IEICE TRANS. INF. & SYST., VOL.E90–D, NO.9 SEPTEMBER 2007

PAPER Special Section on Software Agent and Its Applications

Agent-Based Speculative Constraint Processing

Hiroshi HOSOBE[†a)], **Ken SATOH**[†], ***Members***, ***and*** **Philippe CODOGNET**[††], ***Nonmember***

SUMMARY In this paper, we extend our framework of speculative computation in multi-agent systems by introducing default constraints. In research on multi-agent systems, handling incomplete information due to communication failure or due to other agents' delay in communication is a very important issue. For a solution to this problem, we previously proposed speculative computation based on abduction in the context of master-slave multi-agent systems and gave a procedure in abductive logic programming. In our previous proposal, a master agent prepares a default value for a yes/no question in advance, and it performs speculative computation using the default without waiting for a reply to the question. This computation is effective unless the contradictory reply to the default is returned. In this paper, we formalize speculative constraint processing, and propose a correct operational model for such computation so that we can handle not only yes/no questions, but also more general types of questions.

key words: *agents, speculative computation, constraints*

1. Introduction

In most of current research on *multi-agent systems*, people assume that communication of agents is guaranteed. Also, when an agent asks a question of other agents, a process is suspended until some response from other agents is obtained. However, in real settings such as the Internet, this assumption is not guaranteed. Moreover, even if communication is guaranteed, when an agent needs to communicate with other agents and the computation in other agents takes much time before sending an answer, we encounter a similar situation.

For problem solving in the above situations, we previously proposed *speculative computation* [1] based on *abduction*. When communication is delayed or failed, we use a default hypothesis as a tentative answer, and continue computation. When some response is obtained, we check the consistency of the response with the default. If the response is consistent, then we continue the current computation; else if the response is inconsistent, we perform an alternative computation. This is desirable in the situation where some action should be taken in advance although a complete plan cannot be decided because of incompleteness of information.

In [1], we proposed an implementation of speculative computation for a master-slave multi-agent system and showed that abduction plays an important role in speculative computation.

As an example of speculative computation, consider the following meeting room reservation problem.

- There are three persons A, B, and C to attend a meeting during days 1, 2, and 3.
- If a person is available on a day, then he/she will attend the meeting on the day.
- We ask each person on which day he/she is free or busy.
- If all the persons are available on the same day, we can reserve a large room for the day.
- If only two persons are available on the same day, we can reserve a small room for the day.

Our task is to make a plan of possible meeting room reservation for days 1, 2 and 3. We assume that there is a considerable difference between a small room and a large room, and therefore we have to choose either of them.

Suppose that we receive an answer from A that A is free on day 1 and day 2 and busy on day 3, and an answer from B that B is free on day 2 and day 3 and busy on day 1, and also suppose that an answer from C is delayed.

If we follow the requirement that the communication must be completed before we take any further action, then we cannot make any reservation until we get an answer from C. In ordinary life, however, if we know that C is normally free on a specific day, then we can tentatively make a conclusion. For example, suppose that we know that C is normally free on day 2 and day 3. Then, we could make a reservation of the large room for day 2 if we decide to have a meeting on day 2, or make a reservation of the small room for day 3 if we decide to have a meeting on day 3.

In the previous framework, we can only ask a yes/no question, so we have to make a question of availability for each day. However, it is more efficient if we ask available days in one question.

In this paper, to solve the problem, we propose *speculative constraint processing* by introducing *constraints* into our framework so that we can ask other agents about possible values or constraints of questions (such as available days in the previous example).

The basic idea of this method is as follows.

1. The agent M prepares a default constraint for variables in a question in advance.
2. When an agent M asks a question of another agent S,

Manuscript received November 30, 2006.
Manuscript revised March 22, 2007.

[†]The authors are with the National Institute of Informatics, Tokyo, 101–8430 Japan
[††]The author is with the University of Paris 6, Paris, 75005 France and the Embassy of France in Japan, Tokyo, 106–8514 Japan.
a) E-mail: hosobe@nii.ac.jp

DOI: 10.1093/ietisy/e90–d.9.1354

the agent M uses the default constraint as a tentative answer and continues a computation along with the tentative answer.

3. When the response comes from the agent S, one of the following is performed.
   - If the response entails the default answer, then the agent M continues the computation.
   - If the response is inconsistent with the default answer, the agent M withdraws the computation process which uses the default answer, and then M restarts a computation with the true answer.
   - If the response does not entail the default answer, but is consistent with the default answer (that is, the conjunction of the response and the default answer is satisfiable), M continues the computation which uses the default, and simultaneously M starts alternative computation as well.

We assume that the default answer is prepared to cover the normal answer of each agent. This means that the response usually entails the default answer. We, therefore, expect that the computation which uses the default answer is usually effective.

Unlike the case of yes/no questions, however, we must consider the above third case. In the case of yes/no questions, the response either entails the default or is inconsistent with the default. On the other hand, in our extended setting, there is a possibility that the response neither entails the default nor contradicts with the default. In this case, we must consider both of the computations one of which uses the default and the other of which does not use the default.

In this paper, we restrict our attention to a master-slave multi-agent system. In this system, only the master agent performs speculative computation.

The rest of this paper is organized as follows. Section 2 describes related research. Section 3 provides a framework of constraint processing in master-slave systems. Section 4 introduces a speculative framework in master-slave systems. Section 5 proposes an operational model for speculative constraint processing, and illustrates the effect of speculative computation by an example. Section 6 proves the correctness of the operational model. Section 7 discusses the proposed framework. Finally, Section 8 mentions the conclusions and future work of this research.

2. Related Research

In computer science, there are studies on speculative computation such as optimistic transaction in databases, three-phase transition in fault-tolerant systems, efficient execution mechanisms for functional programming [2] and parallel logic programming [3]. We were inspired by some of these studies, and our work is regarded as an application of the above techniques to multi-agent systems.

There is research on constraint processing in concurrent environments [4] and distributed environments [5]. In particular, some work on agent-based distributed constraint processing handles meeting scheduling as a problem example. For instance, in [6], Hassine et al. use dynamic valued constraint satisfaction problems to allow modeling preferences of users and dynamic changes of problems, which they apply to meeting scheduling. Also, in [7], Wallace and Freuder discuss multi-agent constraint processing that keeps other agents' privacy as much as possible, which they illustrate with the meeting schedule problem. In this sense, their work is applicable to the same problem domain as our work. However, to our knowledge, such work on agent-based distributed constraint processing adopts the Constraint Satisfaction Problem (CSP) framework, whereas we employ the Constraint Logic Programming (CLP) framework. This causes fundamental differences between our and their work. The most fundamental difference is that CLP generates a set of CSPs in the process of executing a program. By contrast, in the CSP framework, problems are specified by an external entity such as a user (rather than by a program). Therefore, the CLP framework needs to maintain more complex information such as states of program execution. However, this additional complexity also gives more power to the CLP framework. For example, in the case of meeting scheduling, the CLP framework allows a program to generate different CSPs in different situations. Thus CLP allows richer specification of problems. Moreover, in this paper, we integrate speculative computation into CLP, by which we allow the master agent to do non-stop execution of programs by using default constraints, which is impossible in the case of the ordinary (non-speculative) CLP framework.

There is work on asynchronous algorithms for agent-based constraint processing. For example, [5] presents two algorithms called asynchronous backtracking and the asynchronous weak-commitment search for solving distributed CSPs. Also, [8] provides an asynchronous algorithm called Adopt for distributed constraint optimization. Such asynchronous algorithms allow agents to perform computation by using data that may be overridden in the future. Therefore, in a sense, their asynchronous operations are similar to speculative computation in our work. However, there is an essential difference between them. In such asynchronous algorithms, asynchronous operations are executed in the process of solving constraint problems, and asynchronism is purely a solving aspect. By contrast, in our work, speculativeness is associated with the master agent's execution of a program, which allows the programmer to control speculative computation (in fact, the programmer can disable speculative computation by providing no default constraints). This indicates that our speculative computation involves a modeling aspect as well as a solving aspect. Also, it should be noted that the CSP framework does not have the notion of programs, and therefore cannot introduce speculative computation in the same way as our framework.

Most related research to our work would be constraint programming languages such as AKL (Andorra Kernel Language) [9] and Oz [10], [11] which perform a kind of speculative computation. AKL allows local speculative variable bindings in a guard of each clause until one of guards suc-

ceeds and Oz can control multiple computation spaces each of which represents an alternative path of constraint processing. As far as we understand, however, speculative computation used in these languages is mainly motivated for or-parallel computing where multiple paths of computation are executed in parallel until one of the paths succeeds eventually. On the other hand, we regard speculative computation as default computation where most plausible paths of computation are executed. Moreover, they do not consider the usage of speculative computation for incomplete communication environments. However, we believe that Oz and AKL could be good platforms for implementation of speculative computation using defaults.

After the original work [1] of speculative computation in multi-agent systems was proposed, related research to speculative computation has been published [12]–[15]. [12] gives a local semantics of an agent by translating a program into another program where a time stamp is attached with each predicate. The revision of each agent is formalized as overriding the truth value of the predicate with the previous time stamp by the truth value of the predicate with the new time stamp. [13] gives a bottom-up evaluation of speculative computation by calculating every answer w.r.t. possible replies. [14] generalizes speculative computation for belief revision on the fly where belief revision occurs during reasoning, and [15] extends the master-slave setting to more general multi-agent systems.

3. Framework of Constraint Processing in Master-Slave Systems

In this section, we firstly provide a framework of constraint handling in a master-slave system. The framework follows the Constraint Logic Programming (CLP) framework.

Definition 1: A *constraint framework for a master-slave system* is a pair $\langle \Sigma, \mathcal{P} \rangle$ where

- Σ is a finite set of constants, each of which is called a *slave agent identifier*; when Q is an atom and S is a slave agent identifier, we call $Q@S$ an *askable atom*;
- $\mathcal{P}$ is a constraint logic program, that is, a set of rules in the form

  $$H \leftarrow C \parallel B_1, B_2, \ldots, B_n$$

  where
  - H is an atom called *head* of the above rule R denoted as $head(R)$;
  - C is a set of constraints called *body constraints* of R denoted as $const(R)$;
  - each of $B_1, \ldots, B_n$ is either an atom or an askable atom and we refer to $B_1, \ldots, B_n$ as *body* of R denoted as $body(R)$.

Example 1: The example of meeting room reservation in Sect. 1 (without speculative computation) is represented as the following framework $\langle \Sigma, \mathcal{P} \rangle$ with set constraints.

- $\Sigma = \{a, b, c\}$.
- $\mathcal{P}$ is the following set of rules:[†]

  $plan(small\_room, [X, Y], D) \leftarrow$
  $\quad D \in \{1, 2, 3\} \parallel meeting([X, Y], D).$
  $plan(large\_room, [a, b, c], D) \leftarrow$
  $\quad D \in \{1, 2, 3\} \parallel meeting([a, b, c], D).$
  $meeting([a, b], D) \leftarrow$
  $\quad \parallel available(a, D), available(b, D),$
  $\quad non\_available(c, D).$
  $meeting([b, c], D) \leftarrow$
  $\quad \parallel non\_available(a, D), available(b, D),$
  $\quad available(c, D).$
  $meeting([c, a], D) \leftarrow$
  $\quad \parallel available(a, D), non\_available(b, D),$
  $\quad available(c, D).$
  $meeting([a, b, c], D) \leftarrow$
  $\quad \parallel available(a, D), available(b, D),$
  $\quad available(c, D).$
  $available(P, D) \leftarrow \parallel free(D)@P.$
  $non\_available(P, D) \leftarrow \parallel busy(D)@P.$

The definition of the execution of the above framework is straightforwardly adapted from that of the usual CLP framework, and is as follows. For a non-askable atom in a goal, we reduce it into subgoals by the above rule and for an askable atom in a goal, a master agent asks a slave agent a query with the current constraints and waits for the answer. The answer is returned as a set of constraints on variables in the query. When the answer constraints are returned, they are added into the current constraints and the execution is resumed. The execution is completed when an empty goal is found.

Definition 2: A goal is in the form

$$\leftarrow C \parallel B_1, \ldots, B_n$$

where

- C is a set of constraints;
- each of $B_1, \ldots, B_n$ is either an atom or an askable atom.

For a semantics of the above framework, we handle an askable atom as if we knew the reply for a question in the askable atom beforehand.

Definition 3: A reply set is a set of rules in the form $Q@S \leftarrow C \parallel$ where $Q@S$ is an askable atom and each argument of Q is a variable and C is a set of constraints over those variables.

Note that a reply set is not necessarily specified for every askable atom. If a reply set is not specified for an askable atom, we regard an answer for the askable atom as undecided.

Definition 4: A reduction of a goal $\leftarrow C \parallel B_1, \ldots, B_n$ w.r.t.

[†]A string beginning with an upper case letter represents a variable and a string beginning with a lower case letter represents a constant.

a constraint logic program $\mathcal{P}$, a reply set $\mathcal{R}$ and a subgoal B_i is a goal $\leftarrow C' \parallel B'$ s.t.

- there is a rule R in $\mathcal{P} \cup \mathcal{R}$ s.t. $C \wedge \{B_i = head(R)\} \wedge const(R)$ is consistent, where $\{B_i = head(R)\}$ is a conjunction of constraints equating the arguments of atoms B_i and $head(R)$;
- $C' = C \wedge \{B_i = head(R)\} \wedge const(R)$;
- $B' = \{B_1, \ldots B_{i-1}, B_{i+1}, \ldots, B_n\} \cup body(R)$.

Definition 5: A derivation of a goal $\leftarrow C \parallel B_1, \ldots, B_n$ w.r.t. a constraint framework $\langle \Sigma, \mathcal{P} \rangle$ and a reply set $\mathcal{R}$ is a sequence of reductions $\leftarrow C \parallel B_1, \ldots, B_n, \ldots, \leftarrow C' \parallel \emptyset$ w.r.t. $\mathcal{P}$ and $\mathcal{R}$ where $\emptyset$ denotes an empty goal. C' is called an *answer constraint* w.r.t. the goal, the framework and the reply set.

Example 2: Suppose that a reply set is the following:
$free(D)@a \leftarrow D \in \{1\} \parallel .$
$busy(D)@a \leftarrow D \in \{2, 3\} \parallel .$
$free(D)@b \leftarrow D \in \{2, 3\} \parallel .$
$free(D)@c \leftarrow D \in \{1, 3\} \parallel .$
$busy(D)@c \leftarrow D \in \{2\} \parallel .$
Then, we have $R = small\_room$, $L = [X, Y]$, $X = b$, $Y = c$, $D \in \{3\}$ as an answer constraint w.r.t. a goal $\leftarrow \parallel plan(R, L, D)$, and the above framework and the above reply set. Note that we cannot conclude $R = small\_room$, $L = [X, Y]$, $X = c$, $Y = a$, $D \in \{1\}$ since there is no information on $busy(D)@b$.

4. Framework of Speculative Constraint Processing in Master-Slave Systems

Now we extend the constraint framework to perform a speculative computation.

Definition 6: A *speculative framework for a master-slave system* SF_{MS} is a triple $\langle \Sigma, \Delta, \mathcal{P} \rangle$ where

- Σ and $\mathcal{P}$ are the same as in the constraint framework;
- Δ is a set of rules in the form

  $$Q@S \leftarrow C \parallel$$

  called *default rule* w.r.t. $Q@S$, where $Q@S$ is an askable atom and C is a set of constraints called *default constraint for* $Q@S$; we denote a default rule w.r.t. $Q@S$ as $\delta(Q@S)$.

In the above definition an askable atom is used for two purposes. One is for a question sent by a master agent to a slave agent and the other is for a specification of default constraints. If there is no replies returned already for an askable atom, we use a default constraint for the askable atom as a tentative answer from the other agents.

Example 3: The example of meeting room reservation in Sect. 1 is represented as the following speculative framework $SF_{MS} = \langle \Sigma, \Delta, \mathcal{P} \rangle$, with the set constraints meaning that a is expected to be free on days 1 and 2 and busy on day 3, b is expected to be free on days 1 and 3, and c is expected to be free on day 3 and busy on day 2.

- Σ and $\mathcal{P}$ are the same as in Example 1.
- Δ is the set of the following rules:
  $free(D)@a \leftarrow D \in \{1, 2\} \parallel .$
  $busy(D)@a \leftarrow D \in \{3\} \parallel .$
  $free(D)@b \leftarrow D \in \{1, 3\} \parallel .$
  $free(D)@c \leftarrow D \in \{3\} \parallel .$
  $busy(D)@c \leftarrow D \in \{2\} \parallel .$

5. Operational Model for Speculative Constraint Processing

In this section, we propose an operational model of speculative computation. The execution of the speculative framework is based on two phases, the *process reduction phase* and the *fact arrival phase*. The process reduction phase is a normal execution of a program in a master agent and the fact arrival phase is an interruption phase when an answer arrives from a slave agent.

An active process consists of the following objects; (a) the current status of computation including the current constraint set and (b) a set of askable atoms which have been reduced already. Each process represents an alternative way of computation. Intuitively, processes are created when a choice point of computation is encountered such as case splitting or default handling. An active process ends successfully if all the computation is done and the constraints associated with reduced askable atoms have not been contradictory with the current reply set. A process fails when some used default constraints are found to contradict the newly returned answer.

In the process reduction phase, we reduce an active process set (see Definition 9 for the formal definition) into a new process set. During the reduction, for a process using a default constraint, the default is assumed unless an inconsistent constraint with the default has already been assumed or found to be true, whereas a process will be killed when the default used in the process contradicts the current constraint. When we start a speculative computation using the default value, we create another process to keep alternative computation which does not use the default value. We suspend this alternative process since the probability of failure of the alternate process is high.

When an answer comes from a slave agent, we consider the following three possibilities.

- If the answer entails the default, we continue the process using the default and remove the alternative suspended process which was created when a speculative computation starts.
- If the answer contradicts the default, we remove processes using the default and resume the alternative suspended processes.
- If the answer does not entail the default, but is consistent with the default, we not only continue the process using the default by adding the returned answer, but also resume the alternative process.

Initial Step: Let GS be an initial goal set. We give $\langle \leftarrow \| GS, \emptyset \rangle$ to a proof procedure; that is, $APS = \{\langle \leftarrow \| GS, \emptyset \rangle\}$. Let $SPS = AAQ = RF = \emptyset$.

Iteration Step: Do the following.

Case 1: If there is an active process $\langle \leftarrow C \| \emptyset, AD \rangle$, then output constraints C and a set of assumed askable atoms which is in AD and is not in $HEAD(RF)$ where $HEAD(RF) = \{head(R) \mid R \in RF\}$.

Case 2: Otherwise, select an active process $\langle \leftarrow C \| GS, AD \rangle$ from APS and select an atom L in GS. Let $APS' = APS - \{\langle \leftarrow C \| GS, AD \rangle\}$ and $GS' = GS - \{L\}$. For the selected atom L, do the following.

- If L is a non-askable atom, then
  $NewAPS = APS' \cup$
  $\{\langle \leftarrow (C \wedge \{B_i = head(R)\} \wedge const(R)) \| (body(R) \cup GS'), AD \rangle \mid C \wedge \{B_i = head(R)\} \wedge const(R)$ is consistent$\}$.
- If L is an askable atom $Q@S$, then do the following.
  - If $L \notin AAQ$, then send a question Q to a slave agent S and let $NewAAQ = AAQ \cup \{L\}$.
  - If $L \in AD$, then $NewAPS = APS' \cup \{\langle \leftarrow C \| GS', AD \rangle\}$.
  - Else if $(L \leftarrow C_r \|) \in RF$, then do the following.
    * If $C \wedge C_r$ is consistent, then $NewAPS = APS' \cup \{\langle \leftarrow C \wedge C_r \| GS', AD \rangle\}$
      else $NewAPS = APS'$.
  - Else if a default constraint C_d for L exists, then do the following.
    * If $C \wedge C_d$ is consistent, then $NewAPS = APS' \cup \{\langle \leftarrow C \wedge C_d \| GS', AD \cup \{L\} \rangle\}$
      else $NewAPS = APS'$.
    * If $C \wedge \neg C_d$ is consistent, then $NewSPS = SPS \cup \{\langle L, \leftarrow C \wedge \alpha \| GS', AD \rangle\}$ where $C \wedge \neg C_d \models \alpha$.

Fig. 1 Process reduction phase.

Let $NewRF = RF \cup \{Q@S \leftarrow C_r \|\}$. If a default constraint C_d for $Q@S$ exists, then do the following.

- If C_r entails C_d, then do the following.
  - $NewAPS = APS - DeletedAPS \cup AddedAPS$
    where $DeletedAPS = \{\langle \leftarrow C \| GS, AD \rangle \in APS \mid Q@S \in AD\}$
    and $AddedAPS = \{\langle \leftarrow C \wedge C_r \| GS, AD \rangle \mid \langle \leftarrow C \| GS, AD \rangle \in DeletedAPS$ and $C \wedge C_r$ is consistent$\}$.
  - $NewSPS = SPS - DeletedSPS \cup AddedSPS$
    where $DeletedSPS = \{\langle SG, \leftarrow C \| GS, AD \rangle \in SPS \mid SG = Q@S$ or $Q@S \in AD\}$
    and $AddedSPS = \{\langle SG, \leftarrow C \wedge C_r \| GS, AD \rangle \mid$
    $\langle SG, \leftarrow C \| GS, AD \rangle \in DeletedSPS$ and $Q@S \in AD$ and $C \wedge C_r$ is consistent$\}$.
- If C_r contradicts C_d, then do the following.
  - $NewAPS = APS - DeletedAPS \cup ResumedSPS$
    where $DeletedAPS = \{\langle \leftarrow C \| GS, AD \rangle \in APS \mid Q@S \in AD\}$
    and $ResumedSPS = \{\langle \leftarrow C \wedge C_r \| GS, AD \rangle \mid \langle Q@S, \leftarrow C \| GS, AD \rangle \in SPS$ and $C \wedge C_r$ is consistent$\}$.
  - $NewSPS = SPS - DeletedSPS$
    where $DeletedSPS = \{\langle SG, \leftarrow C \| GS, AD \rangle \in SPS \mid SG = Q@S$ or $Q@S \in AD\}$.
- If C_r does not entail C_d nor contradicts C_d, then do the following.
  - $NewAPS = APS - DeletedAPS \cup AddedAPS \cup ResumedSPS$
    where $DeletedAPS = \{\langle \leftarrow C \| GS, AD \rangle \in APS \mid Q@S \in AD\}$
    and $AddedAPS = \{\langle \leftarrow C \wedge C_r \| GS, AD \rangle \mid \langle \leftarrow C \| GS, AD \rangle \in DeletedAPS$ and $C \wedge C_r$ is consistent$\}$
    and $ResumedSPS = \{\langle \leftarrow C \wedge C_r \| GS, AD \rangle \mid \langle Q@S, \leftarrow C \| GS, AD \rangle \in SPS$ and $C \wedge C_r$ is consistent$\}$.
  - $NewSPS = SPS - DeletedSPS \cup AddedSPS$
    where $DeletedSPS = \{\langle SG, \leftarrow C \| GS, AD \rangle \in SPS \mid SG = Q@S$ or $Q@S \in AD\}$
    and $AddedSPS = \{\langle SG, \leftarrow C \wedge C_r \| GS, AD \rangle \mid$
    $\langle SG, \leftarrow C \| GS, AD \rangle \in DeletedSPS$ and $Q@S \in AD$ and $C \wedge C_r$ is consistent$\}$.

Fig. 2 Fact arrival phase.

5.1 Preliminary Definitions

We define the following objects for process reduction.

Definition 7: An *active process* is a pair $\langle \leftarrow C \| GS, AD \rangle$ where

- $\leftarrow C \| GS$ is a goal consisting of a set of atoms GS and a set of constraints C;
- AD is a set of askable atoms assumed already called *assumed askable atoms*.

Definition 8: A *suspended process* is a triple $\langle SG, \leftarrow C \| GS, AD \rangle$ where

- SG is an askable atom called a *suspended atom*;
- $\leftarrow C \| GS$ is a goal;

1. Active: $\langle(\leftarrow \| \mathit{plan}(R, L, D)), \emptyset\rangle$
2. Active: $\langle(\leftarrow D{\in}\{1,2,3\}, R = \mathit{small\_room}, L = [X, Y] \| \mathit{plan}(\mathit{small\_room}, [X, Y], D)), \emptyset\rangle$,
   $\langle(\leftarrow D{\in}\{1,2,3\}, R = \mathit{large\_room}, L = [a, b, c] \| \mathit{plan}(\mathit{large\_room}, [a, b, c], D)), \emptyset\rangle$
3. Active: $\langle(\leftarrow D{\in}\{1,2,3\}, R = \mathit{small\_room}, L = [X, Y] \| \mathit{meeting}([X, Y], D)), \emptyset\rangle$,
   $\langle(\leftarrow D{\in}\{1,2,3\}, R = \mathit{large\_room}, L = [a, b, c] \| \mathit{plan}(\mathit{large\_room}, [a, b, c], D)), \emptyset\rangle$
4. Active: $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{ab} \| \mathit{available}(a, D), \mathit{available}(b, D), \mathit{non\_available}(c, D)), \emptyset\rangle$,
   $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{bc} \| \mathit{non\_available}(a, D), \mathit{available}(b, D), \mathit{available}(c, D)), \emptyset\rangle$,
   $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{ca} \| \mathit{available}(a, D), \mathit{non\_available}(b, D), \mathit{available}(c, D)), \emptyset\rangle$,
   $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{abc} \| \mathit{plan}(\mathit{large\_room}, [a, b, c], D)), \emptyset\rangle$
5. Active: $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{ab} \| \mathit{free}(D)@a, \mathit{available}(b, D), \mathit{non\_available}(c, D)), \emptyset\rangle, P_{bc}, P_{ca}, P_{abc}$
6. $\mathit{free}(D)$ is asked of a, and since $(\mathit{free}(D)@a \leftarrow D{\in}\{1,2\} \|) \in \Delta$,
   Active: $\langle(\leftarrow D{\in}\{1,2\}, \theta_{ab} \| \mathit{available}(b, D), \mathit{non\_available}(c, D)), \{\mathit{free}(D)@a\}\rangle, P_{bc}, P_{ca}, P_{abc}$
   Suspended: $\langle \mathit{free}(D)@a, (\leftarrow D{\in}\{3\}, \theta_{ab} \| \mathit{available}(b, D), \mathit{non\_available}(c, D)), \emptyset\rangle$
7. Active: $\langle(\leftarrow D{\in}\{1,2\}, \theta_{ab} \| \mathit{free}(D)@b, \mathit{non\_available}(c, D)), \{\mathit{free}(D)@a\}\rangle, P_{bc}, P_{ca}, P_{abc}$
   Suspended: SP_1
8. $\mathit{free}(D)$ is asked of b, and since $(\mathit{free}(D)@b \leftarrow D{\in}\{1,3\} \|) \in \Delta$,
   Active: $\langle(\leftarrow D{\in}\{1\}, \theta_{ab} \| \mathit{non\_available}(c, D)), \{\mathit{free}(D)@a, \mathit{free}(D)@b\}\rangle, P_{bc}, P_{ca}, P_{abc}$
   Suspended: $SP_1, \langle \mathit{free}(D)@b, (\leftarrow D{\in}\{2\}, \theta_{ab} \| \mathit{non\_available}(c, D)), \{\mathit{free}(D)@a\}\rangle$
9. Active: $\langle(\leftarrow D{\in}\{1\}, \theta_{ab} \| \mathit{busy}(D)@c), \{\mathit{free}(D)@a, \mathit{free}(D)@b\}\rangle, P_{bc}, P_{ca}, P_{abc}$
   Suspended: SP_1, SP_2
10. $\mathit{busy}(D)$ is asked of c, and since $(\mathit{busy}(D)@c \leftarrow D{\in}\{2\} \|) \in \Delta$,
    Active: P_{bc}, P_{ca}, P_{abc}
    Suspended: $SP_1, SP_2, \langle \mathit{busy}(D)@c, (\leftarrow D{\in}\{1\}, \theta_{ab} \| \emptyset), \{\mathit{free}(D)@a, \mathit{free}(D)@b\}\rangle$
11. Active: $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{bc} \| \mathit{busy}(D)@a, \mathit{available}(b, D), \mathit{available}(c, D)), \emptyset\rangle, P_{ca}, P_{abc}$
    Suspended: SP_1, SP_2, SP_3
12. $\mathit{busy}(D)$ is asked of a, and since $(\mathit{busy}(D)@a \leftarrow D{\in}\{3\} \|) \in \Delta$,
    Active: $\langle(\leftarrow D{\in}\{3\}, \theta_{bc} \| \mathit{available}(b, D), \mathit{available}(c, D)), \{\mathit{busy}(D)@a\}\rangle, P_{ca}, P_{abc}$
    Suspended: $SP_1, SP_2, SP_3, \langle \mathit{busy}(D)@a, (\leftarrow D{\in}\{1,2\}, \theta_{bc} \| \mathit{available}(b, D), \mathit{available}(c, D)), \emptyset\rangle$
13. Active: $\langle(\leftarrow D{\in}\{3\}, \theta_{bc} \| \mathit{free}(D)@b, \mathit{available}(c, D)), \{\mathit{busy}(D)@a\}\rangle, P_{ca}, P_{abc}$
    Suspended: SP_1, SP_2, SP_3, SP_4
14. Since $\mathit{free}(D)@b$ has been asked already, we do not send a question to b. Since $(\mathit{free}(D)@b \leftarrow D{\in}\{1,3\} \|) \in \Delta$,
    Active: $\langle(\leftarrow D{\in}\{3\}, \theta_{bc} \| \mathit{available}(c, D)), \{\mathit{busy}(D)@a, \mathit{free}(D)@b\}\rangle, P_{ca}, P_{abc}$
    Suspended: SP_1, SP_2, SP_3, SP_4
15. Active: $\langle(\leftarrow D{\in}\{3\}, \theta_{bc} \| \mathit{free}(D)@c), \{\mathit{busy}(D)@a, \mathit{free}(D)@b\}\rangle, P_{ca}, P_{abc}$
    Suspended: SP_1, SP_2, SP_3, SP_4
16. Suppose that $\mathit{free}(D)@b \leftarrow D{\in}\{3\} \|$ is returned from b.
    Active: $\langle(\leftarrow D{\in}\{3\}, \theta_{bc} \| \mathit{free}(D)@c), \{\mathit{busy}(D)@a, \mathit{free}(D)@b\}\rangle, P_{ca}, P_{abc}$
    Suspended: SP_1, SP_4
17. Suppose that $\mathit{busy}(D)@a \leftarrow D{\in}\{2,3\} \|$ is returned from a.
    Active: $\langle(\leftarrow D{\in}\{3\}, \theta_{bc} \| \mathit{free}(D)@c), \{\mathit{busy}(D)@a, \mathit{free}(D)@b\}\rangle$,
    $\langle(\leftarrow D{\in}\{2\}, \theta_{bc} \| \mathit{available}(b, D), \mathit{non\_available}(c, D)), \emptyset\rangle, P_{ca}, P_{abc}$
    Suspended: SP_1
18. $\mathit{free}(D)$ is asked of c, and since $(\mathit{free}(D)@c \leftarrow D{\in}\{3\} \|) \in \Delta$,
    Active: $\langle(\leftarrow D{\in}\{3\}, \theta_{bc} \| \emptyset), \{\mathit{busy}(D)@a, \mathit{free}(D)@b, \mathit{free}(D)@c\}\rangle$,
    $\langle(\leftarrow D{\in}\{2\}, \theta_{bc} \| \mathit{available}(b, D), \mathit{non\_available}(c, D)), \emptyset\rangle, P_{ca}, P_{abc}$
    Suspended: SP_1
19. $(D{\in}\{3\}, R = \mathit{small\_room}, L = [X, Y], X = b, Y = c)$ is output as an answer constraint, and $\{\mathit{free}(D)@c\}$ as a set of assumed askable atoms.

Fig. 3 Execution trace for the program in Example 3. Here "$R = \mathit{small\_room}$, $L = [X, Y]$, $X = a$, $Y = b$" is denoted as θ_{ab}, "$R = \mathit{small\_room}$, $L = [X, Y]$, $X = b$, $Y = c$" as θ_{bc}, "$R = \mathit{small\_room}$, $L = [X, Y]$, $X = c$, $Y = a$" as θ_{ca}, "$R = \mathit{large\_room}$, $L = [a, b, c]$" as θ_{abc}, $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{bc} \| \mathit{non\_available}(a, D), \mathit{available}(b, D), \mathit{available}(c, D)), \emptyset\rangle$ as P_{bc}, $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{ca} \| \mathit{available}(a, D), \mathit{non\_available}(b, D), \mathit{available}(c, D)), \emptyset\rangle$ as P_{ca}, $\langle(\leftarrow D{\in}\{1,2,3\}, \theta_{abc} \| \mathit{plan}(\mathit{large\_room}, [a, b, c], D)), \emptyset\rangle$ as P_{abc}, $\langle \mathit{free}(D)@a, (\leftarrow D{\in}\{3\}, \theta_{ab} \| \mathit{available}(b, D), \mathit{non\_available}(c, D)), \emptyset\rangle$ as SP_1, $\langle \mathit{free}(D)@b, (\leftarrow D{\in}\{2\}, \theta_{ab} \| \mathit{non\_available}(c, D)), \{\mathit{free}(D)@a\}\rangle$ as SP_2, $\langle \mathit{busy}(D)@c, (\leftarrow D{\in}\{1\}, \theta_{ab} \| \emptyset), \{\mathit{free}(D)@a, \mathit{free}(D)@b\}\rangle$ as SP_3, and $\langle \mathit{busy}(D)@a, (\leftarrow D{\in}\{1,2\}, \theta_{bc} \| \mathit{available}(b, D), \mathit{available}(c, D)), \emptyset\rangle$ as SP_4.

- AD is a set of assumed askable atoms.

We use the following four sets for process reduction.

Definition 9:

- *An active process set* APS is a set of active processes.
- *A suspended process set* SPS is a set of suspended processes.
- *Already asked queries* AAQ are a set of askable atoms.
- *Returned facts* RF are a set of rules in the form $Q@S \leftarrow C \|$ where $Q@S$ is an askable atom and C is a set of constraints.

AAQ is used to avoid asking redundant questions of slave agents, and RF is a set of true constraints returned from slave agents about askable atoms and in the form $Q@S \leftarrow C \|$.

5.2 Process Reduction Phase

Figure 1 shows the procedure for the process reduction phase. Here we specify changed APS, SPS, AAQ, RF as $NewAPS$, $NewSPS$, $NewAAQ$, $NewRF$; otherwise each of APS, SPS, AAQ, and RF is unchanged.

If the constraint solver can manipulate any logical combination of constraints such as negations and disjunctions, the constraint α, which is used in this procedure, can be equivalent to $C \wedge \neg C_d$. If the solver cannot do so, α might be a partial constraint weaker than $C \wedge \neg C_d$ or sometimes no constraint.

5.3 Fact Arrival Phase

Figure 2 gives the procedure for the fact arrival phase. Here we suppose that a constraint is returned from a slave agent S for a question $Q@S$. We denote the returned constraint as $Q@S \leftarrow C_r \|$. Then we execute the procedure after one step of process reduction is finished.

5.4 Example

We illustrate the execution of this operational model, using the program in Example 3. We take the following strategy for process reduction.

- When we reduce an atom, new processes are created along with the rule order in the program which are unifiable with the atom.
- We always select a newly created or a newly resumed process and a left-most atom.

Figure 3 shows the execution trace for $plan(R, L, D)$ based on this strategy. We assume that answers from the agents b and a come at step 16 and step 17 respectively. We show changes of active processes and suspended processes during the execution.

At Step 6, we split a process into two processes; one active process using a default constraint and one suspended process using the negation of the default constraint. If we had to wait for an answer from the agent a, we would have to suspend this process. This is an effect of speculative computation.

At Step 16, the answer constraint for $free(D)$ is returned from b. Since this constraint entails the default constraint, nothing changes and we can continue the reduction. Therefore, in this case we receive a benefit of speculative computation.

At Step 17, the answer constraint for $busy(D)$ is returned from a. Since this constraint does not entail the default constraint nor contradicts the default constraint, we not only continue a process using the default constraint, but also resume a process using the negation of the default constraint. This is the difference between our previous work and the mechanism proposed in this paper.

6. Correctness of the Operational Model

The following theorem shows the correctness of the operational model presented in the previous section.

Theorem 1: Let $SF_{MS} = \langle \Sigma, \Delta, \mathcal{P} \rangle$ be a speculative framework. Let GS_{init} be an initial goal set, AD be a set of used assumed askable atoms, C be an answer constraint obtained from the operational model, and RF be a set of constraints returned from the other agents when the answer is obtained. Then, there exists an answer constraint C' w.r.t. the constraint framework $\langle \Sigma, \mathcal{P} \rangle$ and the reply set $RF \cup \{\delta(Q@S) | Q@S \in AD\}$ s.t. $\pi_V(C)$ entails $\pi_V(C')$, where V is the set of the variables that occur in GS_{init}, and π_V is the projection of constraints onto V.

See Appendix for the proof of this theorem.

7. Discussion

Speculative constraint processing requires appropriate default constraints to obtain good results based on speculative computation. Therefore, its success relies on problem domains to which it is applied. For example, the problems of meeting room reservation and meeting scheduling are promising examples for speculative constraint processing, since people usually have regular schedules that are appropriate to default constraints. However, even if completely inappropriate default constraints are specified, speculative constraint processing gives performance comparable to non-speculative computation. This is because, in such a case, the fact arrival phase immediately kills the active processes based on the inappropriate default constraints, and then resumes the suspended processes that have been waiting for the answer constraints. Note that this is similar to the non-speculative case because non-speculative computation must wait for answer constraints without proceeding its computation process. Also, it should be noted that, when an answer constraint does not entail but is consistent with the default, speculative constraint processing can immediately output corrected partial results.

As described in Sect. 1, speculative constraint processing handles more expressive questions than our previous speculative computation framework that allows only yes/no questions. However, speculative constraint processing currently does not support negation as failure that is supported in the previous yes/no-type framework; in this sense, speculative constraint processing is not a complete generalization of the yes/no-type framework. Since negation is often useful for modeling problems, it is necessary to further extend the speculative constraint processing framework to handle negation as failure.

8. Conclusions and Future Work

The contributions of this paper are as follows.

- We presented speculative constraint processing in multi-agent systems.
- We proposed a correct operational model of speculative constraint processing in master-slave multi-agent systems.

The following issues remain for future research.

- Extensions to other kinds of multi-agent systems such as the one where every agent can perform speculative computation.
- Extensions in order to handle negation as failure.
- Decision theoretic analysis on effects of speculative computation.

References

[1] K. Satoh, K. Inoue, K. Iwanuma, and C. Sakama, "Speculative computation by abduction under incomplete communication environments," Proc. Intl. Conf. on Multi-Agent Systems (ICMAS2000), pp.263–270, 2000.

[2] F.W. Burton, "Speculative computation, parallelism, and functional programming," IEEE Trans. Comput., vol.C-34, no.12, pp.1190–1193, 1985.

[3] S. Gregory, "Experiment with speculative parallelism in Parlog," Proc. Intl. Symp. on Logic Programming (ILPS'93), pp.370–387, 1993.

[4] V.A. Saraswat, Concurrent constraint programming, Logic Programming, MIT Press, 1993.

[5] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," IEEE Trans. Knowl. Data Eng., vol.10, no.5, pp.673–685, 1998.

[6] A.B. Hassine, X. Defago, and T.B. Ho, "Agent-based approach to dynamic meeting scheduling problems," Proc. Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS2004), pp.1132–1139, 2004.

[7] R.J. Wallace and E.C. Freuder, "Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving," Artif. Intell., vol.161, no.1-2, pp.209–227, 2005.

[8] P.J. Modi, W.M. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," Artif. Intell., vol.161, no.1-2, pp.149–180, 2005.

[9] S. Janson and S. Haridi, "Programming paradigms of the Andorra Kernel Language," Proc. Intl. Symp. on Logic Programming (ISLP'91), pp.167–186, 1991.

[10] G. Smolka, "The Oz programming model," Computer Science Today: Recent Trends and Developments, LNCS, vol.1000, pp.324–343, Springer, 1995.

[11] C. Schulte, Programming Constraint Services: High-Level Programming of Standard and New Constraint Services, LNCS, vol.2302, Springer, 2002.

[12] C. Sakama, K. Inoue, K. Iwanuma, and K. Satoh, "A defeasible reasoning system in multi-agent environments," Proc. CL2000 Workshop on Computational Logic in Multi-Agent Systems (CLIMA-00), pp.1–6, 2000.

[13] K. Inoue, S. Kawaguchi, and H. Haneda, "Controlling speculative computation in multi-agent environments," Proc. ICLP2001 Workshop on Computational Logic in Multi-Agent Systems (CLIMA-01), pp.9–18, 2001.

[14] K. Satoh, "Speculative computation and abduction for an autonomous agent," Proc. Intl. Workshop on Non-Monotonic Reasoning (NMR2002), pp.191–199, 2002.

[15] K. Satoh and K. Yamamoto, "Speculative computation with multi-agent belief revision," Proc. Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS2002), pp.897–904, 2002.

Appendix: Proof of Theorem 1

We show that a more general property holds for any existing active or suspended process at any "step" in the process reduction or fact arrival phase. By a "step," we mean the execution of operations in the iteration step part of the process reduction phase or the whole of the fact arrival phase from its beginning to its end, without returning to the beginning, and without transferring to the other phase. Then the property that we show is the following: $(*)$ at any k-th step, for any process P, there exists a sequence of reductions

$$\text{“} \leftarrow \| GS_{init} \text{”}, \ldots, \text{“} \leftarrow C'_P \| GS_P \text{”}$$

w.r.t. $\mathcal{P}$ and

$$\mathcal{R}_P^{(k)} = RF_k \cup \{\delta(Q@S) | Q@S \in AD_P\}$$

s.t.

$$\pi_V(C_P) \text{ entails } \pi_V(C'_P),$$

where $C_P = C_{P_a}$, $GS_P = GS_{P_a}$, and $AD_P = AD_{P_a}$ if the process P is an active process $P_a = \langle \leftarrow C_{P_a} \| GS_{P_a}, AD_{P_a} \rangle$, and $C_P = C_{P_s}$, $GS_P = \{SG_{P_s}\} \cup GS_{P_s}$, and $AD_P = AD_{P_s}$ if the process P is a suspended process $P_s = \langle SG_{P_s}, \leftarrow C_{P_s} \| GS_{P_s}, AD_{P_s} \rangle$, and RF_k is the set of the returned constraints at the k-th step.

Below we prove the property $(*)$ by induction on the progress of process reduction and fact arrival steps.

Induction base. In the initial step, an active process $\langle \leftarrow \| GS_{init}, \emptyset \rangle$ is created, which satisfies $(*)$.

Induction step. Assume that, at the k-th step, the property $(*)$ holds.

Now consider the $(k+1)$-th step. It is straightforward to show that $(*)$ holds for the process reduction phase.

Here we consider the processing of a returned answer $Q@S \leftarrow C_r \|$ in the fact arrival phase. Assume that there is a default constraint C_d for $Q@S$.

Let $P_a = \langle \leftarrow C_{P_a} \| GS_{P_a}, AD_{P_a} \rangle$ be any existing active process s.t. $Q@S \in AD_{P_a}$. Note that P_a is deleted at this step.

Consider the newly added active process that is created from P_a. We have the following three cases.

1. *Case C_r entails C_d.* If $C_{P_a} \wedge C_r$ is consistent, the active process $P'_a = \langle \leftarrow C_{P_a} \wedge C_r \| GS_{P_a}, AD_{P_a} \rangle$ is created, and we have $\mathcal{R}_{P'_a}^{(k+1)} = \mathcal{R}_{P_a}^{(k)} \cup \{Q@S \leftarrow C_r \|\} \setminus \{Q@S \leftarrow C_d \|\}$. By the induction hypothesis, P_a satisfies $(*)$ for some C'_{P_a}; that is, there is a sequence of reductions "$\leftarrow \| GS_{init}$", ..., "$\leftarrow C_1 \| \{Q@S\} \cup GS$", "$\leftarrow C_1 \wedge C_d \| GS$", ..., "$\leftarrow C_1 \wedge C_d \wedge C_2 \| GS_{P_a}$" w.r.t. $\mathcal{P}$ and $\mathcal{R}_{P_a}^{(k)}$ s.t. $\pi_V(C_{P_a})$ entails $\pi_V(C_1 \wedge C_d \wedge C_2)$, where C_1 and C_2 are the constraints obtained before and after processing $Q@S$ respectively, and $C_1 \wedge C_d \wedge C_2 = C'_{P_a}$. Then we can consider the sequence of reductions "$\leftarrow \| GS_{init}$", ..., "$\leftarrow C_1 \| \{Q@S\} \cup GS$",

"$\leftarrow C_1 \wedge C_r \| GS$", ..., "$\leftarrow C_1 \wedge C_r \wedge C_2 \| GS_{P_a}$" w.r.t. $\mathcal{P}$ and $\mathcal{R}^{(k+1)}_{P'_a}$. Since C_r entails C_d, and also since $\pi_V(C_{P_a})$ entails $\pi_V(C_1 \wedge C_d \wedge C_2)$, $\pi_V(C_{P_a} \wedge C_r)$ entails $\pi_V(C_1 \wedge C_r \wedge C_2)$. Thus $(*)$ holds for this new process.
2. *Case C_r contradicts C_d.* No such active process is created from P_a.
3. *Case C_r does not entail C_d nor contradicts C_d.* If $C_{P_a} \wedge C_r$ is consistent, an active process is created from P_a, for which we can show $(*)$ in a similar way to case 1.

Next, let $P_s = \langle SG_{P_s}, \leftarrow C_{P_s} \| GS_{P_s}, AD_{P_s} \rangle$ be any existing suspended process s.t. $SG_{P_s} = Q@S$ or $Q@S \in AD_{P_s}$. Note that P_s is deleted at this step.

Consider the newly added active process that is created from P_s s.t. $SG_{P_s} = Q@S$ (which corresponds to the resumed process). We have the following three cases.

1. *Case C_r entails C_d.* No such active process is created from P_s.
2. *Case C_r contradicts C_d.* If $C_{P_s} \wedge C_r$ is consistent, the active process $P'_a = \langle \leftarrow C_{P_s} \wedge C_r \| GS_{P_s}, AD_{P_s} \rangle$ is created, and we have $\mathcal{R}^{(k+1)}_{P'_a} = \mathcal{R}^{(k)}_{P_s} \cup \{Q@S \leftarrow C_r \|\} \setminus \{Q@S \leftarrow C_d \|\}$. By the induction hypothesis, P_s satisfies $(*)$ for some C'_{P_s}; that is, there is a sequence of reductions "$\leftarrow \| GS_{init}$", ..., "$\leftarrow C'_{P_s} \|\{Q@S\} \cup GS_{P_s}$" w.r.t. $\mathcal{P}$ and $\mathcal{R}^{(k)}_{P_s}$ s.t. $\pi_V(C_{P_s})$ entails $\pi_V(C'_{P_s})$. Then we can consider the sequence of reductions "$\leftarrow \| GS_{init}$", ..., "$\leftarrow C'_{P_s} \|\{Q@S\} \cup GS_{P_s}$", "$\leftarrow C'_{P_s} \wedge C_r \| GS_{P_s}$" w.r.t. $\mathcal{P}$ and $\mathcal{R}^{(k+1)}_{P'_a}$. Since $\pi_V(C_{P_s})$ entails $\pi_V(C'_{P_s})$, $\pi_V(C_{P_s} \wedge C_r)$ entails $\pi_V(C'_{P_s} \wedge C_r)$. Thus $(*)$ holds for this new process.
3. *Case C_r does not entail C_d nor contradicts C_d.* If $C_{P_s} \wedge C_r$ is consistent, an active process is created from P_s, for which we can show $(*)$ in a similar way to case 2.

Next, consider the newly added suspended process that is created from P_s s.t. $Q@S \in AD_{P_s}$. We have the following three cases.

1. *Case C_r entails C_d.* If $C_{P_s} \wedge C_r$ is consistent, the suspended process $P'_s = \langle SG_{P_s}, \leftarrow C_{P_s} \wedge C_r \| GS_{P_s}, AD_{P_s} \rangle$ is created, and we have $\mathcal{R}^{(k+1)}_{P'_s} = \mathcal{R}^{(k)}_{P_s} \cup \{Q@S \leftarrow C_r \|\} \setminus \{Q@S \leftarrow C_d \|\}$. By the induction hypothesis, P_s satisfies $(*)$ for some C'_{P_s}; that is, there is a sequence of reductions "$\leftarrow \| GS_{init}$", ..., "$\leftarrow C_1 \|\{Q@S\} \cup GS$", "$\leftarrow C_1 \wedge C_d \| GS$", ..., "$\leftarrow C_1 \wedge C_d \wedge C_2 \|\{SG_{P_s}\} \cup GS_{P_s}$" w.r.t. $\mathcal{P}$ and $\mathcal{R}^{(k)}_{P_s}$ s.t. $\pi_V(C_{P_s})$ entails $\pi_V(C_1 \wedge C_d \wedge C_2)$, where C_1 and C_2 are the constraints obtained before and after processing $Q@S$ respectively, and $C_1 \wedge C_d \wedge C_2 = C'_{P_s}$. Then we can consider the sequence of reductions "$\leftarrow \| GS_{init}$", ..., "$\leftarrow C_1 \|\{Q@S\} \cup GS$", "$\leftarrow C_1 \wedge C_r \| GS$", ..., "$\leftarrow C_1 \wedge C_r \wedge C_2 \|\{SG_{P_s}\} \cup GS_{P_s}$" w.r.t. $\mathcal{P}$ and $\mathcal{R}^{(k+1)}_{P'_s}$. Since C_r entails C_d, and also since $\pi_V(C_{P_s})$ entails $\pi_V(C_1 \wedge C_d \wedge C_2)$, $\pi_V(C_{P_s} \wedge C_r)$ entails $\pi_V(C_1 \wedge C_r \wedge C_2)$. Thus $(*)$ holds for this new process.
2. *Case C_r contradicts C_d.* No such suspended process is created from P_s.
3. *Case C_r does not entail C_d nor contradicts C_d.* If $C_{P_s} \wedge C_r$ is consistent, a suspended process is created from P_s, for which we can show $(*)$ in a similar way to case 1.

The property $(*)$ is kept satisfied for the other processes that are not handled in these cases, since $Q@S$ has not been used in the reduction of those processes.

Therefore, $(*)$ holds for any processes after processing an answer in the fact arrival phase.

Since the property described in this theorem corresponds to the special case of the property $(*)$, where $GS_P = \emptyset$, this theorem holds. □

Hiroshi Hosobe received his doctoral degree from the University of Tokyo in 1998. After serving as a JSPS Postdoctoral Research Fellow at the University of Tokyo and as a Research Associate at the National Center for Science Information Systems, he joined the National Institute of Informatics as a Research Associate in 2000, where he has been working as an Associate Professor since 2004. In 2005, he spent two months as an Invited Professor in the LINA laboratory at the University of Nantes. His research interests include constraint programming, hybrid systems, user interfaces, information visualization, and interactive graphics. He was presented with the Takahashi Award in 2003 by the Japan Society for Software Science and Technology.

Ken Satoh was born in 1959. He joined Fujitsu Laboratories in 1981 and he received Doctor of Science from University Tokyo in 1993. He became an associate professor in Hokkaido University in 1995 and he has been a professor in National Institute of Informatics (NII) and Sokendai since 2001.

Philippe Codognet is working since September 2003 as Attache for Science and Technology at the French Embassy in Japan (Tokyo), on leave from University of Paris 6. After receiving a PhD in Computer Science from University of Bordeaux and working at Thomson-CSF (now Thales) Research Lab in Paris, he joined the French National Institute for Computer Science (INRIA) as senior researcher. After a sabbatical leave at Sony Computer Science Laboratory in Paris in 1997/8, he joined University Pierre et Marie Curie (Paris 6) in September 1998, as full professor. His researches focused on programming languages, artificial intelligence, logic, multi-agent systems and virtual reality. Over 70 publications in international journals and conferences are detailing his researches.