$st(p_3, t_0) : [\mathtt{f}, \beta] \tag{41}$$

are derived; moreover, as the bf-EVALP clauses (41) satisfy the body of the bf-EVALPSN clause (23), the permission for starting process Pr_0, $st(p_0, t_0) : [\mathtt{f}, \gamma]$ is derived; therefore, process Pr_0 is permitted to start at this stage.

Stage 1(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,

$$R(p_0, p_1, t_1) : [(0, 8), \alpha], \tag{42}$$
$$R(p_1, p_2, t_1) : [(0, 0), \alpha], \tag{43}$$
$$R(p_2, p_3, t_1) : [(0, 0), \alpha] \tag{44}$$

are obtained, where the bf-EVALP clause (42) is derived by basic bf-inference rule $(0, 0)$-rule-1; moreover, the bf-EVALP clauses,

$$R(p_0, p_2, t_1) : [(0, 8), \alpha], \tag{45}$$
$$R(p_0, p_3, t_1) : [(0, 8), \alpha] \tag{46}$$

are obtained by transitive bf-inference rule TR1; as the bf-EVALP clause (42) does not satisfy the body of the bf-EVALPSN clause (20), the forbiddance from starting process Pr_1,

$$st(p_1, t_1) : [\mathtt{f}, \beta] \tag{47}$$

cannot be derived; Then, as there is not the forbiddance (47), the body of the bf-EVALPSN clause (25) is satisfied, and the permission for starting process Pr_1, $st(p_1, t_1) : [\mathtt{f}, \gamma]$ is derived; on the other hand, as the bf-EVALP clauses (45) and (46) satisfy the body of the bf-EVALPSN clauses (27) and (31) respectively, the forbiddance from starting both processes Pr_2 and Pr_3, $st(p_2, t_1) : [\mathtt{f}, \beta]$, and $st(p_3, t_1) : [\mathtt{f}, \beta]$ are derived; therefore, process Pr_1 is permitted to start.

Stage 2(at time t_2) process Pr_1 has just finished and process Pr_0 has not finished yet; then, the bf-EVALP clauses,

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

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,

$$R(p_0, p_2, t_2) : [(2, 8), \alpha], \tag{48}$$
$$R(p_0, p_3, t_2) : [(0, 12), \alpha] \tag{49}$$

are obtained by transitive bf-inference rules TR1-4-6 and TR1-2, respectively; then, as the bf-EVALP clause (48) does not satisfy the body of the bf-EVALPSN clause (27), the forbiddance from starting process Pr_2,

$$st(p_2, t_2) : [\mathtt{f}, \beta] \tag{50}$$

cannot be derived; as there is not the forbiddance (50), it satisfies the body of the bf-EVALPSN clause (28), and the permission for starting process Pr_2, $st(p_2, t_2) : [\mathtt{f}, \gamma]$ is derived; on the other hand, as the bf-EVALP clause (49) satisfies the body of the bf-EVALPSN clause (31), the forbiddance from starting process Pr_3, $st(p_3, t_2) : [\mathtt{f}, \beta]$ is derived; therefore, process Pr_2 is permitted to start, however process Pr_3 is still forbidden from starting.

Stage 3(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,

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

which have the same vector annotations as the previous stage are obtained; moreover, the bf-EVALP clauses,

$$R(p_0, p_2, t_3) : [(2, 10), \alpha], \tag{51}$$
$$R(p_0, p_3, t_3) : [(0, 12), \alpha] \tag{52}$$

are obtained, where the bf-EVALP clause (51) is derived by the basic bf-inference rule $(2, 8)$-rule-1; then, the bf-EVALP clause (51) satisfies the body of the bf-EVALP clause (33), 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 from starting because process Pr_2 has not finished yet at this stage.
Stage 4(at time t_4) process Pr_2 has just finished and process Pr_3 has not started yet; then, the bf-EVALP clauses,

$$R(p_0, p_1, t_4) : [(4, 8), \alpha], \tag{53}$$
$$R(p_1, p_2, t_4) : [(1, 11), \alpha], \tag{54}$$
$$R(p_2, p_3, t_4) : [(1, 11), \alpha], \tag{55}$$
$$R(p_0, p_2, t_4) : [(2, 10), \alpha], \tag{56}$$
$$R(p_0, p_3, t_4) : [(0, 12), \alpha] \tag{57}$$

are obtained; the bf-EVALP clause (55) is derived by basic bf-inference rule $(0, 8)$-rule-2; moreover, as the bf-EVALP clauses (53),(56) and (57) do not satisfy the bodies of the bf-EVALP clauses (31),(32) and (33), the forbiddance from starting process Pr_3,

$$st(p_3, t_4) : [\mathtt{f}, \beta] \tag{58}$$

cannot be derived; as there is not the forbiddance (58), the body of the bf-EVALPSN clause (34)is satisfied, and the permission for starting process Pr_3, $st(p_3, t_4) : [\mathtt{f}, \gamma]$ is derived; Therefore, process Pr_3 is permitted to start because processes Pr_0, Pr_1 and Pr_2 have finished.

VI. CONCLUSION

In this paper, we have introduced a logical reasoning system for before-after relations between processes(time intervals) based on a paraconsistent annotated logic program bf-EVALPSN, which consists of two groups of inference rules in bf-EVALPSN called basic and transitive bf-inference rules. As related work, an interval temporal logic has been proposed for developing practical planning and natural language understanding systems in Allen [1], [2]. In his logic, before-after relations between two time intervals are represented in special predicates and treated in a framework of first order temporal logic. On the other hands, in our bf-EVALPSN before-after reasoning system, before-after relations between processes are regarded as paraconsistency and represented more minutely in vector annotations of the special literal $R(p_i, p_j, t)$ called bf-literal, and treated in the framework of annotated logic programming. Moreover, an efficient real-time before-after relation

reasoning mechanism called transitive bf-inference is implemented in our system. Therefore, we would like to conclude that our bf-EVALPSN before-after relation reasoning system is more suitable for dealing with process order control/verification in real-time, with considering its hardware implementation such as on microchips.

Our system has a lot of applications though, our future work focuses on its application to logical design for various process order control systems based on the safety verification.

REFERENCES

[1] J.F. Allen, "Towards a General Theory of Action and Time", *Artificial Intelligence* vol. 23, pp. 123–154, 1984.

[2] J.F. Allen, and G. Ferguson, "Actions and Events in Interval Temporal Logic", *J.Logic and Computation* vol. 4, pp. 531–579, 1994.

[3] H.A. Blair, and V.S. Subrahmanian, "Paraconsistent Logic Programming", *Theoretical Computer Science* vol. 68, pp. 135–154, 1989.

[4] N.C.A. da Costa, V.S. Subrahmanian, and C. Vago, "The Paraconsistent Logics P$\mathcal{T}$", *Zeitschrift für Mathematische Logic und Grundlangen der Mathematik* vol. 37, pp. 139–148, 1989.

[5] K. Nakamatsu, and A. Suzuki, "Annotated semantics for default reasoning", *Proc. 3rd Pacific Rim Intl. Conf. Artificial Intelligence (PRICAI94)*, pp. 180–186, 1994.

[6] K. Nakamatsu, "On the relation between vector annotated logic programs and defeasible theories", *Logic and Logical Philosophy* vol. 8, pp. 181–205, 2000.

[7] K. Nakamatsu, J.M. Abe, and A. Suzuki, "Annotated Semantics for Defeasible Deontic Reasoning", *Rough Sets and Current Trends in Computing*, LNAI vol. 2005, pp.432–440, 2001.

[8] K. Nakamatsu, H. Suito, J.M. Abe, and A. Suzuki, "Paraconsistent logic program based safety verification for air traffic control", *Proc. IEEE Intl. Conf. System, Man and Cybernetics 02(SMC02)* (CD), 2002.

[9] K. Nakamatsu, J.M. Abe, and A. Suzuki, "A railway interlocking safety verification system based on abductive paraconsistent logic programming", *Soft Computing Systems(HIS02)* Frontiers in Artificial Intelligence and Applications, vol. 87, pp. 775–784, 2002.

[10] K. Nakamatsu, T. Seno, J.M. Abe, and A. Suzuki, "Intelligent real-time traffic signal control based on a paraconsistent logic program EVALPSN", *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing(RSFDGrC2003)* LNAI vol. 2639, pp. 719–723, 2003.

[11] K. Nakamatsu, S-L. Chung, H. Komaba, and A. Suzuki, "A discrete event control based on EVALPSN stable model", *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing(RSFDGrC2005)*, LNAI vol. 3641, pp. 671–681, 2005.

[12] K. Nakamatsu, J.M. Abe, S. Akama, "An intelligent safety verification based on a paraconsistent logic program", *Proc. 9th Intl. Conf. Knowledge-Based Intelligent Information and Engineering Systems(KES2005)*, LNAI vol. 3682, pp. 708–715, 2005.

[13] K. Nakamatsu, K. Kawasumi, and A. Suzuki, "Intelligent verification for pipeline based on EVALPSN", *Advances in Logic Based Intelligent Systems*, Frontiers in Artificial Intelligence and Applications, vol. 132, pp. 63–70, 2005.

[14] K. Nakamatsu, "Pipeline Valve Control Based on EVALPSN Safety Verification", J.Advanced Computational Intelligence and Intelligent Informatics, vol. 10, pp. 647-656, 2006.

[15] K. Nakamatsu, Y. Mita, and T. Shibata, "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.

[16] K. Nakamatsu, and J.M. Abe, "The development of Paraconsistent Annotated Logic Program", *Int'l J. Reasoning-based Intelligent Systems*, vol. 1, pp. 92-112, 2009.

[17] K. Nakamatsu, T. Imai, J.M. Abe, and S. Akama, Introduction to Plausible Reasoning Based on EVALPSN, *Advanced in Intelligent Decision Technologies*, SCI vol. 199, pp. 363–372, Springer, 2009.

[18] K. Nakamatsu, J.M. Abe, and S. Akama, A Logical Reasoning System of Process Before-after Relation Based on a Paraconsistent Annotated Logic Program bf-EVALPSN, *Intl J. Knowledge-based and Intelligent Engineering Systems*, 15 (2011), pp. 145–163.