Speculative Computation by Abduction under Incomplete Communication Environments

Ken Satoh
Hokkaido University,
N13W8 Sapporo 060-8628 Japan
ksatoh@db-ei.eng.hokudai.ac.jp

Katsumi Inoue
Kobe University
Rokkodai, Nada, Kobe 657-8501 Japan
inoue@eedept.kobe-u.ac.jp

Koji Iwanuma
Yamanashi University
4-3-11 Takeda, Kofu 400-8511 Japan
iwanuma@esi.yamanashi.ac.jp

Chiaki Sakama
Wakayama University
Sakaedani, Wakayama 640-8510 Japan
sakama@sys.wakayama-u.ac.jp

Abstract

In this paper, we present a method of problem solving in multi-agent systems when communication between agents is not guaranteed. Under incomplete communication environments such as the Internet, the communication might fail and a reply might be significantly delayed. Therefore, research of problem solving under incomplete communication is very important.

To solve the problem, we propose a method using abduction. *Abduction is a way of reasoning where some hypothesis will be used to complement unknown information. The idea is as follows. When communication is delayed or failed, then we use a default hypothesis as a tentative answer and continue computation. When some response is obtained, we check consistency of the response and the current computation. If the response is consistent, then we continue the current computation; else if the response is inconsistent, we seek an alternative computation. This way of computation is called* speculative computation*, since computation using a tentative answer would lead to a significant advantage if it succeeds.*

In this paper, we restrict our attention to a master-slave multi-agent system and propose an implementation of speculative computation and show that abduction plays an important role in speculative computation.

1. Introduction

In this paper, we propose a method of *speculative computation* using abduction in multi-agent systems when communication is incomplete.

In most of the current research on multi-agent systems, people assume that communication of agents is guaranteed. Also, when an agent asks a question to other agents, a process is suspended until some response from other agents is obtained. Under incomplete communication environments such as the Internet, however, these assumptions are not guaranteed; messages between agents might be lost or significantly delayed. Therefore, research of problem solving under incomplete communication is very important.

Moreover, even if communication is guaranteed, we have a similar situation when an agent needs to communicate with a user and the user is absent or when the computation in other agents takes much time before sending an answer. These cases can also be viewed as if communication was lost, and again represent situations under incomplete communication.

As an example of incomplete communication, consider the following meeting room reservation.

- There are three persons A, B and C to attend the meeting.
- If a person is available, then he/she will attend the meeting.
- We ask a person whether he/she is free or not.
- If all the persons are available, we reserve a big room.

- If only two persons are available, we reserve a small room.

Suppose that we have answers from A and C that they are free but we do not have an answer from B, since the communication is lost or he/she is absent. If we follow the requirement that the communication must be completed before taking any further action, then, we cannot reserve a room until we get an answer from B. In ordinary life, however, if we know that B is normally free or expect that B is free, then we tentatively conclude that all the people are free and, therefore we reserve a big room. Eventually, if we know from B's reply that B is actually busy, then, we change the reservation of the big room into a small room. Moreover, there are other effects of speculative computation for reducing risks. Suppose that reservation of the above meeting rooms is very crowded, so that there are possibilities that we cannot reserve a room if the date is very near. Therefore, reserving a room in advance by estimating participants reduces a risk [1] .

In problem solving like the above, it is important to handle *default* information and *tentative* conclusion. In this paper, we propose a multi-agent system where we use defeasible information by extending abductive reasoning or default reasoning widely used in nonmonotonic reasoning in AI [Reiter80, Eshghi89, Kakas98]. The basic idea of this method is as follows.

1. The agent A prepares a default answer for a question in advance.
2. When an agent A asks a question to another agent B, the agent A uses the default value as a tentative answer and continues a computation along with the tentative answer.
3. When the response comes from the agent B,
    - if the response confirms the default answer, then the agent A continues the computation.
    - if the response is inconsistent with the default answer, the agent A withdraws the computation process with the default answer and restarts a computation with the true answer.

We assume that the default answer is prepared to cover the normal answer of each agent. This means that the response is usually consistent with the default answer and we, therefore, expect that the preceding computation is usually effective.

In this paper, we restrict our attention to a *master-slave multi-agent system* where the master uses speculative computation, and propose an implementation of a speculative framework using abduction. Then, we show an effect of speculative computation with an example.

The structure of the paper is as follows. In Section 2, we discuss relationship between speculative computation and abduction. In Section 3, we give a formal definition for speculative computation in terms of logic programming. In Section 4, we propose a method of speculative computation based on an abductive proof procedure, and in Section 5, we give an example of execution by the procedure. In Section 6, we mention related work and in Section 7, we summarize the contributions of this paper and discuss future research.

[1] Of course, in an actual setting, there is a tradeoff between the above risk and a penalty when we cancel the room.

2. Relationship between Speculative Computation and Abduction

We complement unknown information with a default and continue a computation speculatively. If the contrary to the default is found, then an alternative computation is considered. Given a program $\mathcal{P}$, a goal G, and a set Δ of atoms representing *default hypotheses*, *speculative computation of the goal* G is defined as

- $\mathcal{P} \cup \mathcal{H} \models G$,
- $\mathcal{P} \cup \mathcal{H}$ is consistent,
- $\mathcal{H} \subseteq \Delta$.

In the above, hypotheses in $\mathcal{H}$ supplement missing information in $\mathcal{P}$ to derive G.

The above situation is characterized using *abduction*. Abduction computes a set of hypotheses which accounts for a given observation in a program. In the above speculative computation, regarding G as an observation, the set $\mathcal{H}$ is considered an *explanation* of G in the program $\mathcal{P}$.

Thus, speculative computation is characterized by abduction. Comparing the two, in speculative computation, hypotheses $\mathcal{H}$ might be replaced by the "real" information which are known during the computation of G. In this case, the answer of the goal might change according to the real value. By contrast, in abduction, the observation G is initially given and the set Δ of hypotheses is, once assumed, never revised during computation of explanations.

In spite of such differences, we later show that we can utilize an abductive proof procedure in logic programming to realize speculative computation.

3. Framework of Speculative Computation in Master-Slave System

Definition 1 Let A be an atom. We sometimes call A a *positive literal* as well. We call an expression $\sim A$

a *negative literal* where $\sim$ expresses negation as failure. Let Q be a (positive/negative) literal. We define $\sim\sim Q = Q$.

Definition 2 A *speculative framework for a master-slave system* SF_{MS} is a quadruple $\langle \Sigma, \mathcal{E}, \Delta, \mathcal{P} \rangle$ where

- Σ is a finite set of constants. We call an element in Σ a *slave agent identifier.*
- $\mathcal{E}$ is a set of predicates called *external predicates.* When Q is a literal with an external predicate and S is a slave agent identifier or a variable ranging over Σ, we call $Q@S$ an *askable literal.* We define $\sim (Q@S)$ as $(\sim Q)@S$.
- Δ is a set of ground askable literals satisfying the following condition:

  Δ contains either $A@S$ or $\sim A@S$, but not both, for every ground atom A using an external predicate and every slave agent identifier S[2] .

  We call Δ a *default answer set.* Each element of Δ is used for a default answer for a corresponding askable literal.
- $\mathcal{P}$ is a normal logic program, that is, a set of rules of the form:

  $$H \leftarrow B_1, B_2, ..., B_n.$$

  where

  - H is an atom with a non-external predicate, and
  - each of $B_1, ..., B_n$ is either a (positive / negative) literal or an askable literal.

  Let R be a rule of the form $H \leftarrow B_1, B_2, ..., B_n$. We call H a head denoted as $head(R)$ and $B_1, ..., B_n$ a body denoted as $body(R)$. $head(R)$ is non-empty, but if there is no literal in the body, $body(R) = \emptyset$. We denote the set of all the ground instances of rules in $\mathcal{P}$ as $ground(\mathcal{P})$.

An askable literal $Q@S$ is a kind of syntax sugar so that we augment arguments of Q by S. That is, if $Q = p(t_1, ..., t_n)$ $(\sim p(t_1, ..., t_n))$ with an external predicate p and terms t_1,...,t_n, $Q@S$ expresses a formula $p(t_1, ..., t_n, S)$ $(\sim p(t_1, ..., t_n, S))$. We use this special notation, since we would like to emphasize S as a special argument expressing a slave agent identifier.

Intuitively, an askable literal has two kinds of meaning.

1. An askable literal $Q@S$ in a rule in $\mathcal{P}$ represents a question a master agent asks to a slave agent S.
2. An askable literal in Δ represents the default truth value; if $A@S \in \Delta$, A is normally true for a question to an agent S, and if $\sim A@S \in \Delta$, A is normally false for a question to an agent S.

Example 1 The example of meeting room reservation in Introduction is represented as the following speculative framework $SF_{MS} = \langle \Sigma, \mathcal{E}, \Delta, \mathcal{P} \rangle$

- $\Sigma = \{a, b, c\}$
- $\mathcal{E} = \{free\}$
- $\Delta = \{free@a, free@b, free@c\}$
- $\mathcal{P}$ is the following set of rules[3] :
  $meeting([a, b]) \leftarrow$
  $\quad available(a), available(b), \sim available(c).$
  $meeting([b, c]) \leftarrow$
  $\quad \sim available(a), available(b), available(c).$
  $meeting([c, a]) \leftarrow$
  $\quad available(c), \sim available(b), available(a).$
  $meeting([a, b, c]) \leftarrow$
  $\quad available(a), available(b), available(c).$
  $available(P) \leftarrow free@P.$
  $plan(small\_room, [a, b]) \leftarrow meeting([a, b]).$
  $plan(small\_room, [b, c]) \leftarrow meeting([b, c]).$
  $plan(small\_room, [c, a]) \leftarrow meeting([c, a]).$
  $plan(large\_room, [a, b, c]) \leftarrow meeting([a, b, c]).$

4. An Implementation of Speculative Computation

In this section, we show an implementation of speculative computation. The execution of speculative framework is based on two phases, a *process reduction phase* and a *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.

A process consists of the following objects; (a) the current status of computation, (b) already used defaults in the process and (c) answer substitutions of the initial query. Each process represents an alternative way of computation. Intuitively, processes are created when a choice point of computation is encountered such as case splitting. A process ends successfully if all the computation is done and the used defaults have

[2] Note that we can have different default values for different agents. For example, $A@S_1$ and $(\sim A)@S_2$ are possible for S_1 and S_2.

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

not been contradictory with the current returned answers. A process fails when some used default is found to contradict the newly returned answer.

In the process reduction phase, we reduce an active process set (see Definition 6 for the formal definition) into a new process set. Reduction for a positive/negative literal corresponds with treatment of a positive/negative literal in *abductive derivation* of [Eshghi89]. Reduction for a literal of the form $fail(...)$ corresponds with *consistency derivation* of [Eshghi89].

What differs from derivation of [Eshghi89] is treatment of askable literals as follows. During the reduction, for a process using a default, the default is assumed unless its complement has already been assumed or found to be true, whereas the process itself will be killed when the complement of the default has been found to be true. Simultaneously, if the real value of the default is not decided yet, the default will be asked to a slave agent. On the other hand, a process using the complement of a default will be suspended, since the complement is an exceptional situation in which the preceding computation will probably fail. Therefore, we must check the confirmation of the complement of the default by waiting for an answer from a slave agent.

When an answer to confirm a default comes from a slave agent, we remove suspended processes waiting for the complement of the default. When an answer to deny a default comes from a slave agent, we remove processes using the default and resume suspended processes waiting for the complement of the default. The last mechanism corresponds with consideration of alternative computation when the default contradicts a returned answer.

4.1. Preliminary Definitions

We define the following concepts for process reduction.

Definition 3 An *extended literal* is one of the following expression.

- a positive literal.
- a negative literal.
- an askable literal.
- $fail(L_1, ..., L_n)$ where L_i is a literal (positive, negative or askable).

$fail(L_1, ..., L_n)$ is used to show that there is no proof for one of L_i.

Definition 4 An *active process* is a triple $\langle GS, AD, ANS \rangle$ which consists of

- GS: A set of extended literals to be proved called a *goal set.*
- AD: A set of askable literals assumed already called *assumed defaults.*
- ANS: A set of answer substitutions of variables in initial goals.

Definition 5 A *suspended process* is a quadruple $\langle SG, GS, AD, ANS \rangle$ which consists of

- SG: An askable literal waiting for an answer from a slave agent in GS called a *suspended literal.*
- GS: A goal set
- AD: Assumed defaults
- ANS: A set of answer substitutions of variables in initial goals.

We use the following four sets for process reduction.

Definition 6

- *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* is a set of askable literals.
- *Returned facts RF* is a set of askable literals.

AAQ is used to avoid from asking redundant questions to slave agents, and RF is a set of real information returned from slave agents about askable literals.

4.2. Process Reduction Phase

In the following reduction, we specify changed APS, SPS, AAQ, RF as $NewAPS$, $NewSPS$, $NewAAQ$, $NewRF$. Otherwise each APS, SPS, AAQ ,RF is unchanged.

Initial Step Let GS be an initial goal set. We give $\langle GS, \emptyset, \emptyset \rangle$ to a proof procedure. That is, $APS = \{\langle GS, \emptyset, \emptyset \rangle\}$. And let $SPS = AAQ = RF = \emptyset$.
We iterate **Step 1**, **Step 2** and **Step 3**.

Step 1 If there is an active process $\langle \emptyset, AD, ANS \rangle$ appears in APS, then output assumptions $AD - RF$ and substitutions for variables ANS and halt.

Step 2 Otherwise, select an active process $\langle GS, AD, ANS \rangle$ from APS and select an extended literal L in GS. $APS' = APS - \{\langle GS, AD, ANS \rangle\}$ and $GS' = GS - \{L\}$.

Step 3 For the selected extended literal L, do the following.

- If L is a positive literal, then
  /* *processing usual resolution* */
  $NewAPS = APS' \cup$
  $\{\langle(\{body(R)\} \cup GS')\theta, AD, ANS\theta\rangle |$
  there exists $R \in \mathcal{P}$
  and a most general unifier(mgu) θ
  s.t. $head(R)\theta = L\theta\}$

- If L is a ground negative literal, then
  /* *processing negation as failure* */
  $NewAPS = APS' \cup \{\langle NewGS, AD, ANS\rangle\}$
  where $NewGS =$
  $\{fail(body(R\theta))|$
  there exists $R \in \mathcal{P}$ and an mgu θ
  s.t. $head(R)\theta = L\theta\} \cup GS'$

- If L is $fail(BS)$, then
  /* *showing a literal B in BS is not derived* */
  - if $BS = \emptyset$, $NewAPS = APS'$.
  - if $BS \neq \emptyset$ then
    - Select B from BS and $BS' = BS - \{B\}$.
    - if B is a positive literal,
      $NewAPS =$
      $APS' \cup \{\langle NewGS \cup GS', AD, ANS\rangle\}$
      where $NewGS =$
      $\{fail((\{body(R)\} \cup BS')\theta)|$
      there exists $R \in \mathcal{P}$ and an mgu θ
      s.t. $head(R)\theta = B\theta\}$
    - if B is a ground negative literal or a ground askable literal,
      $NewAPS = APS' \cup$
      $\{\langle\{\sim B\} \cup GS', AD, ANS\rangle\} \cup$
      $\{\langle\{fail(BS')\} \cup GS', AD, ANS\rangle\}$

- If L is a ground askable literal, $Q@S$, then
  - if $L \notin AAQ$ and $\sim L \notin AAQ$, then
    send a question Q to a slave agent S and $NewAAQ = AAQ \cup \{L\}$.
    /* *if L or $\sim L$ is not asked, ask Q to S* */
  - if $L \in AD$ or $L \in RF$ then
    $NewAPS = APS' \cup \{\langle GS', AD, ANS\rangle\}$
    /* *if L is already assumed or L is found to be true, continue* */
  - else if $\sim L \in AD$ or $\sim L \in RF$ then
    $NewAPS = APS'$,
    /* *else if $\sim L$ is already assumed or L is found to be false, kill this process* */
  - else if $L \in \Delta$ then
    $NewAPS =$
    $APS' \cup \{\langle GS', AD \cup \{L\}, ANS\rangle\}$,
    /* *else if truth value of L is not decided yet and the default value of L is "yes", then continue computation speculatively* */
  - else if $\sim L \in \Delta$, then
    $NewAPS = APS'$ and
    $NewSPS = SPS \cup \{\langle L, GS', AD, ANS\rangle\}$
    /* *else if truth value of L is not decided yet and the default value of L is "no", then suspend this process* */

In **Step 1**, when GS is finished, we output $AD - RF$ as assumptions in stead of AD. This is because an askable literal in RF represents real value of the literal, and, therefore, $AD - RF$ represents assumptions which are not confirmed yet.

In **Step 3**, if a selected literal is $Q@S$, then the case of $Q@S \in AD$ and $\sim Q@S \in RF$ cannot occur. This is because the following fact arrival phase always avoids this case by removing processes with $Q@S \in AD$ when $\sim Q$ arrives from a slave agent S.

4.3. Fact Arrival Phase

Suppose Q comes from a slave agent S. Then, do the following after one step of process reduction is finished.

- $NewRF = RF \cup \{Q@S\}$
  /* *Store the real truth value for $Q@S$* */

- If $Q@S \in \Delta$ then
  $NewSPS = SPS -$
  $\{\langle SG, GS, AD, ANS\rangle \in SPS | SG = \sim Q@S\}$
  /* *if a returned answer confirms a default value, continue computation and remove processes waiting for $\sim Q@S$.* */

- If $\sim Q@S \in \Delta$ then
  $NewAPS = APS \cup ResumedPS -$
  $\{\langle GS, AD, ANS\rangle \in APS | \sim Q@S \in AD\}$,
  and
  $NewSPS = SPS - ResumedPS -$
  $\{\langle SG, GS, AD, ANS\rangle \in SPS | \sim Q@S \in AD\}$
  where $ResumedPS =$
  $\{\langle GS, AD, ANS\rangle |$
  $\langle Q@S, GS, AD, ANS\rangle \in SPS\}$.
  /* *if a returned answer contradicts a default value, invoke processes waiting for the answer and remove processes using the default* */

4.4. Correctness of the Proof Procedure

In this subsection, we show correctness of the above procedure. We assume familiarity of the notions of stratified programs and the perfect model semantics introduced in [Przymusinski88].

Definition 7 Let $\mathcal{P}$ be a stratified logic program [4]. Let RF be a set of added facts. We define $\mathcal{P}^{RF}$ as follows.

$$\mathcal{P}^{RF} = \{head(R) \leftarrow (body(R) - RF) | R \in ground(\mathcal{P}) \text{ s.t. for every } Q@S \in RF, \sim Q@S \notin body(R)\}$$

$\mathcal{P}^{RF}$ intuitively means a reduced ground program reflecting the additional facts.

Definition 8 Let OD be a set of askable literals. We define $\mathcal{F}(OD)$ as follows.

$$\mathcal{F}(OD) = \{Q@S \leftarrow | Q \text{ is a positive literal and } Q@S \in OD\}$$

Theorem 1 *Let $SF_{MS} = \langle \Sigma, \mathcal{E}, \Delta, \mathcal{P} \rangle$ be a speculative framework where $\mathcal{P}$ is a stratified logic program. Let GS be an initial goal set. Suppose that $\langle \emptyset, AD, \theta \rangle$ is found and RF is the current set of returned facts. Let $OD = AD - RF$ be output assumptions. Then, for the perfect model M for $\mathcal{P}^{RF} \cup \mathcal{F}(OD)$, $M \models GS\theta$.*

Sketch of Proof: Since $\mathcal{P}$ is a stratified logic program, so is $\mathcal{P}^{RF} \cup \mathcal{F}(OD)$. Then, we can construct an abductive derivation [Eshghi89] from process reduction steps from $\langle GS, \emptyset, \emptyset \rangle$ to $\langle \emptyset, AD, \theta \rangle$. This derivation is correct for stratified logic programs by the result of [Eshghi89]. This means $M \models GS\theta$. □

5. Example

We use the program in Example 1 and take the following strategy for process reduction.

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

The following is an execution trace for $plan(L, R)$. We assume that answers from the agents a and b come at Step 25 and Step 28 respectively. We show APS, SPS, AAQ, RF only when a change is occurred. We abbreviate *small_room* as *sr*, *large_room* as *lr* *meeting* as *mt*, *available* as *av* and *free* as *fr*.

[4] A stratified program containing askable literals is defined in the same manner as the usual one.

1. $APS = \{\langle\{plan(L,R)\}, \emptyset, \emptyset\rangle\}$,
   $SPS = \emptyset$, $AAQ = \emptyset$, $RF = \emptyset$
2. $APS = \{$
   $\langle\{mt([a,b])\}, \emptyset, \{R{=}sr, L{=}[a,b]\}\rangle$,
   $\langle\{mt([b,c])\}, \emptyset, \{R{=}sr, L{=}[b,c]\}\rangle$,
   $\langle\{mt([c,a])\}, \emptyset, \{R{=}sr, L{=}[c,a]\}\rangle$,
   $\langle\{mt([a,b,c])\}, \emptyset, \{R{=}lr, L{=}[a,b,c]\}\rangle\}$
3. $APS = \{$
   $\langle\{av(a), av(b), \sim av(c)\}, \emptyset, \{R{=}sr, L{=}[a,b]\}\rangle$,
   AP_1, AP_2, AP_3 [5] $\}$
4. $APS = \{$
   $\langle\{fr@a, av(b), \sim av(c)\}, \emptyset, \{R{=}sr, L{=}[a,b]\}\rangle$,
   $AP_1, AP_2, AP_3\}$
5. $APS = \{$
   $\langle\{av(b), \sim av(c)\}, \{fr@a\}, \{R{=}sr, L{=}[a,b]\}\rangle$,
   $AP_1, AP_2, AP_3\}$,
   $AAQ = \{fr@a\}$
   fr is asked to the slave agent a.
6. $APS = \{$
   $\langle\{fr@b, \sim av(c)\}, \{fr@a\}, \{R{=}sr, L{=}[a,b]\}\rangle$,
   $AP_1, AP_2, AP_3\}$
7. $APS = \{$
   $\langle\{\sim av(c)\}, \{fr@a, fr@b\}, \{R{=}sr, L{=}[a,b]\}\rangle$,
   $AP_1, AP_2, AP_3\}$,
   $AAQ = \{fr@a, fr@b\}$
   fr is asked to the slave agent b.
8. $APS = \{$
   $\langle\{fail(fr@c)\}, \{fr@a, fr@b\},$
   $\{R{=}sr, L{=}[a,b]\}\rangle$,
   $AP_1, AP_2, AP_3\}$
9. $APS = \{$
   $\langle\{\sim fr@c\}, \{fr@a, fr@b\}, \{R{=}sr, L{=}[a,b]\}\rangle$,
   $AP_1, AP_2, AP_3\}$
10. $APS = \{AP_1, AP_2, AP_3\}$,
    $SPS =$
    $\{\langle \sim fr@c, \emptyset, \{fr@a, fr@b\},$
    $\{R{=}sr, L{=}[a,b]\}\rangle\}$,
    $AAQ = \{fr@a, fr@b, \sim fr@c\}$
    $\sim fr$ is asked to the slave agent c.
11. $APS = \{$
    $\langle\{\sim av(a), av(b), av(c)\}, \emptyset, \{R{=}sr, L{=}[b,c]\}\rangle$,
    $AP_2, AP_3\}$
12. $APS = \{$
    $\langle\{fail(fr@a), av(b), av(c)\}, \emptyset,$
    $\{R{=}sr, L{=}[b,c]\}\rangle$,
    $AP_2, AP_3\}$
13. $APS = \{$
    $\langle\{\sim fr@a, av(b), av(c)\}, \emptyset, \{R{=}sr, L{=}[b,c]\}\rangle$,
    $AP_2, AP_3\}$
14. $APS = \{AP_2, AP_3\}$,

[5] From now on, we abbreviate $\langle\{mt([b,c])\}, \emptyset, \{R = sr, L = [b,c]\}\rangle$, as AP_1, $\langle\{mt([c,a])\}, \emptyset, \{R = sr, L = [c,a]\}\rangle$ as AP_2, and $\langle\{mt([a,b,c])\}, \emptyset, \{R = lr, L = [a,b,c]\}\rangle$ as AP_3.

$SPS = \{$
$\langle \sim fr@c, \emptyset, \{fr@a, fr@b\}, \{R{=}sr, L{=}[a,b]\}\rangle,$
$\langle \sim fr@a, \{av(b), av(c)\}, \emptyset, \{R{=}sr, L{=}[b,c]\}\rangle\}$

15. $APS = \{$
$\langle \{av(a), {\sim}av(b), av(c)\}, \emptyset, \{R{=}sr, L{=}[c,a]\}\rangle,$
$AP_3\}$
16. $APS = \{$
$\langle \{fr@a, {\sim}av(b), av(c)\}, \emptyset, \{R{=}sr, L{=}[c,a]\}\rangle,$
$AP_3\}$
17. $APS = \{$
$\langle \{{\sim}av(b), av(c)\}, \{fr@a\}, \{R{=}sr, L{=}[c,a]\}\rangle,$
$AP_3\}$
18. $APS = \{$
$\langle \{fail(fr@b), av(c)\}, \{fr@a\},$
$\{R{=}sr, L{=}[c,a]\}\rangle,$
$AP_3\}$
19. $APS = \{$
$\langle \{\sim fr@b, av(c)\}, \{fr@a\}, \{R{=}sr, L{=}[c,a]\}\rangle,$
$AP_3\}$
20. $APS = \{AP_3\},$
$SPS = \{$
$\langle \sim fr@c, \emptyset, \{fr@a, fr@b\}, \{R{=}sr, L{=}[a,b]\}\rangle,$
$\langle \sim fr@a, \{av(b), av(c)\}, \emptyset, \{R{=}sr, L{=}[b,c]\}\rangle,$
$\langle \sim fr@b, \{av(c)\}, \{fr@a\},$
$\{R{=}sr, L{=}[c,a]\}\rangle\}$
21. $APS = \{$
$\langle \{av(a), av(b), av(c)\}, \emptyset, \{R{=}lr, L{=}[a,b,c]\}\rangle\}$
22. $APS = \{$
$\langle \{fr@a, av(b), av(c)\}, \emptyset, \{R{=}lr, L{=}[a,b,c]\}\rangle\}$
23. $APS = \{$
$\langle \{av(b), av(c)\}, \{fr@a\}, \{R{=}lr, L{=}[a,b,c]\}\rangle\}$
24. $APS = \{$
$\langle \{fr@b, av(c)\}, \{fr@a\}, \{R{=}lr, L{=}[a,b,c]\}\rangle\}$
25. **fr is returned from a!!**
$SPS = \{$
$\langle \sim fr@c, \emptyset, \{fr@a, fr@b\}, \{R{=}sr, L{=}[a,b]\}\rangle,$
$\langle \sim fr@b, \{av(c)\}, \{fr@a\},$
$\{R{=}sr, L{=}[c,a]\}\rangle\},$
$RF = \{fr@a\}$
26. $APS = \{$
$\langle \{av(c)\}, \{fr@a, fr@b\}, \{R{=}lr, L{=}[a,b,c]\}\rangle\}$
27. $APS = \{$
$\langle \{fr@c\}, \{fr@a, fr@b\}, \{R{=}lr, L{=}[a,b,c]\}\rangle\}$
28. **$\sim fr$ is returned from b!!**
$APS = \{$
$\langle \{av(c)\}, \{fr@a\}, \{R{=}sr, L{=}[c,a]\}\rangle\},$
$SPS = \emptyset,\ RF = \{fr@a, \sim fr@b\}$
29. $APS = \{$
$\langle \{fr@c\}, \{fr@a\}, \{R{=}sr, L{=}[c,a]\}\rangle\}$
30. $APS = \{$
$\langle \emptyset, \{fr@c, fr@a\}, \{R{=}sr, L{=}[c,a]\}\rangle\},$
$\{R{=}sr, L{=}[c,a]\}$ and
$\{fr@c, fr@a\} - RF = \{fr@c\}$ is returned.

At Step 5, a default $fr@a$ is assumed and computation of this process continues and at the same time, the real value of fr for a is asked to the slave agent a. 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 10, the complement of a default, $\sim fr@c$ is checked and computation of this process is suspended and at the same time, the real value of fr for c is asked to the slave agent c. We suspend this process, since the process uses the complement of a default which is normally false and therefore, the probability of succeeding this computation is low.

At Step 25, the answer from a is returned. Since this answer confirms the default, we remove processes waiting for the complement of the default and continue the reduction.

At Step 28, the answer from b is returned. Since this answer denies the default, we remove processes assuming the default and resume processes waiting for the complement of the default.

At Step 30, an answer of attending a and c and reserving $small\_room$ is obtained with the default assumption “c is free”.

6. Related Research

Speculative computation has originally been investigated in the context of parallel computing in order to speed computation [Burton85, Gregory93]. When some processors are idle on a parallel computer, these idle processors are used to execute computation which is sometimes useful and sometimes unnecessary. The most popular application of speculative computation is parallel search. In parallel search, some branches are traversed in parallel speculatively. When a solution is found in a branch, searching other branches are aborted.

There are some research applying nonmonotonic reasoning or abductive reasoning to multi-agent systems [Morgenstern90, Baral91, Poole97, Kowalski99].

[Morgenstern90] formalizes agent's reasoning of another agent's nonmonotonic reasoning so that an agent predicts another agent's default conclusions. [Baral91] gives semantics of combination of various agents' different knowledge so that consistent semantics is obtained even if inconsistency arises from a union of each agent's knowledge. [Poole97] adopts abductive logic programming with an uncertainty measure in his Independence Choice Logic for multi-agent systems. [Kowalski99] uses abduction to assimilate obtained information into agent's current knowledge. However, as far as we know, there are no research applying defeasible reasoning to speculative computation.

We use a default assumption to continue speculative computation. If we regard this default assumption as a current fact whose truth value can be changed as time goes, then our work can be regarded as a restricted form of planning in dynamic environments. In this viewpoint, research on such planning is very much related. In fact, [Hayashi99] gives a dynamic proof procedure in which a knowledge base can be changed along with observed facts in the context of logic programming. Although we can use the techniques of planning under dynamic environments, these research, however, are different from ours in motivation.

7. Conclusion

The contributions of this paper are summarized as follows.

- We presented speculative computation by abduction in multi-agent systems and showed that this can be used for incomplete communication environments.
- We proposed a proof procedure of speculative computation in a master-slave multi-agent systems and showed that it returns correct answers for the perfect model semantics in a stratified logic program.

The following issues remain for future research.

- We extend to other kind of multi-agent systems such as a system where every agent can communicate each other.
- We avoid recomputation in a process when there are some fragments of computation already done in an alternative process.
- We investigate when we let a process compute speculatively, since speculative computation might be sometimes expensive. For example, in the meeting room reservation example, cancelling the room might cause a penalty fee and so, there is trade-off between preceding reservation and its risk. Therefore, we need to estimate the trade-off for effective speculative computation.
- Since we might not always know a default value for a query, we need to incorporate a learning mechanism of a default value through observation of user's behavior.

Acknowledgments This research is partly supported by Grant-in-Aid for Scientific Research, "Construction of Logical Multi-agent Systems under Incomplete Communication Environments", The Ministry of Education, Japan. We also thank Ryusuke Masuoka from Fujitsu Laboratories and Prof. Yasuhiko Kitamura from Osaka City University for useful discussions.

References

[Baral91] Baral, C., Kraus, S. and Minker, J., "Combining Multiple Knowledge Bases", *IEEE Trans. Knowledge and Data Engineering*, vol. 3., pp. 208 – 220 (1991).

[Burton85] Burton, F. W., "Speculative Computation, Parallelism, and Functional Programming", *IEEE Transactions on Computers*, vol. c-34 pp. 1190 – 1193 (1985).

[Eshghi89] Eshghi, K., Kowalski, R. A., "Abduction Compared with Negation by Failure", *Proceedings of ICLP'89*, pp. 234 – 254 (1989).

[Gregory93] Gregory, S., "Experiment with Speculative Parallelism in Parlog", *Proceedings of ILPS'93*, pp. 370 – 387 (1993).

[Hayashi99] Hayashi, H., "Replanning in Robotics by Dynamic SLDNF", *Proceedings of IJCAI-99 WS on Scheduling and Planning Meet – Real-Time Monitoring in a Dynamic and Uncertain World* (1999).

[Kakas98] Kakas, A. C., Kowalski, R., Toni, F., "The Role of Abduction in Logic Programming", *Handbook of Logic in Artificial Intelligence and Logic Programming* 5, pp. 235 – 324 (1998).

[Kowalski99] Kowalski, R. A., Sadri, F., "From Logic Programming to Multi-Agent Systems", *Annals of Mathematics and Artificial Intelligence*, vol. 25, pp.391–419 (1999)

[Morgenstern90] Morgenstern, L., "A Formal Theory of Multiple Agent Nonmonotonic Reasoning", *Proceedings of AAAI'90*, pp. 538 – 544 (1990).

[Poole97] Poole, D., "The Independent Choice Logic for Modeling Multiple Agents under Uncertainty", *Artificial Intelligence*, **94**(1-2), pp. 7 – 56 (1997).

[Przymusinski88] Przymusinski, T., "On the Declarative Semantics of Stratified Deductive Databases and Logic Programs", *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, pp. 193 – 216 (1988).

[Reiter80] Reiter, R., "A Logic for Default Reasoning", *Artificial Intelligence*, **13**, pp. 81 – 132 (1980).