Vietnam J Comput Sci (2014) 1:29–37
DOI 10.1007/s40595-013-0002-5

REGULAR PAPER

The paraconsistent process order control method

Kazumi Nakamatsu · Jair M. Abe

Received: 29 September 2013 / Accepted: 30 September 2013 / Published online: 9 November 2013

Abstract We have already developed some kinds of paraconsistent annotated logic programs. In this paper we propose the paraconsistent process order control method based on a paraconsistent annotated logic program called before–after extended vector annotated logic program with strong negation (bf-EVALPSN) with a small example of pipeline process order verification. Bf-EVALPSN can deal with before–after relations between two processes (time intervals) in its annotations, and its reasoning system consists of two kinds of inference rules called the basic bf-inference rule and the transitive bf-inference rule. We introduce how the bf-EVALPSN-based reasoning system can be applied to the safety verification for process order.

Keywords Paraconsistent annotated logic program · Before–after relation · Bf-EVALPSN · Process order control

1 Introduction

A family of paraconsistent logic called annotated logics $P\mathcal{T}$ was proposed by da Costa et al. [4]. They can deal with inconsistency with many truth values called *annotations*, although the semantics of annotated logics is basically two valued. The paraconsistent annotated logic has been developed from the viewpoint of logic programming [3], aiming at application to computer science. Furthermore, we have developed the paraconsistent annotated logic program to deal with inconsistency and some kinds of non-monotonic reasoning in a framework of annotated logic programming by using ontological (strong) negation and the stable model semantics [6], which is called annotated logic program with strong negation (ALPSN for short). Later, to deal with defeasible reasoning [14], we proposed a new version of ALPSN called vector annotated logic program with strong negation (VALPSN for short) and applied it to resolving conflicts [7]. Furthermore, we have extended VALPSN to deal with deontic notions (obligation, forbiddance, etc.) and named extended VALPSN (EVALPSN for short) [8,9]. We have shown that EVALPSN can deal with defeasible deontic reasoning and the safety verification for process control.

Considering the safety verification for process control, there are many cases in which the safety verification for process order is significant. For example, suppose a pipeline network in which two kinds of liquids, nitric acid and caustic soda, are used for cleaning the pipelines. If those liquids are processed continuously and mixed in the same pipeline by accident, explosion by neutralization would be caused. To avoid such a dangerous accident, the safety for process order should be strictly verified in a formal way. However, it seems to be a little difficult to utilize EVALPSN for the safety verification of process order control different from that of process control. Therefore, we have developed EVALPSN toward treating before–after relations between time intervals and applied it to process order control [11], which has been named before–after (bf)-EVALPSN. The before–after relation reasoning system based on bf-EVALPSN consists of two groups of inference rules called the basic bf-inference rule and the transitive bf-inference rule.

K. Nakamatsu (✉)
School of Human Science and Environment, University of Hyogo, Shinzaike, Himeji 670-0092, Japan
e-mail: nakamatu@shse.u-hyogo.ac.jp

J. M. Abe
ICET-Paulista University, São Paulo, SP, CEP 04026-022, Brazil

J. M. Abe
Institute of Advanced Studies, University of Sao Paulo, Cidade Universitaria, São Paulo, SP, CEP 05508-970, Brazil
e-mail: jairabe@uol.com.br

The original ideas of treating such before–after relations in logic were proposed for developing practical planning and natural language understanding systems by Allen [1] and Allen and Ferguson [2]. In his logic, before–after relations between two time intervals are represented in some special predicates and treated in a framework of first-order temporal logic. On the other hands, in bf-EVALPSN, before–after relations between two time intervals are regarded as paraconsistency between before and after degrees, and they can be represented more minutely in vector annotations of a special literal $R(p_i, p_j, t)$ representing the before–after relation between two processes (time intervals) at time t. Bf-EVALPSN-based before–after relation reasoning system consists of two kinds of efficient inference rules called the basic bf-inference rule and the transitive bf-inference rule that can be implemented as a bf-EVALPSN.

This paper is organized as follows: in Sect. 2, EVALPSN is reviewed briefly; in Sect. 3, bf-EVALPSN is formally defined and its simple reasoning example is introduced; in Sect. 4, the bf-EVALPSN reasoning system consisting of two kinds of inference rules is defined and explained in detail with some examples; in Sect. 5, the paraconsistent process order control method based on bf-EVALPSN reasoning is introduced with a small example of pipeline process order control; lastly, we conclude this paper.

2 Annotated logic program EVALPSN

In this section, we review EVALPSN briefly [9]. 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 and 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,

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

in Fig. 1. The ordering $(\preceq_v)$ of $\mathcal{T}_v(n)$ is defined as: let $(x_1, y_1), (x_2, y_2) \in \mathcal{T}_v(n)$,

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

For each extended vector annotated literal $p : [(i, j), \mu]$, the integer i denotes the amount of positive information to support the literal p and the integer j denotes that of negative 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,

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

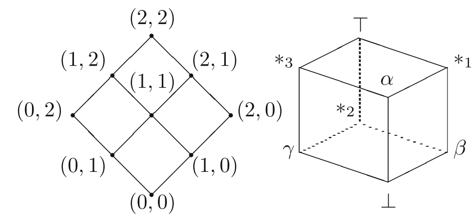

Fig. 1 Lattice $\mathcal{T}_v(2)$ and lattice $\mathcal{T}_d$

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 $\perp$(unknown), α(fact), β(obligation), γ(non-obligation), $*_1$(fact and obligation), $*_2$(obligation and non-obligation) and $*_3$(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 $[(i_1, j_1), \mu_1], [(i_2, j_2), \mu_2] \in \mathcal{T}_e$,

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

There are two kinds of *epistemic negations* ($\neg_1$ 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*)

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

If we regard the epistemic negations as syntactical operations, the epistemic negations followed by literals can be eliminated by the syntactical operations. For example, $\neg_1(p : [(2, 0), \alpha]) = p : [(0, 2), \alpha]$ 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 [4].

Definition 2 (*strong negation* $\sim$) Let F be any formula and $\neg$ be $\neg_1$ or $\neg_2$.

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

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 are non-negative integers and $\mu \in \{\alpha, \beta, \gamma\}$.

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

$$L_1 \wedge \ldots \wedge L_i \wedge \sim L_{i+1} \wedge \ldots \wedge \sim L_n \to L_0$$

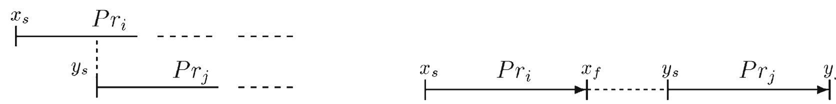

Fig. 2 Before (be)/after (af) and disjoint before (db)/after (da)

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.

3 Before–after EVALPSN

In this section, we review bf-EVALPSN. The details are found in [12,13].

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 $\mathrm{Pr}_m(p_m)$ and $\mathrm{Pr}_n(p_n)$, respectively, and before–after relations are represented paraconsistently in vector annotations.

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, there are 15 kinds of bf-relation according to before–after order between four start/finish times of two processes.

Before (be)/*after* (af) is defined according to the bf-relation between each start time of the 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 the left in Fig. 2.

We introduce other kinds of bf-relations as well as before (be)/after (af).

Disjoint before (db)/*after* (da) is defined as having a time lag between the earlier process finish time and the later one's start time; this is described on the right in Fig. 2.

Immediate before (mb)/*after* (ma) is defined as having no time lag between the earlier process finish time and the later one's start time; it is described on the left in Fig. 3.

Joint before (jb)/*after* (ja) is defined as two processes that overlap, where the earlier process had finished before the later one finished; it is described on the right in Fig. 3.

S-included before (sb)/*S-included after* (sa) is defined as two processes, where one had started before the other started, but finished at the same time; it is described on the left in Fig. 4.

Included before (ib)/*after* (ia) is defined as two processes, where one had started/finished before/after another one started/finished; it is described on the right in Fig. 4.

F-included before (fb)/*after* (fa) is defined as two processes that started at the same time, but with one finishing before another one finished; it is described in the left in Fig. 5.

Paraconsistent before–after (pba) is defined as having two processes that started at the same time and also finished at the same time; it is described on the right in Fig. 5.

The epistemic negation over bf-annotations, be, af, db, da, mb, ma, jb, ja, ib, ia, sb, sa, fb, fa and pba is defined and the complete lattice of bf-annotations is shown in Fig. 6.

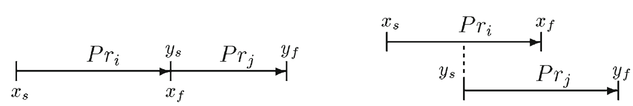

Fig. 3 Immediate before (mb)/after (ma) and joint before (jb)/after (ja)

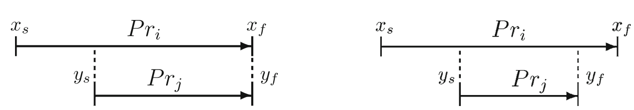

Fig. 4 S-included before (sb)/after (sa) and included before (ib)/after (ia)

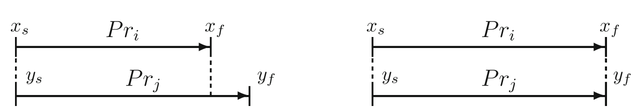

Fig. 5 F-included before (fb)/after (fa) and paraconsistent before–after (pba)

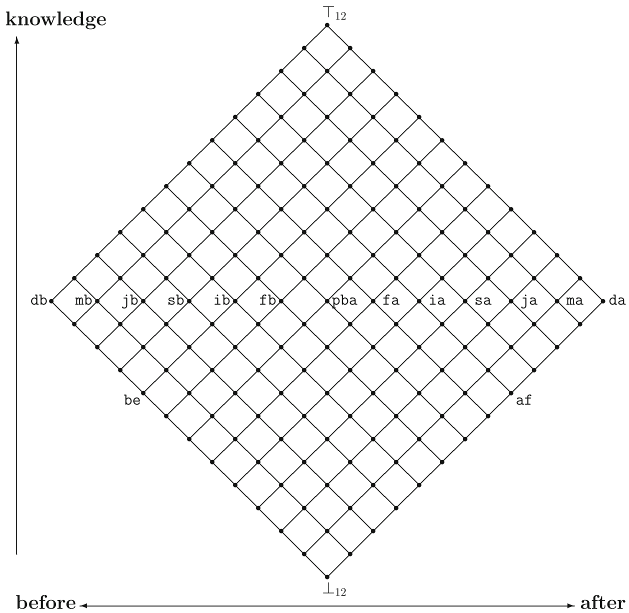

Fig. 6 The complete lattice $\mathcal{T}_v(12)_{\mathrm{bf}}$ for bf-annotations

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:

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

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

4 Reasoning system in bf-EVALPSN

To represent the *basic bf-inference rule* in bf-EVALPSN, we newly introduce two literals:

> $\mathrm{st}(p_i, t)$, which is interpreted as process Pr_i starts at time t, and
> $\mathrm{fi}(p_i, t)$, which is interpreted as process Pr_i finishes at time t.

Those literals are used for expressing process start/finish information and may have one of the vector annotations, $\{\perp(0, 0), \mathtt{t}(1, 0), \mathtt{f}(0, 1), \top(1, 1)\}$, where annotations $\mathtt{t}(1, 0)$ and $\mathtt{f}(0, 1)$ can be intuitively interpreted as “true” and “false”, respectively.

First of all, we introduce a group of basic bf-inference rules to be applied at the initial stage (time t_0), which are named (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 $\mathtt{be}(0, 8)$, which is the greatest lower bound of the set, $\{\mathtt{db}(0, 12),\ \mathtt{mb}(1, 11),\ \mathtt{jb}(2, 10),\ \mathtt{sb}(3, 9),\ \mathtt{ib}(4, 8)\}$.
> *(0,0)-rule-2* If both processes Pr_i and Pr_j have started at the same time, then it is reasonably anticipated that the bf-relation between processes Pr_i and Pr_j will be one of the bf-annotations, $\{\mathtt{fb}(5, 7),\ \mathtt{pba}(6, 6),\ \mathtt{fa}(7, 5)\}$ whose greatest lower bound is $(5, 5)$ (refer to Fig. 6). Therefore, the vector annotation $(0, 0)$ of bf-literal $R(p_i, p_j, t)$ should turn to $(5, 5)$.

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

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

$$R(p_i, p_j, t) : [(0, 0), \alpha] \wedge \mathrm{st}(p_i, t) : [\mathtt{t}, \alpha] \\ \wedge \mathrm{st}(p_j, t) : [\mathtt{t}, \alpha] \to R(p_i, p_j, t) : [(5, 5), \alpha] \qquad (2)$$

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

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

> *(0,8)-rule-1* If process Pr_i has finished before process Pr_j starts, and process Pr_j starts immediately after process Pr_i finished, then the vector annotation $(0, 8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{mb}(1, 11)$.
> *(0,8)-rule-2* If process Pr_i has finished before process Pr_j starts, and process Pr_j has not started immediately after process Pr_i finished, then the vector annotation $(0, 8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{db}(0, 12)$.
> *(0,8)-rule-3* If process Pr_j starts before process Pr_i finishes, then the vector annotation $(0, 8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $(2, 8)$ that is the greatest lower bound of the set, $\{\mathtt{jb}(2, 10),\ \mathtt{sb}(3, 9),\ \mathtt{ib}(4, 8)\}$.

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

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

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

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

(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 $(5, 5)$. We have the following inference rules to be applied for following $(0, 0)$-rule-2.

(5,5)-rule-1 If process Pr_i has finished before process Pr_j finishes, then the vector annotation $(5, 5)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{sb}(5, 7)$.
(5,5)-rule-2 If both processes Pr_i and Pr_j have finished at the same time, then the vector annotation $(5, 5)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{pba}(6, 6)$.
(5,5)-rule-3 If process Pr_j has finished before process Pr_i finishes, then the vector annotation $(5, 5)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{sa}(7, 5)$.

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

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

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

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

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

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

(2,8)-rule-1 If process Pr_i finished before process Pr_j finishes, then the vector annotation $(2, 8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{jb}(2, 10)$.
(2,8)-rule-2 If both processes Pr_i and Pr_j have finished at the same time, then the vector annotation $(2, 8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{fb}(3, 9)$.
(2,8)-rule-3 If process Pr_j has finished before Pr_i finishes, then the vector annotation $(2, 8)$ of bf-literal $R(p_i, p_j, t)$ should turn to $\mathtt{ib}(4, 8)$.

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

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

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

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

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

Now, we introduce the *transitive bf-inference rule*, which can reason a vector annotation of bf-literal transitively.

Suppose that there are three processes Pr_i, Pr_j and Pr_k starting sequentially, then we consider deriving 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 describe only the variation of vector annotations in the following rules.

Table 1 Application orders of basic bf-inference rules

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

Transitive bf-inference rules

TR0 $(0,0) \wedge (0,0) \rightarrow (0,0)$

TR1 $(0,8) \wedge (0,0) \rightarrow (0,8)$

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

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

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

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

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

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

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

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

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

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

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

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

TR1-4-3-6 $(4,8) \wedge (3,9) \rightarrow (4,8)$ (14)

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

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

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

TR1-4-6 $(4,8) \wedge (1,11) \rightarrow (2,8)$

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

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

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

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

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

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

TR2 $(5,5) \wedge (0,8) \rightarrow (0,8)$

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

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

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

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

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

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

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

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

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

TR3 $(5,5) \wedge (5,5) \rightarrow (5,5)$

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

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

Note (I) The name of a transitive bf-inference rule such as **TR1-4-3** indicates the application sequence of transitive bf-inference rules until the transitive bf-inference rule has been applied. For example, if the rule **TR1** has been applied, one of the rules **TR1-1**, **TR1-2**, ... or **TR1-5** should be applied at the following stage; and if the rule **TR1-4** has been applied after the rule **TR1**, one of the 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 the rules **TR1-1**, **TR1-2** or **TR1-3** has been applied after the rule **TR1**, there should be no transitive bf-inference rule to be applied at the following stage because one of bf-relations $\texttt{db}(0,12)$, $\texttt{mb}(1,11)$ has been derived.

Note (II) Transitive bf-inference rules,

TR1-4-2 (12), **TR1-4-3-2** (13), **TR1-4-6** (14),
TR1-5-1 (15), **TR2-2** (16), **TR2-3-2** (17),
TR2-5 (18), **TR3-1** (19)

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

5 The process order control method in bf-EVALPSN

In this section, we present the process order control method with a simple example for pipeline process order verification.

The process order control method has the following three steps:

Step 1 translate the safety properties of the process order control system into bf-EVALPSN;

Step 2 verify if permission for starting the process can be derived from the bf-EVALPSN in *step1* by the basic bf-inference rule and the transitive bf-inference rule or not.

The verification *step 2* can be carried out not only just before starting the process, but also at any time.

We assume a pipeline system consisting of two pipelines, PIPELINE-1 and 2, which deal with pipeline processes Pr_0, Pr_1, Pr_2 and Pr_3. The process schedule of those processes are shown in Fig. 7. Moreover, we assume that the pipeline system has four safety properties SPR $-i(i = 0, 1, 2, 3)$.

SPR-0 process Pr_0 must start before any other processes, and process Pr_0 must finish before process Pr_2 finishes,

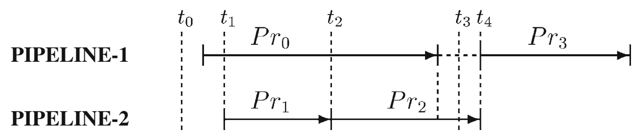

Fig. 7 Pipeline process schedule

SPR-1 process Pr_1 must start after process Pr_0 starts,
SPR-2 process Pr_2 must start immediately after process Pr_1 finishes,
SPR-3 process Pr_3 must start immediately after processes Pr_0 and Pr_2 finish.

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

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

where bf-EVALPSN clauses (20), (21) and (22) declare that if process Pr_0 has not started before other processes $\mathrm{Pr}_i (i = 1, 2, 3)$ start, it should be forbidden from starting each process $\mathrm{Pr}_i (i = 1, 2, 3)$; bf-EVALPSN clause (23) declares that if each process $\mathrm{Pr}_i (i = 1, 2, 3)$ is forbidden from starting, it should be permitted to start process Pr_0; and 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.

$$\text{SPR-1} \sim \mathrm{st}(p_1, t):[\mathtt{f}, \beta] \to \mathrm{st}(p_1, t):[\mathtt{f}, \gamma], \tag{25}$$
$$\sim \mathrm{fi}(p_1, t):[\mathtt{f}, \beta] \to \mathrm{fi}(p_1, t):[\mathtt{f}, \gamma], \tag{26}$$

where bf-EVALPSN clause (25)/(26) declares that if there is no forbiddance from starting/finishing process Pr_1, it should be permitted to start/finish process Pr_1, respectively.

$$\text{SPR-2} \sim R(p_2, p_1, t):[(11, 0), \alpha] \to \mathrm{st}(p_2, t):[\mathtt{f}, \beta], \tag{27}$$
$$\sim \mathrm{st}(p_2, t):[\mathtt{f}, \beta] \to \mathrm{st}(p_2, t):[\mathtt{f}, \gamma], \tag{28}$$
$$\sim R(p_2, p_0, t):[(10, 2), \alpha] \to \mathrm{fi}(p_2, t):[\mathtt{f}, \beta], \tag{29}$$
$$\sim \mathrm{fi}(p_2, t):[\mathtt{f}, \beta] \to \mathrm{fi}(p_2, t):[\mathtt{f}, \gamma], \tag{30}$$

where bf-EVALPSN clause (27) declares that if process Pr_1 has not finished before process Pr_2 starts, it should be forbidden from starting process Pr_2; the vector annotation $(11, 0)$ of bf-literal $R(p_2, p_1, t)$ is the greatest lower bound of $\{\mathtt{da}(12, 0), \mathtt{ma}(11, 1)\}$, which implies that process Pr_1 has finished before process Pr_2 starts; 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 bf-EVALPSN clauses (29) declare that if process Pr_0 has not finished before process Pr_2 finishes, it should be forbidden from finishing process Pr_2.

$$\text{SPR-3} \sim R(p_3, p_0, t):[(11, 0), \alpha] \to \mathrm{st}(p_3, t):[\mathtt{f}, \beta], \tag{31}$$
$$\sim R(p_3, p_1, t):[(11, 0), \alpha] \to \mathrm{st}(p_3, t):[\mathtt{f}, \beta], \tag{32}$$
$$\sim R(p_3, p_2, t):[(11, 0), \alpha] \to \mathrm{st}(p_3, t):[\mathtt{f}, \beta], \tag{33}$$
$$\sim \mathrm{st}(p_3, t):[\mathtt{f}, \beta] \to \mathrm{st}(p_3, t):[\mathtt{f}, \gamma], \tag{34}$$
$$\sim \mathrm{fi}(p_3, t):[\mathtt{f}, \beta] \to \mathrm{fi}(p_3, t):[\mathtt{f}, \gamma], \tag{35}$$

where bf-EVALPSN clauses (31), (32) and (33) declare that if one of processes $\mathrm{Pr}_i (i = 0, 1, 2)$ has not finished yet, it should be forbidden from starting process Pr_3; and bf-EVALPSN clauses (34)/(35) declare that if there is no forbiddance from starting/finishing process Pr_3, it should be permitted to start/finish process Pr_3, respectively.

Step 2 Here, we show how the bf-EVALPSN process order safety verification is carried out at five time points, t_0, t_1, t_2, t_3 and t_4 in the process schedule (Fig. 7). We consider five bf-relations between processes Pr_0, Pr_1, Pr_2 and Pr_3, represented by the vector annotations of bf-literals,

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

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

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

$$R(p_0, p_1, t_0):[(0, 0), \alpha], \tag{36}$$
$$R(p_1, p_2, t_0):[(0, 0), \alpha], \tag{37}$$
$$R(p_2, p_3, t_0):[(0, 0), \alpha] \tag{38}$$
$$R(p_0, p_2, t_0):[(0, 0), \alpha], \tag{39}$$
$$R(p_0, p_3, t_0):[(0, 0), \alpha] \tag{40}$$

are obtained by transitive bf-inference rule **TR0**; then, bf-EVALP clauses (36), (39) and (40) satisfy each body of bf-EVALPSN clauses (20), (21) and (22), respectively; therefore, the forbiddance,

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

from starting each process $\mathrm{Pr}_i (i = 1, 2, 3)$ is derived. Moreover, since bf-EVALP clauses (41), (42) and (43) satisfy the body of bf-EVALPSN clause (23), the permission for starting process Pr_0,

$$\mathrm{st}(p_0, t_0) : [\mathtt{f}, \gamma]$$

is derived; therefore, process Pr_0 is permitted to start at time t_0.

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

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

are obtained, where the bf-EVALP clause (44) is derived by basic bf-inference rule $(0, 0)$-rule-1. Moreover, the bf-EVALP clauses,

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

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

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

cannot be derived. Then, since there is no forbiddance (49), the body of bf-EVALPSN clause (25) is satisfied and the permission for starting process Pr_1,

$$\mathrm{st}(p_1, t_1) : [\mathtt{f}, \gamma]$$

is derived. On the other hand, since 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,

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

are derived; therefore, process Pr_1 is permitted to start at time t_1.

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

$$R(p_0, p_1, t_2) : [(4, 8), \alpha], \tag{50}$$
$$R(p_1, p_2, t_2) : [(1, 11), \alpha], \tag{51}$$
$$R(p_2, p_3, t_2) : [(0, 8), \alpha] \tag{52}$$

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,

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

are obtained by transitive bf-inference rules **TR1-4-6** and **TR1-2**, respectively. Then, since bf-EVALP clause (51) does not satisfy the body of bf-EVALPSN clause (27), the forbiddance from starting process Pr_2,

$$\mathrm{st}(p_2, t_2) : [\mathtt{f}, \beta] \tag{53}$$

cannot be derived. Since there is no forbiddance (53), 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, since bf-EVALP clause (53) satisfies the body of bf-EVALPSN clause (31), the forbiddance from starting process Pr_3,

$$t(p_3, t_2) : [\mathtt{f}, \beta]$$

is derived; therefore, process Pr_2 is permitted to start. However, process Pr_3 is still forbidden from starting at time t_2.

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

$$R(p_0, p_1, t_3) : [(4, 8), \alpha], \tag{54}$$
$$R(p_1, p_2, t_3) : [(1, 11), \alpha], \tag{55}$$
$$R(p_2, p_3, t_3) : [(0, 8), \alpha] \tag{56}$$

in which the vector annotations are the same as in the previous stage are obtained because bf-annotations of bf-EVALP clauses (54) and (55) have been already reasoned, and the before–after relation between processes Pr_2 and Pr_3 is the same as in the previous stage. Moreover, the bf-EVALP clauses,

$$R(p_0, p_2, t_3) : [(2, 10), \alpha], \tag{57}$$
$$R(p_0, p_3, t_3) : [(0, 12), \alpha] \tag{58}$$

are obtained, where bf-EVALP clause (57) is derived by basic bf-inference rule $(2, 8)$-rule-1. Then, bf-EVALP clause (57) 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 time t_3.

5th Stage (at time t_4) process Pr_2 has just finished and process Pr_3 has not started yet; then, the bf-EVALP clauses,

$$R(p_0, p_1, t_4):[(4, 8), \alpha], \quad (59)$$

$$R(p_1, p_2, t_4):[(1, 11), \alpha], \quad (60)$$

$$R(p_2, p_3, t_4):[(1, 11), \alpha], \quad (61)$$

$$R(p_0, p_2, t_4):[(2, 10), \alpha], \quad (62)$$

$$R(p_0, p_3, t_4):[(0, 12), \alpha] \quad (63)$$

are obtained. bf-EVALP clause (61) is derived by basic bf-inference rule (0, 8)-rule-2. Moreover, since bf-EVALP clauses (59), (62) and (63) do not satisfy the bodies of bf-EVALP clauses (31), (32) and (33), the forbiddance from starting process Pr_3,

$$\mathrm{st}(p_3, t_4):[\mathtt{f}, \beta] \quad (64)$$

cannot be derived. Therefore, the body of bf-EVALPSN clause (34) is satisfied, and the permission for starting process Pr_3,

$$\mathrm{st}(p_3, t_4):[\mathtt{f}, \gamma]$$

is derived. Therefore, process Pr_3 is permitted to start because processes Pr_0, Pr_1 and Pr_2 have finished at time t_4.

6 Concluding remarks

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

We would like to conclude this paper by describing the advantages and disadvantages of the process order control method based on bf-EVALPSN safety verification.

Advantages

– If a bf-EVALPSN is locally stratified [5], it can be easily implemented in Prolog, C language, Programmable Logic Controller (PLC) ladder program, etc. In practice, such control bf-EVALPSNs are locally stratified.
– It has been proved that EVALPSN can be implemented as electronic circuits on micro chips [10]. Therefore, if real-time processing is required in the system, the method might be very useful.
– The safety verification methods for both process control and process order control can be implemented under the same environment.

Disadvantages

– Since EVALPSN/bf-EVALPSN itself is basically not a specific tool of formal safety verification, it includes complicated and redundant expressions to construct safety verification systems. Therefore, it should be better to develop safety verification-oriented tool or programming language based on EVALPSN/bf-EVALPSN if EVALPSN/bf-EVALPSN can be applied to formal safety verification.

References

1. Allen, J.F.: Towards a general theory of action and time. Artif. Intell. **23**, 123–154 (1984)
2. Allen, J.F., Ferguson, G.: Actions and events in interval temporal logic. J. Log. Comput. **4**, 531–579 (1994)
3. Blair, H.A., Subrahmanian, V.S.: Paraconsistent logic programming. Theor. Comput. Sci. **68**, 135–154 (1989)
4. da Costa, N.C.A., et al.: The paraconsistent logics $P\mathcal{T}$. Zeitschrift für Mathematische Logic und Grundlangen der Mathematik **37**, 139–148 (1989)
5. Gelder, A.V., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. Assoc. Comput. Mach. ACM **38**, 620–650 (1991)
6. Nakamatsu, K., Suzuki, A.: Annotated semantics for default reasoning. In: Proceedings of the 3rd Pacific Rim International Conference of Artificial Intelligence (PRICAI94), pp. 180–186. International Academic Publishers (1994)
7. Nakamatsu, K.: On the relation between vector annotated logic programs and defeasible theories. Log. Log. Philos. **8**, 181–205 (2001)
8. Nakamatsu, K., et al.: A defeasible deontic reasoning system based on annotated logic programming. In: Proceedings of the 4th International Conference of Computing Anticipatory Systems (CASYS2000), AIP Conference Proceedings, vol. 573, pp. 609–620. American Institute of Physics (2001)
9. Nakamatsu, K., et al.: Annotated Semantics for Defeasible Deontic Reasoning. LNAI 2005, pp. 432–440. Springer, New York (2001)
10. Nakamatsu, K., Mita, Y., Shibata, T.: An intelligent action control system based on extended vector annotated logic program and its hardware implementation. J. Intell. Autom. Soft Comput. **13**, 222–237 (2007)
11. Nakamatsu, K.: The paraconsistent annotated logic program EVALPSN and its application. Comput. Intell. Compend. Stud. Comput Intell. **115**, 233–306 (2008). Springer
12. Nakamatsu, K., Abe, J.M.: The development of paraconsistent annotated logic program. Int. J. Reason.-Based Intell. Syst. **1**, 92–112 (2009)
13. Nakamatsu, K., Abe, J.M., Akama, S.: A logical reasoning system of process before–after relation based on a paraconsistent annotated logic program bf-EVALPSN. J. Knowl.-Based Intell. Eng. Syst. **15**, 145–163 (2011)
14. Nute, D.: Basic defeasible logics. In: del Cerro, L. F., Penttonen, M. (eds.) Intensional Logics for Programming, pp. 125–154. Oxford University Press, Oxford (1992)