Intelligent Safety Verification for Pipeline Process Order Control Based on bf-EVALPSN

Kazumi Nakamatsu
School of Human Science and Environment
University of Hyogo
Himeji, Japan
Email: nakamatu@shse.u-hyogo.ac.jp

Jair Minoro Abe
GP in Production Eng., ICET
Paulista University
Sao Paulo, Brazil
Email: jairabe@uol.com.br

Seiki Akama
C-corporation
Kawasaki, Japan
Email:akama@jcom.home.ne.jp

***Abstract*—A paraconsistent logic program called Extended Vector Annotated Logic Program with Strong Negation(abbr. EVALPSN) has been developed for dealing with defeasible deontic reasoning and plausible reasoning, and also applied to various kinds of intelligent safety verification and control. Moreover, in order to deal with before-after relation between processes (time intervals), another EVALPSN called bf(before-after)-EVALPSN has been developed recently. In this paper, we review the reasoning system for before-after relation between processes based on bf-EVALPSN and introduce how to apply the reasoning system to real-time pipeline process order safety verification and control with an example.**

Keywords- before-after relation; paraconsistent logic program; safety verification; pipeline process order; 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), VALPSN (Vector ALPSN), EVALPSN (Extended VALPSN) that can deal with defeasible deontic and plausible reasonings, and bf (before-after)-EVALPSN that can deal with before-after relation between processes (time intervals) [9]. We note that "before-after" is abbreviated as just "bf" hereafter. Those annotated logic programs have been applied to various kinds of intelligent control and safety verification such as pipeline valve control based on safety verification [7] and real-time process order control based on safety verification [10], 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 [8].

In this paper, we review the reasoning system for process before-after relation in bf-EVALPSN [10] and show how to apply it to the safety verification for process order with an example. The before-after relation reasoning system based on bf-EVALPSN consists of two groups of inference rules called *basic bf-inference rules* and *transitive bf-inference rules*, 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 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, respectively, and before-after relations between two processes are represented in vector annotations.

In the bf-EVALPSN 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 reduce the frequency of basic bf-inference rules and it is a unique feature of our system.

This paper is organized as follows: 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.

II. EVALPSN

In this section, we review EVALPSN briefly [5]. 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

 ISBN: 978-1-61208-184-7

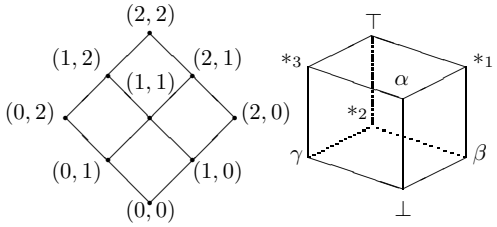

Figure 1. Lattice $\mathcal{T}_v(2)$ and Lattice $\mathcal{T}_d$

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 Fig. 1. 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,

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

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: let $[(i_1, j_1), \mu_1]$, $[(i_2, j_2), \mu_2] \in \mathcal{T}_e$,

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

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. The details are found in [10] 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 paraconsistently in vector annotations.

 ISBN: 978-1-61208-184-7

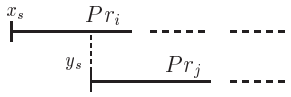

Figure 2. Bf-relations Before(be)/After(af)

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

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 a *bf-annotation*. Exactly speaking, bf-relation has fifteen meaningful kinds according to bf-relations between each start/finish time of two processes in bf-EVALPSN. Let us start from the most basic bf-relations in bf-EVALPSN.

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 in Fig. 2.

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)
are defined as there is a time lag between the earlier process finish time and the later one start time, which are described in Fig. 3.

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 in Fig. 4.

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 in Fig. 5.

S-included Before (sb), S-included After (sa)

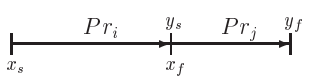

Figure 4. Bf-relations Immediate Before(mb)/After(ma)

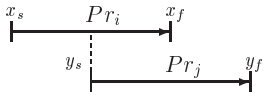

Figure 5. Bf-relations, Joint Before/After

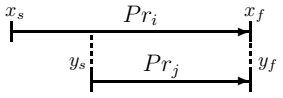

Figure 6. Bf-relations S-included Before(sb)/After(sa)

are defined as one process had started before another one started and they have finished at the same time, which are described in Fig. 6.

Included Before (ib)/After (ia)
are defined as one process had started/finished before/after another one started/finished, which are described in Fig. 7.

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 in Fig. 8.

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 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 is obviously defined as the following mappings :

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

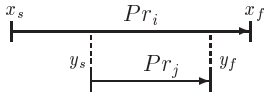

Figure 7. Bf-relations Included Before (ib)/After (ia)

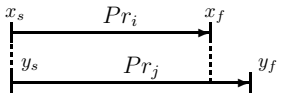

Figure 8. Bf-relations F-included Before(fb)/After(fa)

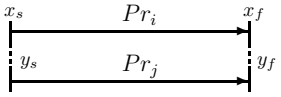

Figure 9. Bf-relation, Paraconsistent Before-after

Therefore, each bf-annotation can be translated into vector annotatations as $\mathtt{bf}=(0,8)$, $\mathtt{db}=(0,12)$, $\mathtt{mb}=(1,11)$, $\mathtt{jb}=(2,10)$, $\mathtt{sb}=(3,9)$, $\mathtt{ib}=(4,8)$, $\mathtt{fb}=(5,7)$, $\mathtt{pba}=(6,6)$.

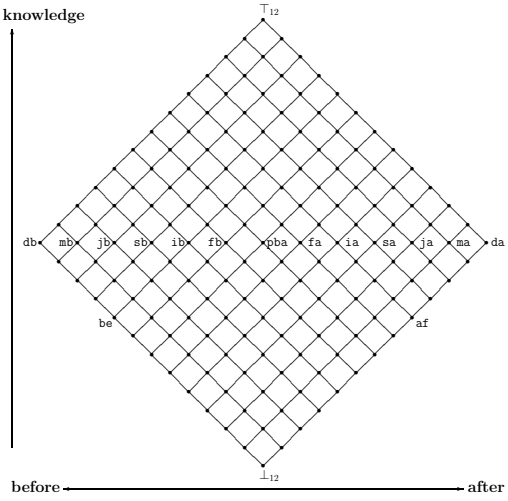

Figure 10. The Complete Lattice $\mathcal{T}_v(12)_{bf}$ of Bf-annotations

IV. Reasoning System in Bf-EVALPSN

In this section, we introduce the bf-relation reasoning system in bf-EVALPSN, which consists of two kinds of inference rules called basic bf-inference rules and transitive bf-inference rules.

A. Basic Before-after Inference Rule

In order to represent basic bf-inference rules in bf-EVALPSN, we newly introduce two 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 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.

Here we introduce the first group of basic bf-inference rules to be applied at the initial stage, 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.

$(0,0)$-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 $\mathtt{be}(0,8)$.

$(0,0)$-rule-2 If both processes Pr_i and Pr_j have started at the same time, then 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 can be translated into the bf-EVALPSN clauses,

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

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

Next, 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 $(0,8)$ or $(5,5)$. Therefore, we have the following two groups of basic bf-inference rules to be applied after basic bf-inference rules $(0,0)$-rule, which are called $(0,8)$-*rules* and $(5,5)$-*rules*.

(0,8)-rules

Suppose that the vector annotation of bf-literal $R(p_i,p_j,t)$ is $\mathtt{be}(0,8)$. Then we have the following bf-inference rules to be applied 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 $\mathtt{mb}(1,11)$.

$(0,8)$-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 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 can be translated into the bf-EVALPSN clauses,

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

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

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

(5,5)-rules

Suppose that the vector annotation of bf-literal $R(p_i,p_j,t)$ is $(5,5)$. Then we have the following bf-inference rules to be applied 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 $\mathtt{sb}(5,7)$.

 ISBN: 978-1-61208-184-7

$(5,5)$-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 $(5,5)$-rules-1,2 and 3 can be translated into the bf-EVALPSN clauses,

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

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

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

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

(2,8)-rules

Suppose that the vector annotation of bf-literal $R(p_i,p_j,t)$ is $(2,8)$. Then we have the following bf-inference rules to be applied after basic bf-inference rule $(0,8)$-rule-3.

$(2,8)$-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 can be translated into the bf-EVALPSN clauses,

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

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

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

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

B. Transitive Before-after Inference Rule

Here we introduce another kind of bf-inference rules, transitive bf-inference rule.

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	$(7,5)$		

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. First of all, we will show a simple example for forming 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. 11, 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 $\mathtt{be}(0,8)$. Thus, we may obtain the following bf-EVALP clause as a transitive bf-inference rule,

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

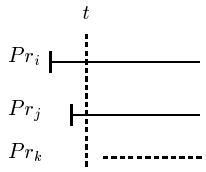

Figure 11. Process Time Chart Ex-3

Here we list all transitive bf-inference rules. The details of how to construct transitive bf-inference rules are in [10] 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

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

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

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

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

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

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

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

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

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

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

$\mathbf{TR1-4-3-2}$ $(4,8) \wedge (2,10) \rightarrow (2,8)$ (13)

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

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

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

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

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

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

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

$\mathbf{TR1-4-6}$ $(4,8) \wedge (1,11) \rightarrow (2,8)$ (14)

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

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

$\mathbf{TR1-5-1}$ $(4,8) \wedge (5,7) \rightarrow (2,8)$ (15)

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

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

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

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

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

$\mathbf{TR2-2}$ $(7,5) \wedge (0,12) \rightarrow (0,8)$ (16)

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

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

$\mathbf{TR2-3-2}$ $(7,5) \wedge (2,10) \rightarrow (2,8)$ (17)

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

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

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

$\mathbf{TR2-5}$ $(7,5) \wedge (1,11) \rightarrow (2,8)$ (18)

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

$\mathbf{TR3-1}$ $(7,5) \wedge (5,7) \rightarrow (5,5)$ (19)

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

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) The number chain 1-4-3 of transitive bf-inference rule TR1-4-3 show the rule applicable order, that is to say, rule TR1-4-3 should be applied after rule TR1-4 and rule TR1-4 should be applied after TR1, if they are applicable.

(II) 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. For example, suppose that rule TR1-4-3-2 has been applied, then the vector annotation $(2,8)$ of the bf-literal $R(p_i, p_k, t)$ 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. For instance, if rule TR1-4-3-2 has been applied, basic bf-inference rule $(2,8)$-rule should be applied at the following stage.

V. Application of Bf-EVALPSN to Pipeline 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.

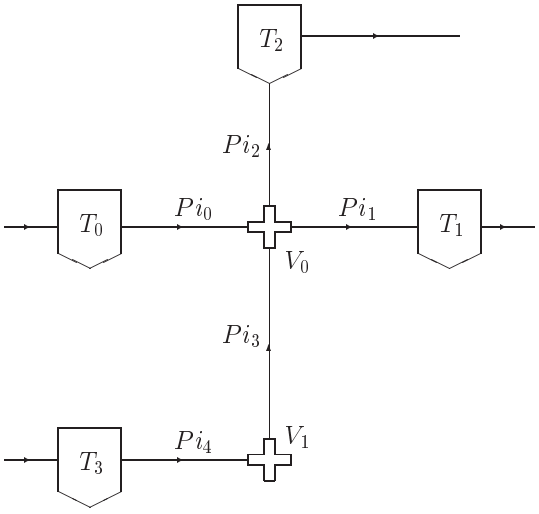

Figure 12. Pipeline Network

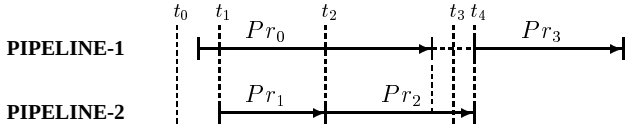

Figure 13. Pipeline Process Schedule

Example 2

We consider a simple brewery pipeline network in Fig. 12, which consists of four tanks, $\{T_0, T_1, T_2, T_3\}$; five pipes,

$\{Pi_0,Pi_1,Pi_2,Pi_3,Pi_4\}$; two valves, $\{V_0, V_1\}$. Two kinds of pipeline cleaning processes by nitric acid and cold water, respectively, are supposed to be processed after brewery processes, and the following four processes in the brewery pipeline network are scheduled according to the time chart in Fig. 13:

- process Pr_0 (brewery process 1), beer is transfered in the pipeline,

$$T_0 \ \rightarrow \ Pi_0 \ \rightarrow \ V_0 \ \rightarrow \ Pi_1 \ \rightarrow \ T_1;$$

- process Pr_1 (cleaning process 1), the pipeline,

$$T_3 \ \rightarrow \ pi_4 \ \rightarrow \ V_1 \ \rightarrow \ Pi_3 \ \rightarrow \ V_0 \ \rightarrow \ Pi_2,$$

  is cleaned by nitric acid;
- process Pr_2 (cleaning process 2), the pipeline,

$$T_3 \ \rightarrow \ pi_4 \ \rightarrow \ V_1 \ \rightarrow \ Pi_3 \ \rightarrow \ V_0 \ \rightarrow \ Pi_2$$

  is cleaned by cold water;
- process Pr_3 (brewery process 2), beer is transfered in the pipelines,

$$\begin{array}{r} T_0 \ \rightarrow \ Pi_0 \ \rightarrow \ V_0 \ \rightarrow \ Pi_1 \ \rightarrow \ T_1, \\ T_3 \ \rightarrow \ Pi_4 \ \rightarrow \ V_1 \ \rightarrow \ Pi_3 \ \rightarrow \ V_0 \ \rightarrow \ Pi_2 \ \rightarrow \ T_2 \end{array}$$

  with mixing beer at valve V_0.

Moreover the pipeline system has four safety properties $SPR-i(i = 0, 1, 2, 3)$ to be strictly secured in terms of process order.

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.

Our safety verification system consists of the following three steps:

STEP 1 in order to verify the safety of process order, the safety properties should be translated into bf-EVALPSN and stored.

STEP 2 the before-after relations between processes at each time should also be translated into bf-EVALP clauses and added to the stored bf-EVALPSN in the previous step.

STEP 3 the safety of starting a process is verified as bf-EVALPSN logic programming at each time.

We introduce the above safety verification steps with the pipeline process order. First of all, the safety properties are translated. Safety property *SPR-0* can be translated into the bf-EVALPSN clauses,

$$\sim R(p_0, p_1, t) : [(0, 8), \alpha] \rightarrow st(p_1, t) : [\mathtt{f}, \beta], \quad (20)$$
$$\sim R(p_0, p_2, t) : [(0, 8), \alpha] \rightarrow st(p_2, t) : [\mathtt{f}, \beta], \quad (21)$$
$$\sim R(p_0, p_3, t) : [(0, 8), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \quad (22)$$
$$st(p_1, t) : [\mathtt{f}, \beta] \wedge st(p_2, t) : [\mathtt{f}, \beta] \wedge st(p_3, t) : [\mathtt{f}, \beta]$$
$$\rightarrow st(p_0, t) : [\mathtt{f}, \gamma], \quad (23)$$
$$\sim fi(p_0, t) : [\mathtt{f}, \beta] \rightarrow fi(p_0, t) : [\mathtt{f}, \gamma], \quad (24)$$

where bf-EVALPSN clauses (20),(21) and (22) declare that if process Pr_0 has not started before any other processes, it should be forbidden to start processes $Pr_i(i = 1, 2, 3)$; 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.

Safety property *SPR-1* can be translated into the EVALP -SN clauses,

$$\sim st(p_1, t) : [\mathtt{f}, \beta] \rightarrow st(p_1, t) : [\mathtt{f}, \gamma], \quad (25)$$
$$\sim fi(p_1, t) : [\mathtt{f}, \beta] \rightarrow fi(p_1, t) : [\mathtt{f}, \gamma], \quad (26)$$

where EVALPSN clauses (25) and (26) declare that if there is no forbiddance from starting and finishing process Pr_1, respectively, it should be permitted to start and finish process Pr_1.

Safety property *SPR-2* can be translated into the bf-EVALPSN clauses,

$$\sim R(p_2, p_1, t) : [(11, 0), \alpha] \rightarrow st(p_2, t) : [\mathtt{f}, \beta], \quad (27)$$
$$\sim st(p_2, t) : [\mathtt{f}, \beta] \rightarrow st(p_2, t) : [\mathtt{f}, \gamma], \quad (28)$$
$$\sim R(p_2, p_0, t) : [(10, 2), \alpha] \rightarrow fi(p_2, t) : [\mathtt{f}, \beta], \quad (29)$$
$$\sim fi(p_2, t) : [\mathtt{f}, \beta] \rightarrow fi(p_2, t) : [\mathtt{f}, \gamma], \quad (30)$$

where 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; EVALPSN clauses (28) and (30) declare that if there is no forbiddance from starting and finishing process Pr_2, it should be permitted to start and finish process Pr_2, respectively; and 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.

Safety property *SPR-3* can be translated into the bf-EVALPSN clauses,

$$\sim R(p_3, p_0, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \quad (31)$$
$$\sim R(p_3, p_1, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \quad (32)$$
$$\sim R(p_3, p_2, t) : [(11, 0), \alpha] \rightarrow st(p_3, t) : [\mathtt{f}, \beta], \quad (33)$$
$$\sim st(p_3, t) : [\mathtt{f}, \beta] \rightarrow st(p_3, t) : [\mathtt{f}, \gamma], \quad (34)$$
$$\sim fi(p_3, t) : [\mathtt{f}, \beta] \rightarrow fi(p_3, t) : [\mathtt{f}, \gamma], \quad (35)$$

 ISBN: 978-1-61208-184-7

where 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 bf-EVALPSN clauses (34) and (35) declare that if there is no forbiddance from starting and finishing process Pr_3, it should be permitted to start and finish process Pr_3, respectively.

Now we show how the process order verification in bf-EVALPSN is carried out at time t_0, $\cdots$, time t_4 according to the process schedule in Fig. 13. Five bf-relations between processes Pr_0, Pr_1, Pr_2 and Pr_3, which are represented by 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 based on safety properties *SPR-0*, *SPR-1*, *SPR-2* and *SPR-3* in bf-EVALPSN.
Stage 0 (at time t_0) no process has started at time t_0, thus, the bf-EVALP clauses,

$$R(p_0, p_1, t_0) : [(0, 0), \alpha], \quad (36)$$
$$R(p_1, p_2, t_0) : [(0, 0), \alpha], \quad (37)$$
$$R(p_2, p_3, t_0) : [(0, 0), \alpha] \quad (38)$$

are obtained; also the bf-EVALP clauses,

$$R(p_0, p_2, t_0) : [(0, 0), \alpha], \quad (39)$$
$$R(p_0, p_3, t_0) : [(0, 0), \alpha] \quad (40)$$

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

$$st(p_1, t_0) : [\mathtt{f}, \beta], \quad (41)$$
$$st(p_2, t_0) : [\mathtt{f}, \beta], \quad (42)$$
$$st(p_3, t_0) : [\mathtt{f}, \beta] \quad (43)$$

are derived; moreover as bf-EVALP clauses (41), (42) and (43) satisfy the body of bf-EVALP 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.
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,

$$R(p_0, p_1, t_1) : [(0, 8), \alpha], \quad (44)$$
$$R(p_1, p_2, t_1) : [(0, 0), \alpha], \quad (45)$$
$$R(p_2, p_3, t_1) : [(0, 0), \alpha] \quad (46)$$

are obtained, where bf-EVALP clause (44) is derived by $(0, 0)$-rule-1; moreover the bf-EVALP clauses,

$$R(p_0, p_2, t_1) : [(0, 8), \alpha], \quad (47)$$
$$R(p_0, p_3, t_1) : [(0, 8), \alpha] \quad (48)$$

are obtained by rule TR1; as bf-EVALP clause (44) does not satisfy the body of bf-EVALPSN clause (20), the forbiddance from starting process Pr_1,

$$st(p_1, t_1) : [\mathtt{f}, \beta] \quad (49)$$

cannot be derived; then, as there is not the forbiddance (49), the body of 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 bf-EVALP clauses (47) and (48) satisfy the body of bf-EVALPSN clauses (27) and (31), respectively, the forbiddance from starting both processes Pr_2 and Pr_3,

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

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,

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

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

$$R(p_0, p_2, t_2) : [(2, 8), \alpha], \quad (50)$$
$$R(p_0, p_3, t_2) : [(0, 12), \alpha] \quad (51)$$

are obtained by rules TR1-4-6 and TR1-2, respectively; then, as bf-EVALP clauses (50), (50) and (50) do not satisfy the body of bf-EVALPSN clause (27), the forbiddance from starting process Pr_2,

$$st(p_2, t_2) : [\mathtt{f}, \beta] \quad (52)$$

cannot be derived; as there is not the forbiddance (52), it satisfies the body of 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 bf-EVALP clause (51) satisfies the body of 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 time 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,

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

 ISBN: 978-1-61208-184-7

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

$$R(p_0, p_2, t_3) : [(2, 10), \alpha], \quad (53)$$
$$R(p_0, p_3, t_3) : [(0, 12), \alpha] \quad (54)$$

are obtained, where bf-EVALP clause (53) is derived by $(2, 8)$-rule-1; then bf-EVALP clause (53) satisfies the body of 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,

$$R(p_0, p_1, t_4) : [(4, 8), \alpha], \quad (55)$$
$$R(p_1, p_2, t_4) : [(1, 11), \alpha], \quad (56)$$
$$R(p_2, p_3, t_4) : [(1, 11), \alpha], \quad (57)$$
$$R(p_0, p_2, t_4) : [(2, 10), \alpha], \quad (58)$$
$$R(p_0, p_3, t_4) : [(0, 12), \alpha] \quad (59)$$

are obtained; the bf-EVALP clause (57) is derived by $(0, 8)$-rule-2; moreover, as bf-EVALP clauses (55), (58) and (59) do not satisfy the bodies of bf-EVALP clauses (31), (32) and (33), the forbiddance from starting process Pr_3,

$$st(p_3, t_4) : [\mathtt{f}, \beta] \quad (60)$$

cannot be derived; as there is not the forbiddance (60), the body of 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 safety verification and control 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] Allen, J.F., "Towards a General Theory of Action and Time", *Artificial Intelligence* vol. 23, pp. 123–154, 1984.

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

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

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

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

[6] Nakamatsu, K., Abe, J.M., and Akama, S., "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.

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

[8] Nakamatsu, K., Mita, Y., and 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.

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

[10] Nakamatsu, K., Abe, J.M., and Akama, S., 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* vol. 15, pp. 145–163, 2011.

 ISBN: 978-1-61208-184-7