Speculative Constraint Processing for Hierarchical Agents

Hiroshi Hosobe [a,*], Ken Satoh [a], Jiefei Ma [b], Alessandra Russo [b], and Krysia Broda [b]
[a] *National Institute of Informatics, Japan*
E-mail: {hosobe,ksatoh}@nii.ac.jp
[b] *Imperial College London, United Kingdom*
E-mail: {jm103,ar3,kb}@doc.ic.ac.uk

Speculative computation is an effective means for solving problems with incomplete information in multi-agent systems. It allows such a system to compute tentative solutions by using default knowledge about agents even if communications between agents are delayed or fail. Previously we have proposed a logical framework for speculative constraint processing for master-slave multi-agent systems. In this paper, we extend the framework to support more general multi-agent systems that are hierarchically structured. We provide an operational model for the framework and present a prototype implementation of the model.

Keywords: multi-agent systems, speculative computation, logic programming, constraints

1. Introduction

Multi-agent systems typically rely on communications between agents. Most of multi-agent systems are designed to work well as long as there is no problem with communications between agents. In practice, however, it is often difficult to guarantee efficient and reliable communications between agents. If a multi-agent system is deployed on an unreliable network such as the Internet, or if a multi-agent system requires involvement of human users, communications might be largely delayed or even fail.

Speculative computation is an effective means for coping with such problems in multi-agent systems [2,5,6,11,15,16,17,18,19]. It allows a multi-agent system to compute tentative solutions if communications between agents are delayed or fail. This speculative computation is done by using *default* knowledge about other agents instead of waiting for their answers.

Previously we have proposed a logical framework for *speculative constraint processing* for multi-agent systems [2]. The framework allowed agents to communicate by means of *constraints* that are powerful in modeling problems. In addition, it supported the revision of previous answers that is useful for modeling complex problems involving, e.g., human users. However, the framework was limited to master-slave multi-agent systems.

In this paper, we extend our previous framework to support more general multi-agent systems that are hierarchically structured. Unlike the previous framework, the extended framework allows speculative computation agents to communicate with other speculative computation agents. Therefore, with this extension, we can model more complex problems as *hierarchical multi-agent systems*. For example, this extension allows us to model a multi-agent planning system where the root agent speculatively executes the entire planning task while other personal agents also speculatively perform the management of the corresponding human users. It should be noted that such a system with multiple speculative computation agents cannot be modeled as a master-slave multi-agent system.

Our framework can also be regarded as a model and a mechanism for distributed problem solving. Knowledge about a problem is hierarchically distributed over a set of agents; some agents might be specialized in particular tasks, and other agents might be human users. Since the whole problem cannot be solved by a single agent, these agents must cooperatively solve the problem by exchanging questions and answers represented as con-

*Corresponding author: Hiroshi Hosobe, National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan.

AI Communications
ISSN 0921-7126, IOS Press. All rights reserved

straints. Since our framework supports speculative computation, an agent that has sent a question to one of its child agents does not need to wait for the answer of the child agent; it speculatively continues its computation by using default knowledge about the child agent.

Our ultimate goal is to provide a powerful framework for speculative computation that realizes effective and efficient information processing in distributed multi-agent systems. For this purpose, we need to allow complex structures of agents as well as to enhance the power of individual agents. Although our previous framework for master-slave multi-agent systems achieved a powerful constraint-based mechanism for handling individual agents, its limitation on the structure of agents posed a major difficulty. Thus our new framework for hierarchical multi-agent systems marks an incremental but essential advance toward our ultimate goal.

Also, our technical contributions in this paper are multifold:

– we present a logical framework for hierarchical multi-agent systems by extending the formulation and semantics of our previous master-slave framework;
– we provide an operational model of hierarchical multi-agent systems by modifying our previous master-slave model;
– we formally prove the correctness of the operational model in the sense of both soundness and completeness;
– we present a prototype implementation of the operational model.

The rest of this paper is organized as follows. After describing related work in Section 2, we provide the formulation and semantics of our extended framework in Section 3. Next, we present the operational model for the extended framework in Section 4, and the prototype implementation of the model in Section 5. In Section 6, we show an example of executing a multi-agent system with our implementation. After discussing our work in Section 7, we describe conclusions and future work in Section 8.

2. Related Work

Speculative computation has been studied in several fields of computer science [1]. An example in the field of logic programming is the use of speculative parallelism in the Parlog language [3]. Its aim is to exploit the speculative parallelism to speed up the execution of parallel search algorithms such as the parallel A^*. Although our work was motivated by such previous work, we are particularly interested in the use of speculative computation for multi-agent systems.

Originally in [18], we proposed a logical framework for speculative computation for master-slave multi-agent systems, which we realized by exploiting abduction. Later we extended it to support more general multi-agent systems that are hierarchically structured [19]; this work also enabled agents to revise their answers (i.e. belief revision), which is caused by the speculative computation of other agents. We also proposed a framework for combining speculative computation and abduction [16]. Sakama et al. proposed an alternative logical multi-agent framework that translates a program by attaching time stamps to predicates [15]. Inoue and Iwanuma proposed a different approach to speculative computation that uses a consequence-finding procedure [6]. It should be noted that all these studies were restricted to yes/no questions.

To handle more general questions, we proposed a framework for speculative constraint processing (firstly in [17], and also in its journal version [5]). In the framework, constraints facilitated the modeling of more general problems. Later we extended the constraint-based framework to support the revision of answers [2]. However, both of these frameworks were limited to master-slave multi-agent systems.

Constraint programming languages such as AKL [8] and Oz [20] perform a kind of speculative computation. AKL allows local speculative variable bindings in a guard of each clause until one of guards succeeds, 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 AKL and Oz could be good platforms for the implementation of speculative computation using defaults.

There have been studies on the extension of logic programming to multi-agent systems (e.g. [10]). Unlike those studies, our work is focused on speculative computation for multi-agent systems.

Researchers have been studying agent-based frameworks for processing constraint satisfaction problems (e.g. [12,22]) and similar problems (e.g. [4,21]). Our framework is more computationally complex than these frameworks, since our framework allows programmers to dynamically generate such problems and also control speculative computation due to the power of constraint logic programming.

In the field of artificial intelligence, there has been much research on non-monotonic reasoning such as default logic [14], abductive logic programming [9], and multi-agent non-monotonic reasoning [13]. Unlike such research, our work is focused on the use of default knowledge to enable speculative computation in multi-agent systems.

3. Hierarchical Multi-Agent System

This section provides a formulation and a semantics of hierarchical multi-agent systems.

3.1. Formulation

We first formulate multi-agent systems. In this paper, we restrict our attention to a tree-structured composition of agents that we call an *agent hierarchy*.

Definition 1 (agent hierarchy)**.** An agent hierarchy H is a tree consisting of a set of nodes called agents. Let $\mathrm{root}(H)$ be the root node of H, called the *root agent*. Let $\mathrm{int}(H)$ be the set of all the non-leaf nodes of H, each called an *internal agent*. Let $\mathrm{ext}(H)$ be the set of all the leaf nodes of H, each called an *external agent*. Given an internal agent M, let $\mathrm{chi}(M,H)$ be the set of all the child nodes of M, each called a *child agent* of M. Given a non-root agent S, let $\mathrm{par}(S,H)$ be the parent node of S, called the *parent agent* of S.

By convention, we use either M or S (possibly with primes and subscripts) to indicate an agent. Also, when we refer to specific agents, we adopt r as the root agent, another small letter (e.g. a and b) as a non-root internal agent, and a small letter with a prime (e.g. a' and b') as an external agent.

Example 1. Let H be the tree consisting of nodes r, a, b, a', and b' and parent-to-child edges $r \to a$, $a \to a'$, $r \to b$, and $b \to b'$. Then H can be regarded as the agent hierarchy with agents r, a, b, a', and b', satisfying $\mathrm{root}(H) = r$, $\mathrm{int}(H) = \{r, a, b\}$, $\mathrm{ext}(H) = \{a', b'\}$, $\mathrm{chi}(r, H) = \{a, b\}$, $\mathrm{par}(a, H) = \mathrm{par}(b, H) = r$, $\mathrm{chi}(a, H) = \{a'\}$, $\mathrm{par}(a', H) = a$, $\mathrm{chi}(b, H) = \{b'\}$, and $\mathrm{par}(b', H) = b$.

We restrict internal agents to artificial agents that are based on a variant of *constraint logic programming* (CLP) [7]. Specifically, each internal agent is associated with a *specification* that is a constraint logic program with *default rules*. Rules in constraint logic programs as well as default rules consist of atoms and constraints. Atoms are categorized into *askable* and *non-askable atoms*; intuitively, an agent treats an askable atom as a question that should be posed to one of its children, and a non-askable atom as a piece of knowledge that should be acquired from its program.

Definition 2 (askable/non-askable atom)**.** Given an agent hierarchy H and an agent $M \in \mathrm{int}(H)$, an atom is either $p(X_1, X_2, \ldots, X_n)@S$ called an askable atom, or $p(t_1, t_2, \ldots, t_n)@M$ called a non-askable atom, where $S \in \mathrm{chi}(M,H)$, p is an n-ary predicate, each X_i is a variable, and each t_i is a term. Given an askable atom $Q@S$, the set of the variables that appear in Q is written $\mathrm{var}(Q)$.

Definition 3 (specification of an agent)**.** Given an agent hierarchy H and an agent $M \in \mathrm{int}(H)$, a specification F_M of M is a pair $\langle \Delta_M, \mathcal{P}_M \rangle$ with the following Δ_M and $\mathcal{P}_M$.

- Δ_M is a set of rules in the form

  $$Q@S \leftarrow C \,||$$

  each called a default rule w.r.t. $Q@S$, where S is a child agent of M, $Q@S$ is an askable atom, and C is a set of constraints.
- $\mathcal{P}_M$ is a constraint logic program that is a set of rules R in the form

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

  where:

- $*$ H is a non-askable atom called the *head* of R and written $\mathrm{head}(R)$;
- $*$ C is a set of constraints written $\mathrm{const}(R)$;
- $*$ each B_i is either an askable or non-askable atom, and the sequence $B_1, B_2, \ldots, B_n$ is called the *body* of R and written $\mathrm{body}(R)$.

A multi-agent system is an agent hierarchy, each of whose internal agents is associated with a specification.

Definition 4 (multi-agent system)**.** A multi-agent system is a pair $\langle H, \mathcal{F}\rangle$, where H is an agent hierarchy, and $\mathcal{F}$ is a set of specifications $F_M = \langle \Delta_M, \mathcal{P}_M\rangle$ of $M \in \mathrm{int}(H)$, i.e. $\mathcal{F} = \{\langle \Delta_M, \mathcal{P}_M\rangle\}_{M\in\mathrm{int}(H)}$.

We use the following room reservation problem as a running example.

Example 2. Let H be the agent hierarchy given in Example 1. Let $\mathcal{F} = \{F_r, F_a, F_b\}$ with the following specifications of r, a, and b.

- $F_r = \langle \Delta_r, \mathcal{P}_r\rangle$ where
  - $*$ Δ_r contains the following default rules:
    $\mathit{available}(D)@a \leftarrow D \in \{1,2,3\} \,||$
    $\mathit{available}(D)@b \leftarrow D \in \{1,2,3\} \,||$
  - $*$ $\mathcal{P}_r$ is the following constraint logic program:
    $\mathit{reserve}(R, L, D)@r \leftarrow$
    $\quad R = \mathit{twin\_room}, L = [a, b] \,||$
    $\quad \mathit{available}(D)@a, \mathit{available}(D)@b$
    $\mathit{reserve}(R, L, D)@r \leftarrow$
    $\quad R = \mathit{single\_room}, L = [a] \,||$
    $\quad \mathit{available}(D)@a, \mathit{unavailable}(D)@b$
    $\mathit{reserve}(R, L, D)@r \leftarrow$
    $\quad R = \mathit{single\_room}, L = [b] \,||$
    $\quad \mathit{unavailable}(D)@a, \mathit{available}(D)@b$
- $F_a = \langle \Delta_a, \mathcal{P}_a\rangle$ where
  - $*$ Δ_a contains the following default rules:
    $\mathit{free}(D)@a' \leftarrow D \in \{1,2\} \,||$
    $\mathit{busy}(D)@a' \leftarrow D \in \{3\} \,||$
  - $*$ $\mathcal{P}_a$ is the following constraint logic program:
    $\mathit{available}(D)@a \leftarrow \,||\, \mathit{free}(D)@a'$
    $\mathit{unavailable}(D)@a \leftarrow \,||\, \mathit{busy}(D)@a'$
- $F_b = \langle \Delta_b, \mathcal{P}_b\rangle$ where
  - $*$ Δ_b contains the following default rules:
    $\mathit{free}(D)@b' \leftarrow D \in \{2\} \,||$
  - $*$ $\mathcal{P}_b$ is the following constraint logic program:
    $\mathit{available}(D)@b \leftarrow \,||\, \mathit{free}(D)@b'$
    $\mathit{unavailable}(D)@b \leftarrow \,||\, \mathit{busy}(D)@b'$

Then $\langle H, \mathcal{F}\rangle$ is a multi-agent system.

In this example, there are two human users represented as external agents a' and b', whose availability is maintained by internal agents a and b respectively. Intuitively, the root agent r speculatively reserves a twin or single room for a' and/or b' by using the default rules and by asking a and b about the availability of a' and b'; then a and b speculatively compute the availability of a' and b' by using their more detailed default rules.

3.2. Semantics

Next, we present the semantics of hierarchical multi-agent systems. We define it by extending the semantics for our previous framework [2], which is based on the CLP scheme [7].

We first define a *goal* that is a question posed to a multi-agent system.

Definition 5 (goal)**.** Let $\langle H, \mathcal{F}\rangle$ be a multi-agent system. A goal G is "$\leftarrow C \,||\, B_1, B_2, \ldots, B_n$", where C is a set of constraints called the constraints of G, and each B_i is either an askable or non-askable atom. The sequence $B_1, B_2, \ldots, B_n$ is called the body of G.

The *belief state* of a multi-agent system gives the set of answers and default rules about external agents that should be used to obtain theoretical solutions to the entire system.

Definition 6 (belief state)**.** Let $\langle H, \mathcal{F}\rangle$ be a multi-agent system with $\mathcal{F} = \{\langle \Delta_M, \mathcal{P}_M\rangle\}_{M\in\mathrm{int}(H)}$. Let $\mathcal{A}_H$ be a set of most recent answers of the external agents, each of which is a rule "$Q@S \leftarrow C||$" with $S \in \mathrm{ext}(H)$. The belief state of $\langle H, \mathcal{F}\rangle$ w.r.t. $\mathcal{A}_H$, written $\mathrm{bel}(\mathcal{A}_H, \langle H, \mathcal{F}\rangle)$, is

$$\mathrm{bel}(\mathcal{A}_H, \langle H, \mathcal{F}\rangle) = \mathcal{A}_H \cup \{\text{“}Q@S \leftarrow C'||\text{”} \mid S \in \mathrm{ext}(H) \wedge \text{“}Q@S \leftarrow C'||\text{”} \in \Delta_{\mathrm{par}(S,H)} \wedge \neg\,\exists C, \text{“}Q@S \leftarrow C||\text{”} \in \mathcal{A}_H\}.$$

As in the ordinary CLP scheme, a solution to the entire multi-agent system is obtained by a *derivation* of a goal that is a sequence of *reductions*.

Definition 7 (reduction)**.** Let $\langle H, \mathcal{F}\rangle$ be a multi-agent system with $\mathcal{F} = \{\langle \Delta_M, \mathcal{P}_M\rangle\}_{M\in\mathrm{int}(H)}$, and $\mathcal{A}_H$ be a set of most recent answers of the external agents. A reduction of a goal "$\leftarrow C \,||\, B_1, B_2, \ldots, B_n$" w.r.t. $\langle H, \mathcal{F}\rangle$, $\mathcal{A}_H$, and B_i is a goal "$\leftarrow C' \,||\, GS$" such that:

– there exists a rule R in $(\bigcup_{M\in\text{int}(H)} \mathcal{P}_M) \cup \text{bel}(\mathcal{A}_H, \langle H, \mathcal{F}\rangle)$ such that $C \wedge (B_i = \text{head}(R)) \wedge \text{const}(R)$ is consistent;[1]
– $C' = C \wedge (B_i = \text{head}(R)) \wedge \text{const}(R)$;
– $GS = B_1, \ldots, B_{i-1}, \text{body}(R), B_{i+1}, \ldots, B_n$.

Definition 8 (derivation)**.** Let $\langle H, \mathcal{F}\rangle$ be a multi-agent system with $\mathcal{F} = \{\langle \Delta_M, \mathcal{P}_M\rangle\}_{M\in\text{int}(H)}$ and $M_{\text{root}} = \text{root}(H)$, and $\mathcal{A}_H$ be a set of most recent answers of the external agents. A derivation of a goal $G =$ "$\leftarrow || \, Q_{\text{init}}@M_{\text{root}}$" w.r.t. $\langle H, \mathcal{F}\rangle$ and $\mathcal{A}_H$ is a sequence of reductions "$\leftarrow || \, Q_{\text{init}}@M_{\text{root}}$", …, "$\leftarrow C||$" w.r.t. $\langle H, \mathcal{F}\rangle$ and $\mathcal{A}_H$, where an atom in the body of the current goal is selected in each reduction. C is called an answer w.r.t. $\langle H, \mathcal{F}\rangle$, $\mathcal{A}_H$, and G.

4. Operational Model

This section provides the operational model of hierarchical multi-agent systems defined in the previous section. After an overview of the operational model, we present its data structures, procedure, and correctness.

4.1. Overview

This operational model is an extension of the model that we previously constructed for master-slave multi-agent systems [2]. Since a hierarchical multi-agent system can be regarded as a hierarchy of master-slave multi-agent systems, the main task of the extension is to appropriately connect such master-slave multi-agent systems in a hierarchical manner. For this purpose, we made to the previous model a modification related to the treatment of returned answers and finished processes.

As in the previous model, the execution of an agent is based on two kinds of phases: *process reduction phases* and *fact arrival phases*. A process reduction phase is a normal execution of a program in an internal agent, and a fact arrival phase is an interruption phase that is invoked when an *answer* arrives from a child agent.

A computational state in an internal agent is represented as a *process*. Processes are created when a choice point of computation, such as case splitting, default handling, and answer arrival, is encountered. Figures 1–4 illustrate how processes are updated. In these trees, each node represents a process, but we only show constraints associated with the process. Each root node represents a constraint for the original process, and the other nodes represent the constraints added to the descendant processes. Note that we specify true for non-root nodes without added constraints, since the addition of the true constraint does not change the solutions to existing constraints. The leaves of the process trees represent the current processes. In other words, the processes that are not at the leaves have been deleted.

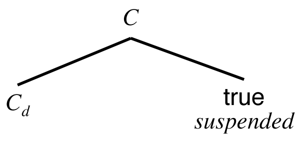

Fig. 1. Handling an askable atom $Q@S$ in a process reduction phase.

Figure 1 shows a situation where an agent treats an askable atom $Q@S$ whose answer has not yet arrived in a process reduction phase. In this case, the current process represented by C is split into two different kinds of processes: a process using the default C_d, called a *default process*,[2] and the current process C itself, called an *original process*, that is suspended at this point.

If there are multiple default rules for $Q@S$, we will have more than one default process, but still only one original process. The reason for suspending such a process (which is kept in memory) is that, in case of a contradictory revision of the default or a later arrival of an alternative answer, the intermediate result of the suspended process can be reused.

Figure 2 illustrates a situation where the agent receives a first answer to $Q@S$, expressed by the constraint C_f, after reductions of the default processes (represented by the dashed lines). Then the default and original processes are updated as follows:

– each default process is reduced to two different kinds of processes, i.e. a process including C_f, and the current process itself that is suspended at this point;

[1]If B_i is unifiable with head(R), $B_i =$ head(R) represents the conjunction of the constraints that equate the arguments of B_i with those of head(R); otherwise, $B_i =$ head(R) represents false.

[2]We assume for simplicity that there is only one default rule for $Q@S$.

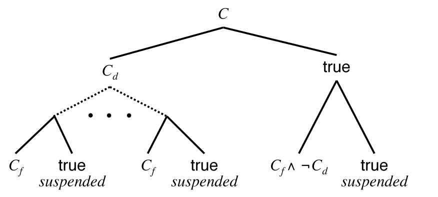

Fig. 2. Handling a first answer C_f for $Q@S$.

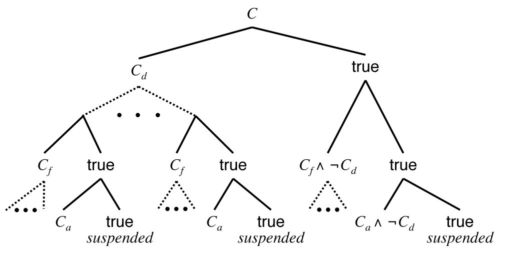

Fig. 3. Handling an alternative answer C_a for $Q@S$.

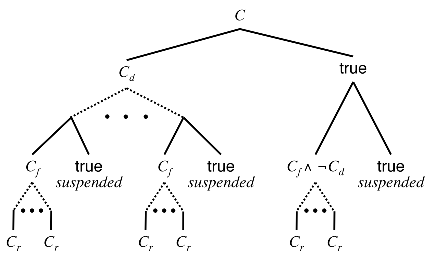

Fig. 4. Handling a revised answer C_r for $Q@S$.

– the original process is also reduced to two different kinds of processes, i.e. a process including $C_f \wedge \neg C_d$, and the original process suspended at this point.

Figure 3 depicts a situation where the agent receives an alternative answer to $Q@S$ whose constraint is C_a. We need to update processes basically in the same way as in handling the first answer C_f, without affecting the processes that treat C_f.

Figure 4 shows a situation where the agent receives an answer to $Q@S$ that revises the first answer C_f to C_r. In our operational model, a revised answer is always narrower than its previous answer, i.e. C_r entails C_f.[3] Therefore, we only need to update the processes that treat C_f.

[3]It should be noted that an answer that is not narrower than a previous answer can be represented as an alternative answer.

4.2. Data Structures

We now define necessary data structures for the operational model. A *process identifier* is an element of a countably infinite set $\{p_1, p_2, \ldots\}$. An *answer identifier* is an element of $\{o_\mathrm{s}, o_\mathrm{n}\} \cup \{d_1, d_2, \ldots\} \cup \{p_1, p_2, \ldots\}$, where $\{d_1, d_2, \ldots\}$ is a countably infinite set, and $\{o_\mathrm{s}, o_\mathrm{n}\}$, $\{d_1, d_2, \ldots\}$, and $\{p_1, p_2, \ldots\}$ are disjoint. A *labeled askable atom* is a pair $\langle Q@S, o_\mathrm{s}\rangle$, $\langle Q@S, o_\mathrm{n}\rangle$, $\langle Q@S, d_i\rangle$, or $\langle Q@S, p_i\rangle$, where $Q@S$ is an askable atom, and o_s, o_n, d_i, and p_i are answer identifiers.

An *answer* is a data structure sent by an agent to its parent agent to reply to a question.

Definition 9 (answer). Given a multi-agent system $\langle H, \mathcal{F}\rangle$ and an agent $M \in \mathrm{int}(H)$, an answer to M is a quadruple $\langle Q@S, AID, C, AID_\mathrm{prev}\rangle$, where $S \in \mathrm{chi}(M, H)$, $Q@S$ is an askable atom, AID is an answer identifier, C is a set of constraints, and AID_prev is either nil or a different answer identifier from AID. If $AID_\mathrm{prev} = \mathsf{nil}$, this answer is called a *new answer*; otherwise, it is called a *revised answer*. For any pair of answers $\langle Q@S, AID, C, AID_\mathrm{prev}\rangle$ and $\langle Q@S, AID_\mathrm{prev}, C_\mathrm{prev}, AID'\rangle$ (which is sometimes called the previous answer) with $AID_\mathrm{prev} \neq \mathsf{nil}$, C must entail C_prev.

A *process* is a data structure that holds a (possibly intermediate) result of computing an answer. A single agent maintains a set of processes to keep different ways of possible computation. In the operational model, processes are divided into two categories, i.e. *ordinary* and *finished processes*. Although they are always distinguished, they have a common structure defined below.

Definition 10 (process). Given a multi-agent system $\langle H, \mathcal{F}\rangle$ and an agent $M \in \mathrm{int}(H)$, a process P of M is a quintuple $\langle PID, C, GS, WA, AA\rangle$, where PID is a process identifier written $\mathrm{pid}(P)$, C is a set of constraints written $\mathrm{pconst}(P)$, GS is a set of atoms written $\mathrm{gs}(P)$, WA and AA are sets of labeled askable atoms written $\mathrm{wa}(P)$ and $\mathrm{aa}(P)$ respectively. If $\mathrm{wa}(P) = \emptyset$, P is said to be *active*, and otherwise *suspended*.

An *answer entry* is a data structure that keeps an answer and the identifiers of the processes using the answer.

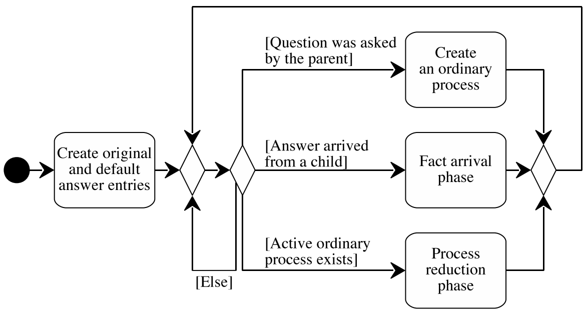
Create original and default answer entries
[Question was asked by the parent]
Create an ordinary process
[Answer arrived from a child]
Fact arrival phase
[Active ordinary process exists]
Process reduction phase
[Else]

Fig. 5. Workflow of an internal agent.

Definition 11 (answer entry)**.** Given a multi-agent system $\langle H, \mathcal{F}\rangle$ and an agent $M \in \mathrm{int}(H)$, an answer entry A for M is a quadruple $\langle Q@S, \mathit{AID}, C, \mathit{UPS}\rangle$, where $S \in \mathrm{chi}(M, H)$, $Q@S$ is an askable atom written $\mathrm{aq}(A)$, AID is an answer identifier written $\mathrm{aid}(A)$, C is a set of constraints written $\mathrm{aconst}(A)$, and UPS is a set of process identifiers written $\mathrm{ups}(A)$. If $\mathrm{aid}(A)$ is either o_{s} or o_{n}, A is called an original answer entry; if $\mathrm{aid}(A)$ is d_i, A is called a default answer entry; otherwise, A is called an ordinary answer entry.

An original answer entry is associated with either o_{s} or o_{n}. Intuitively, an entry with o_{s} records processes from which processes using default answer entries were speculatively created; by contrast, an entry with o_{n} keeps processes from which processes using ordinary answer entries were created.

4.3. Procedure

Now we present the procedure of the operational model. The main routine illustrated in Figure 5 and described in Figure 6 processes an internal agent M. It consists of two parts. The first part creates answer entries used for speculative computation. The second part is the main loop that performs one of the following three operations. The first operation creates a process for a newly asked question. The second and third operations invoke a *fact arrival phase* and a *process reduction phase* respectively.

The process reduction phase presented in Figure 7 treats an active ordinary process P by the normal execution of its constraint logic program. This phase carries out one of the following three operations. The first operation changes P into a finished one if P has an empty goal. The second operation reduces P w.r.t. a non-askable atom L in a similar way to the CLP scheme. The third operation reduces P w.r.t. an askable atom $Q@S$ by using either the ordinary or the default answer entries corresponding to $Q@S$.

The fact arrival phase given in Figure 8 updates the related answer entries and processes when the agent M receives an answer $\langle Q@S, \mathit{AID}, C_p, \mathit{AID}_{\mathrm{prev}}\rangle$ from a child agent S. This phase executes one of the following two operations. The first operation treats a new answer; it reflects the returned constraints C_p by creating new processes from the processes that are referred by the default and original answer entries corresponding to $Q@S$. The second operation handles a revised answer; it adds C_p to the processes using the previous answer whose identifier is $\mathit{AID}_{\mathrm{prev}}$.

4.4. Correctness

We show the correctness of the operational model presented in this section. For this purpose, we begin with two lemmas that hold for local parts of hierarchical multi-agent systems called master-slave restrictions.

Definition 12 (master-slave restriction)**.** Given a multi-agent system $\langle H,\ \mathcal{F}\rangle$ with $\mathcal{F} = \{\langle \Delta_{M'}, \mathcal{P}_{M'}\rangle\}_{M' \in \mathrm{int}(H)}$ and an agent $M \in \mathrm{int}(H)$, the master-slave restriction of $\langle H, \mathcal{F}\rangle$ to M, written $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$, is the multi-agent system $\langle H_M, \{\langle \Delta_M, \mathcal{P}_M\rangle\}\rangle$, where H_M is the agent hierarchy consisting of M as its root and $\mathrm{chi}(M, H)$ as the child agents of M.

The first lemma shows the soundness of master-slave restrictions of hierarchical multi-agent systems in a similar way to the theorem on the soundness of master-slave multi-agent systems [2].

Lemma 1 (soundness of the master-slave restriction)**.** *For any multi-agent system* $\langle H, \mathcal{F}\rangle$*, any agent* $M \in \mathrm{int}(H)$*, any initial goal "*$\leftarrow \| \, Q_{\mathrm{init}}@M$*", any execution of* M*, and any process* P *of* M*, there exists a sequence of reductions "*$\leftarrow \| \, Q_{\mathrm{init}}@M$*", ..., "*$\leftarrow \ C \, \| \, GS_o \cup \mathrm{gs}(P)$*" w.r.t.* $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ *and* $\mathcal{A}_P$ *such that* $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P))$ *entails* $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C)$*, where*

$$GS_o = \{Q@S \mid \langle Q@S, o_{\mathrm{s}}\rangle \in \mathrm{wa}(P) \vee \langle Q@S, o_{\mathrm{n}}\rangle \in \mathrm{wa}(P)\}$$

$$\mathcal{A}_P = \{\, \text{“} Q@S \leftarrow C' \| \text{”} \mid$$

```
foreach askable atom $Q@S$ that appears in $\mathcal{P}_M$ do
    create original answer entries $\langle Q@S, o_\mathrm{s}, \mathsf{true}, \emptyset\rangle$ and $\langle Q@S, o_\mathrm{n}, \mathsf{true}, \emptyset\rangle$;
    foreach default rule "$Q@S \leftarrow C_d||$" $\in \Delta_M$ do
        create a default answer entry $\langle Q@S, d, C_d, \emptyset\rangle$;
repeat
    if a question $Q_\mathrm{init}$ was asked by the parent then
        $PID$ := a new process ID;
        create an ordinary process $\langle PID, \mathsf{true}, \{Q_\mathrm{init}@M\}, \emptyset, \emptyset\rangle$;
    else if an answer $\langle Q@S, AID, C_p, AID_\mathrm{prev}\rangle$ arrived from a child then
        invoke a fact arrival phase for $\langle Q@S, AID, C_p, AID_\mathrm{prev}\rangle$;
    else if there is an active ordinary process $P$ then
        invoke a process reduction phase for $P$;
until $M$ is terminated ;
```

Fig. 6. Processing an internal agent M.

there exists an answer entry

$\langle Q@S, AID, C', UPS\rangle$ *for* M

such that $\langle Q@S, AID\rangle \in \mathrm{aa}(P)\}$,

and $\pi_V(C)$ *indicates the projection of a constraint* C *onto a set* V *of variables.*

Proof. See Appendix. □

Intuitively, Lemma 1 states that, for any process P of an internal agent M, there exists a sequence of reductions w.r.t. the master-slave restriction to M that yields a constraint entailed by the constraint of P. We prove this lemma by induction on the number of steps for the execution of M. At the induction step, we show that, for any execution consisting of $n+1$ steps, we can construct such a sequence of reductions from that for an execution with n steps.

The second lemma gives the completeness of master-slave restrictions of hierarchical multi-agent systems.

Lemma 2 (completeness of the master-slave restriction)**.** *For any multi-agent system* $\langle H, \mathcal{F}\rangle$, *any agent* $M \in \mathrm{int}(H)$, *any initial goal* "$\leftarrow || Q_\mathrm{init}@M$", *any execution of* M, *any valuation* θ *of* $\mathrm{var}(Q_\mathrm{init})$ *that satisfies the answer obtained from some derivation of* "$\leftarrow || Q_\mathrm{init}@M$" *w.r.t.* $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ *and* $\mathcal{A}_M$, *where*

$\mathcal{A}_M = \{$ "$Q@S \leftarrow C'||$" $|$

there exists an ordinary answer entry

$\langle Q@S, AID, C', UPS\rangle$ *for* $M\}$,

there exist an active process P *of* M *and a sequence of reductions* "$\leftarrow \mathrm{pconst}(P) \,||\, \mathrm{gs}(P)$", ..., "$\leftarrow C||$" *w.r.t.* $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ *and* $\mathcal{A}_M$ *such that* θ *satisfies* C.

Proof. See Appendix. □

Intuitively, Lemma 2 says that, for any valuation θ obtained from some derivation w.r.t. the master-slave restriction of an internal agent M, there exists an active process P of M that will derive an answer including θ. We prove this lemma by induction on the number of steps for the execution of M. At the induction step, we show that, for any execution consisting of $n+1$ steps, we can find such an active process by comparing it with a "reduced" execution consisting of n steps.

We prove two theorems to show that hierarchical multi-agent systems eventually obtain all and only correct solutions. This state is called hierarchical stability.

Definition 13 (hierarchical stability)**.** An execution of a multi-agent system $\langle H, \mathcal{F}\rangle$ is hierarchically stable if and only if the following conditions hold:

– for any agent $M \in \mathrm{int}(H)$, there exists no active ordinary process P of M;
– for any agent $M \in \mathrm{int}(H)$ and any active finished process P of M, there exists no default answer entry A_d for M such that $\langle Q@S, \mathrm{aid}(A_d)\rangle \in \mathrm{aa}(P)$ and $S \in \mathrm{int}(H)$, where $Q@S = \mathrm{aq}(A_d)$;
– for any agent $S \in \mathrm{int}(H) \setminus \{\mathrm{root}(H)\}$, there exists an ordinary answer entry $\langle Q@S, AID, C, UPS\rangle$ for $\mathrm{par}(S)$ if and only if there exists an active finished process P of S such that $AID = \mathrm{pid}(P)$, $C = \mathrm{pconst}(P)$, and "$\leftarrow || Q@S$" is the initial goal of P;

```
if $\mathrm{gs}(P) = \emptyset$ then
    change $P$ into a finished process with the same data;
    answer $\langle Q_{\mathrm{init}}@M, \mathrm{pid}(P), \mathrm{pconst}(P), \mathsf{nil}\rangle$ to the parent;
else
    select an atom $L$ from $\mathrm{gs}(P)$;
    if $L$ is a non-askable atom then
        foreach rule $R \in \mathcal{P}_M$ do
            $C := \mathrm{pconst}(P) \wedge (L = \mathrm{head}(R)) \wedge \mathrm{const}(R)$;
            if $C$ is consistent then
                $PID$ := a new process ID;  $GS := \mathrm{body}(R) \cup \mathrm{gs}(P) \setminus \{L\}$;
                create an ordinary process $\langle PID, C, GS, \emptyset, \mathrm{aa}(P)\rangle$;
                foreach answer entry $A$ s.t. $\langle \mathrm{aq}(A), \mathrm{aid}(A)\rangle \in \mathrm{aa}(P)$ do
                    $\mathrm{ups}(A) := \mathrm{ups}(A) \cup \{PID\}$;
        foreach answer entry $A$ s.t. $\langle \mathrm{aq}(A), \mathrm{aid}(A)\rangle \in \mathrm{aa}(P)$ do
            $\mathrm{ups}(A) := \mathrm{ups}(A) \setminus \{\mathrm{pid}(P)\}$;
        kill $P$;
    else // $L$ is an askable atom $Q@S$.
        $Q@S := L$;
        $AS_p := \{A_p \mid A_p$ is an ordinary answer entry s.t. $\mathrm{aq}(A_p) = Q@S\}$;
        if $AS_p \neq \emptyset$ then  $AS := AS_p$;  $AID_o := o_{\mathrm{n}}$;
        else  $AS := \{A_d \mid A_d$ is a default answer entry s.t. $\mathrm{aq}(A_d) = Q@S\}$;  $AID_o := o_{\mathrm{s}}$;
        foreach answer entry $A \in AS$ do
            $C := \mathrm{pconst}(P) \wedge \mathrm{aconst}(A)$;
            if $C$ is consistent then
                $PID$ := a new process ID;  $GS := \mathrm{gs}(P) \setminus \{Q@S\}$;  $AA := \mathrm{aa}(P) \cup \{\langle Q@S, \mathrm{aid}(A)\rangle\}$;
                create an ordinary process $\langle PID, C, GS, \emptyset, AA\rangle$;
                $\mathrm{ups}(A) := \mathrm{ups}(A) \cup \{PID\}$;
                foreach answer entry $A'$ s.t. $\langle \mathrm{aq}(A'), \mathrm{aid}(A')\rangle \in \mathrm{aa}(P)$ do
                    $\mathrm{ups}(A') := \mathrm{ups}(A') \cup \{PID\}$;
        select the original answer entry $A_o$ s.t. $\mathrm{aq}(A_o) = Q@S \wedge \mathrm{aid}(A_o) = AID_o$;
        if $AID_o = o_{\mathrm{s}} \wedge \mathrm{ups}(A_o) = \emptyset$ then  send a question $Q$ to the child $S$;
        $\mathrm{ups}(A_o) := \mathrm{ups}(A_o) \cup \{\mathrm{pid}(P)\}$;  $\mathrm{gs}(P) := \mathrm{gs}(P) \setminus \{Q@S\}$;  $\mathrm{wa}(P) := \{\langle Q@S, AID_o\rangle\}$;
```

Fig. 7. Process reduction phase for an active ordinary process P.

– for any agent $S \in \mathrm{ext}(H)$, there exists an ordinary answer entry $\langle Q@S, AID, C, UPS\rangle$ for $\mathrm{par}(S)$ if and only if there exists a most recent answer "$Q@S \leftarrow C||$" of S whose answer identifier is AID.

The first theorem provides the soundness of hierarchical multi-agent systems; i.e. only correct solutions in the sense of the semantics are eventually computed.

Theorem 1 (soundness of the hierarchically stable system)**.** *For any multi-agent system $\langle H, \mathcal{F}\rangle$ with $M_{\mathrm{root}} = \mathrm{root}(H)$, any initial goal "$\leftarrow || \, Q_{\mathrm{init}}@M_{\mathrm{root}}$", any hierarchically stable execution of $\langle H, \mathcal{F}\rangle$ with a set $\mathcal{A}_H$ of most recent answers of the external agents, and any active finished process P of M_{root}, there exists a derivation "$\leftarrow || \, Q_{\mathrm{init}}@M_{\mathrm{root}}$", ..., "$\leftarrow C||$" w.r.t. $\langle H, \mathcal{F}\rangle$ and $\mathcal{A}_H$ such that $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P))$ entails $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C)$.*

Proof. See Appendix. □

Intuitively, Theorem 1 claims that, for any active finished process P of the root agent in a hierarchically stable multi-agent system, there exists a derivation w.r.t. the multi-agent system that yields a constraint entailed by the constraint of P. We prove this theorem by induction on the tree structure of the multi-agent system. At the induction step, we show that, for any subtree of the multi-agent system, we can construct such a derivation by applying Lemma 1.

```
if $AID_{\mathrm{prev}} = \mathsf{nil}$ then // A new answer arrived.
    create an ordinary answer entry $A_p = \langle Q@S, AID, C_p, \emptyset\rangle$;
    foreach default answer entry $A_d$ s.t. $\mathrm{aq}(A_d) = Q@S$ do
        foreach process $P_d$ s.t. $\mathrm{pid}(P_d) \in \mathrm{ups}(A_d)$ do
            $C := \mathrm{pconst}(P_d) \wedge C_p$;
            if $C$ is consistent then
                $PID :=$ a new process ID;  $WA := \mathrm{wa}(P_d) \setminus \{\langle Q@S, \mathrm{aid}(A_d)\rangle\}$;
                $AA := (\mathrm{aa}(P_d) \setminus \{\langle Q@S, \mathrm{aid}(A_d)\rangle\}) \cup \{\langle Q@S, AID\rangle\}$;
                create a process $\langle PID, C, \mathrm{gs}(P_d), WA, AA\rangle$ having the same type as $P_d$;
                $\mathrm{ups}(A_p) := \mathrm{ups}(A_p) \cup \{PID\}$;
                if $P_d$ is an active finished process then
                    answer $\langle Q_{\mathrm{init}}@M, PID, C, \mathrm{pid}(P_d)\rangle$ to the parent;
            else // $C$ is inconsistent.
                if $P_d$ is an active finished process then
                    answer $\langle Q_{\mathrm{init}}@M, \mathrm{pid}(P_d), \mathsf{false}, \mathrm{pid}(P_d)\rangle$ to the parent;
            $\mathrm{wa}(P_d) := \mathrm{wa}(P_d) \cup \{\langle Q@S, \mathrm{aid}(A_d)\rangle\}$;  $\mathrm{aa}(P_d) := \mathrm{aa}(P_d) \setminus \{\langle Q@S, \mathrm{aid}(A_d)\rangle\}$;
    foreach original answer entry $A_o$ s.t. $\mathrm{aq}(A_o) = Q@S$ do
        foreach process $P_o$ s.t. $\mathrm{pid}(P_o) \in \mathrm{ups}(A_o)$ do
            $C := \mathrm{pconst}(P_o) \wedge C_p$;
            if $\mathrm{aid}(A_o) = o_{\mathrm{s}}$ then $C := C \wedge (\bigwedge_{\text{“}Q@S \leftarrow C_d ||\text{”} \in \Delta_M} \neg C_d)$;
            if $C$ is consistent then
                $PID :=$ a new process ID;  $WA := \mathrm{wa}(P_o) \setminus \{\langle Q@S, \mathrm{aid}(A_o)\rangle\}$;  $AA := \mathrm{aa}(P_o) \cup \{\langle Q@S, AID\rangle\}$;
                create a process $\langle PID, C, \mathrm{gs}(P_o), WA, AA\rangle$ having the same type as $P_o$;
                $\mathrm{ups}(A_p) := \mathrm{ups}(A_p) \cup \{PID\}$;
else // A revised answer arrived.
    select the ordinary answer entry $A_p$ s.t. $\mathrm{aq}(A_p) = Q@S \wedge \mathrm{aid}(A_p) = AID_{\mathrm{prev}}$;
    $\mathrm{aid}(A_p) := AID$;  $\mathrm{aconst}(A_p) := C_p$;  $UPS := \mathrm{ups}(A_p)$;
    foreach process $P$ s.t. $\mathrm{pid}(P) \in UPS$ do
        $\mathrm{pconst}(P) := \mathrm{pconst}(P) \wedge C_p$;
        if $P$ is an active finished process then
            answer $\langle Q_{\mathrm{init}}@M, \mathrm{pid}(P), \mathrm{pconst}(P), \mathrm{pid}(P)\rangle$ to the parent;
        if $\mathrm{pconst}(P)$ is consistent then
            $\mathrm{aa}(P) := (\mathrm{aa}(P) \setminus \{\langle Q@S, AID_{\mathrm{prev}}\rangle\}) \cup \{\langle Q@S, AID\rangle\}$;
        else // $\mathrm{pconst}(P)$ is inconsistent.
            $\mathrm{ups}(A_p) := \mathrm{ups}(A_p) \setminus \{\mathrm{pid}(P)\}$;
            kill $P$;
```

Fig. 8. Fact arrival phase for an answer $\langle Q@S, AID, C_p, AID_{\mathrm{prev}}\rangle$.

The second theorem gives the completeness of hierarchical multi-agent systems; i.e. all correct solutions in the sense of the semantics are eventually computed.

Theorem 2 (completeness of the hierarchically stable system)**.** *For any multi-agent system $\langle H, \mathcal{F}\rangle$ with $M_{\mathrm{root}} = \mathrm{root}(H)$, any initial goal “$\leftarrow || \, Q_{\mathrm{init}}@M_{\mathrm{root}}$”, any hierarchically stable execution of $\langle H, \mathcal{F}\rangle$ with a set $\mathcal{A}_H$ of most recent answers of the external agents, and any valuation θ of $\mathrm{var}(Q_{\mathrm{init}})$ that satisfies the answer obtained from some derivation of “$\leftarrow || \, Q_{\mathrm{init}}@M_{\mathrm{root}}$” w.r.t. $\langle H, \mathcal{F}\rangle$ and $\mathcal{A}_H$, there exists an active finished process P of M_{root} such that θ satisfies $\mathrm{pconst}(P)$.*

Proof. See Appendix. □

Intuitively, Theorem 2 states that, if the execution of a multi-agent system is hierarchically stable, for any valuation θ obtained from some derivation w.r.t. the multi-agent system, the root agent has an active finished process p whose answer includes θ. We prove this theorem by induction on the tree structure of the multi-agent system. At the induction step, we show that, for any subtree of the multi-agent system, we can find such an active finished process by applying Lemma 2.

5. Implementation

Using the operational model proposed in the previous section, we have developed a prototype system, called SpecCp, for speculative constraint processing for hierarchical multi-agent systems.[4] Our current implementation is written in the Objective Caml[5] language, and consists of approximately 2500 lines of code. The system implements the necessary basic mechanisms for the CLP scheme, including a finite-domain constraint solver.[6]

Instead of truly concurrent execution, the prototype system performs pseudo-concurrent execution of agents in a serialized manner; it performs one atomic operation of an agent at a time by possibly selecting different agents one after another. The system provides an interactive interpreter that allows a user to experiment with various executions. At every step, a user is prompted to select which agent to execute next, together with which process of the selected agent to reduce, or which answer to be received by the agent. Thus the pseudo-concurrent execution of agents is completely under the control of the user.

6. Illustrative Example

This section shows an example of executing a multi-agent system. We use the multi-agent system for the room reservation problem presented in Example 2. We executed it by running our prototype system described in the previous section. Below we illustrate its execution by extracting parts of the textual output of the prototype system from the entire output.[7]

We begin by asking the root agent r a goal "$\leftarrow$ *reserve*$(R, L, D)@r$" (Figure 9(a)), and then obtain the following state of r.[8]

```
Answer entries:
Ordinary processes:
(1, {}, {rsv(R,L,D)}, {}, {})
Finished processes:
```

At present, there is only one ordinary process whose ID is 1 and that contains the initial goal, and there is no finished process. Answer entries exist internally, but are not printed here since they are not yet used by any processes.

Next, the system indicates that the ordinary process 1 of agent r can be reduced, and therefore the user chooses its reduction (by entering "`r 1`" after "`Which step?`").

```
SELECT NEXT STEP
Reducible processes (agent pid):
r 1
Receivable answers (receiver sender):
none
Which step?
r 1
```

Then the system prints out the following state of agent r, which means that the process was reduced into the new processes 2, 3, and 4.

```
Answer entries:
Ordinary processes:
(2, {L=[a,b], R=tr}, {av(D)@a, av(D)@b},
 {}, {})
(3, {L=[a], R=sr}, {av(D)@a, unav(D)@b},
 {}, {})
(4, {L=[b], R=sr}, {unav(D)@a, av(D)@b},
 {}, {})
Finished processes:
```

After the user selects the reduction of the process 2, the system outputs the following, where agent r sends a question *available*(D) to its child agent a.[9]

```
Agent r asked: av(D@r#2)@a

Answer entries:
(av(D)@a, os, {true}, {2})
(av(D)@a, d(1), {D:{1,2,3}}, {5})
Ordinary processes:
(2, {L=[a,b], R=tr}, {av(D)@b},
```

[4]`http://www.informaticians.org/speccp/`

[5]`http://caml.inria.fr/ocaml/`

[6]We could have reduced the work if we had implemented our prototype system on top of a CLP language rather than Objective Caml, which is a functional programming language. However, we adopted Objective Caml simply because the primary developer has considerable experience in functional programming.

[7]We sometimes break a long output line to fit it to the page.

[8]For brevity, we denote *reserve* as `rsv`, *available* as `av`, *unavailable* as `unav`, *twin_room* as `tr`, *single_room* as `sr`, *free* as `fr`, and *busy* as `bs`.

[9]In the asked question `av(D@r#2)@a`, the variable `D` was renamed into `D@r#2` to avoid a conflict of the variable name.

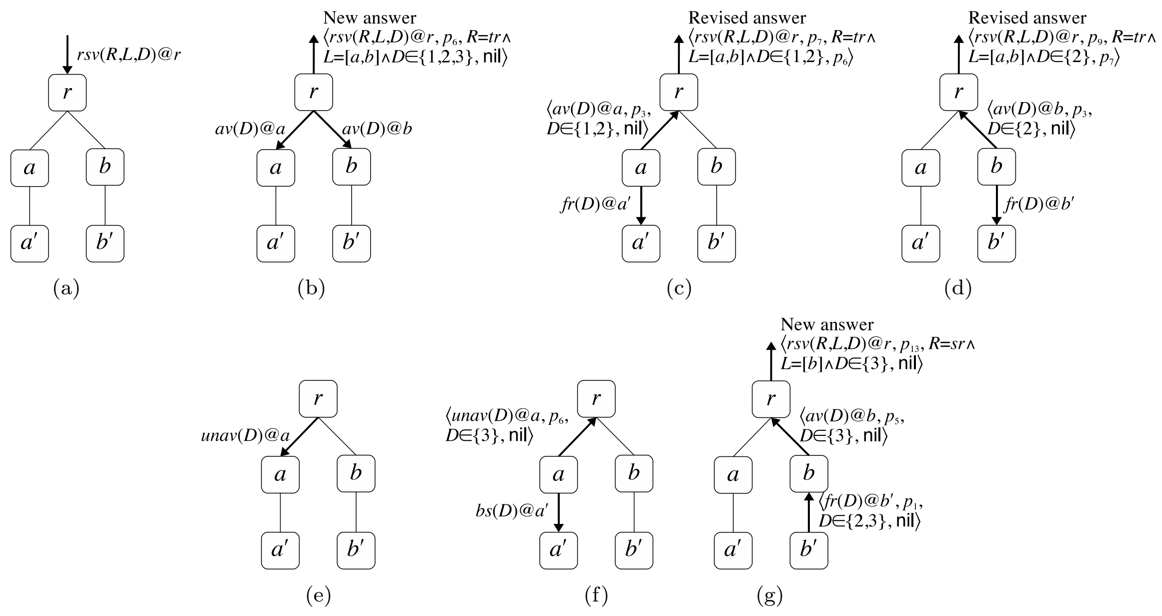
rsv(R,L,D)@r
r
a
b
a′
b′
(a)
New answer
⟨rsv(R,L,D)@r, p₆, R=tr∧ L=[a,b]∧D∈{1,2,3}, nil⟩
av(D)@a
av(D)@b
(b)
Revised answer
⟨rsv(R,L,D)@r, p₇, R=tr∧ L=[a,b]∧D∈{1,2}, p₆⟩
⟨av(D)@a, p₃, D∈{1,2}, nil⟩
fr(D)@a′
(c)
Revised answer
⟨rsv(R,L,D)@r, p₉, R=tr∧ L=[a,b]∧D∈{2}, p₇⟩
⟨av(D)@b, p₃, D∈{2}, nil⟩
fr(D)@b′
(d)
unav(D)@a
(e)
⟨unav(D)@a, p₆, D∈{3}, nil⟩
bs(D)@a′
(f)
New answer
⟨rsv(R,L,D)@r, p₁₃, R=sr∧ L=[b]∧D∈{3}, nil⟩
⟨av(D)@b, p₅, D∈{3}, nil⟩
⟨fr(D)@b′, p₁, D∈{2,3}, nil⟩
(g)

Fig. 9. Execution of the multi-agent system in Example 2.

```
 {(av(D)@a, os)}, {})
(3, {L=[a], R=sr}, {av(D)@a, unav(D)@b},
 {}, {})
(4, {L=[b], R=sr}, {unav(D)@a, av(D)@b},
 {}, {})
(5, {L=[a,b], R=tr, D:{1,2,3}},
 {av(D)@b}, {}, {(av(D)@a, d(1))})
Finished processes:
```

After further two reductions (during which agent r sends a question $available(D)$ to agent b), the system yields the following (Figure 9(b)).

```
Agent r returned to Root_caller
the new answer:
(rsv(R,L,D)@r, p(6),
 {R=tr, L=[a,b], D:{1,2,3}}, nil)

Answer entries:
(av(D)@a, os, {true}, {2})
(av(D)@a, d(1), {D:{1,2,3}}, {5,6})
(av(D)@b, os, {true}, {5})
(av(D)@b, d(2), {D:{1,2,3}}, {6})
Ordinary processes:
(2, {L=[a,b], R=tr}, {av(D)@b},
 {(av(D)@a, os)}, {})
(3, {L=[a], R=sr}, {av(D)@a, unav(D)@b},
 {}, {})
(4, {L=[b], R=sr}, {unav(D)@a, av(D)@b},
 {}, {})
(5, {L=[a,b], R=tr, D:{1,2,3}}, {},
 {(av(D)@b, os)}, {(av(D)@a, d(1))})
Finished processes:
(6, {L=[a,b], R=tr, D:{1,2,3},
     D:{1,2,3}}, {}, {},
 {(av(D)@a, d(1)), (av(D)@b, d(2))})
```

Now we have the first answer $R = twin\_room$, $L = [a, b]$, and $D \in \{1, 2, 3\}$. Note that it is a tentative answer speculatively computed from the default rules of r.

Since agent r sent $available(D)$ to its child agent a, it soon returns an answer that it speculatively computes from its default "$free(D)@a' \leftarrow D \in \{1, 2\} \,||$".

```
Agent a returned to r the new answer:
(av(D@r#2)@a, p(3), {D@r#2:{1,2}}, nil)
```

Then agent r returns a revised answer $R = twin\_room$, $L = [a, b]$, and $D \in \{1, 2\}$ (Figure 9(c)).

```
Agent r returned to Root_caller
the revised answer:
(rsv(R,L,D)@r, p(7),
 {R=tr, L=[a,b], D:{1,2}}, p(6))
```

Similarly, agent b returns an answer that it computes from its default rule "$free(D)@b' \leftarrow D \in \{2\} \,||$".

```
Agent b returned to r the new answer:
(av(D@r#2)@b, p(3), {D@r#2:{2}}, nil)
```

Then agent r returns a further revised answer $R = twin\_room$, $L = [a, b]$, and $D \in \{2\}$ (Figure 9(d)).

```
Agent r returned to Root_caller
the revised answer:
(rsv(R,L,D)@r, p(9),
 {R=tr, L=[a,b], D:{2}}, p(7))
```

Next, switch our attention to the ordinary process 4 of agent r. Its reduction causes r to send a question $\mathit{unavailable}(D)$ to agent a (Figure 9(e)).

```
Agent r asked: unav(D@r#4)@a
```

Then agent a returns an answer that it computes from its default rule "$\mathit{busy}(D)@a' \leftarrow D \in \{3\} \,||$" (Figure 9(f)).

```
Agent a returned to r the new answer:
(unav(D@r#4)@a, p(6), {D@r#4:{3}}, nil)
```

Next, suppose that the external agent b' answers $\mathit{free}(D)@b'$ by returning $D \in \{2, 3\}$ to agent b. Then b returns a new answer to agent r after computing the difference.

```
Agent b returned to r the new answer:
(av(D@r#2)@b, p(5), {D@r#2:{3}}, nil)
```

Then r returns an answer $R = \mathit{single\_room}$, $L = [b]$, and $D \in \{3\}$ (Figure 9(g)).

```
Agent r returned to Root_caller
the new answer:
(rsv(R,L,D)@r, p(13),
 {R=sr, L=[b], D:{3}}, nil)
```

Note that this is not a revised answer but a new answer, which means that the answer $R = \mathit{twin\_room}$, $L = [a, b]$, and $D \in \{2\}$ is still valid.

As illustrated in this example, speculative constraint processing for hierarchical multi-agent systems computes tentative solutions as soon as possible by using default rules associated with the internal agents. If such default rules are overridden by answers returned by child agents, previous answers are replaced with narrower revised ones, or new answers are incrementally added.

7. Discussion

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

As described in Section 1, speculative constraint processing handles more expressive questions than our previous speculative computation frameworks [16,18,19] that allow only yes/no questions. However, speculative constraint processing currently does not support negation that is supported in the previous yes/no-type frameworks; in this sense, speculative constraint processing is not a complete generalization of the yes/no-type frameworks. Since negation is often useful for modeling problems, it is desirable to further extend speculative constraint processing to handle negation.

8. Conclusions and Future Work

In this paper, we proposed a logical framework for speculative constraint processing for hierarchical multi-agent systems. We provided an operational model for our new framework that we constructed by extending our previous operational model for master-slave multi-agent systems. We also presented a prototype implementation of the operational model that performs pseudo-concurrent execution of agents in a serialized manner.

Our future work includes a multi-threaded implementation of our new framework. We have already developed a multi-threaded implementation of our previous framework for master-slave multi-agent systems [11]. We will extend the existing multi-threaded implementation to cover hierarchical multi-agent systems. The resulting multi-threaded implementation will enable truly concurrent execution of agents in a distributed environment. We are also interested in supporting negation in speculative constraint processing, since it is useful for modeling various problems as discussed in Section 7.

Acknowledgement

This work was supported in part by the Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B), 19300053.

Appendix

Proof of Lemma 1. We prove this lemma by induction on the number of steps for the execution of M.

Induction base. When a query $Q_{\mathrm{init}}@M$ is asked at the initial step, a process $P = \langle \mathit{PID}, \mathsf{true}, \{Q_{\mathrm{init}}@M\}, \emptyset, \emptyset\rangle$ is created, and therefore this lemma holds.

Induction step. Assume that this lemma holds for any execution with n steps.

Consider any execution with $n+1$ steps. It is straightforward to show that this lemma holds for the process reduction phase.

Here we consider the case that the fact arrival phase treats a new answer $\langle Q@S, \mathit{AID}, C_p, \mathsf{nil}\rangle$. In this case, there is no answer entry in the form $\langle Q@S, \mathit{AID}, C_p, \mathit{UPS}'\rangle$.

Let $\langle Q@S, d, C_d, \mathit{UPS}_d\rangle$ be any default answer entry and P_d be any ordinary process such that $\mathrm{pid}(P_d) \in \mathit{UPS}_d$. By the induction hypothesis, P_d satisfies this lemma for some C'' and $\mathcal{A}^{(n)}_{P_d}$; i.e. there is a sequence of reductions "$\leftarrow \| \, Q_{\mathrm{init}}@M$", …, "$\leftarrow C_1 \, \| \{Q@S\} \cup GS$", "$\leftarrow C_1 \wedge C_d \, \| \, GS$", …, "$\leftarrow C_1 \wedge C_d \wedge C_2 \, \| \{Q'@S' \mid \langle Q'@S', o_{\mathrm{s}}\rangle \in \mathrm{wa}(P_d) \vee \langle Q'@S', o_{\mathrm{n}}\rangle \in \mathrm{wa}(P_d)\} \cup \mathrm{gs}(P_d)$" w.r.t. $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}^{(n)}_{P_d}$ such that $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P_d))$ entails $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C_1 \wedge C_d \wedge C_2)$, where C_1 and C_2 are the constraints obtained before and after processing $Q@S$ respectively.

Assume that $\mathrm{pconst}(P_d) \wedge C_p$ is consistent. Then a process $P = \langle \mathit{PID}, \mathrm{pconst}(P_d) \wedge C_p, \mathrm{gs}(P_d), \mathrm{wa}(P_d) \backslash \{\langle Q@S, d\rangle\}, (\mathrm{aa}(P_d) \backslash \{\langle Q@S, d\rangle\}) \cup \{\langle Q@S, \mathit{AID}\rangle\}\rangle$ is created, and we have $\mathcal{A}^{(n+1)}_{P} = (\mathcal{A}^{(n)}_{P_d} \setminus \{\text{"}Q@S \leftarrow C_d \|\text{"}\}) \cup \{\text{"}Q@S \leftarrow C_p \|\text{"}\}$. Then we can consider the sequence of reductions "$\leftarrow \| \, Q_{\mathrm{init}}@M$", …, "$\leftarrow C_1 \, \| \{Q@S\} \cup GS$", "$\leftarrow C_1 \wedge C_p \, \| \, GS$", …, "$\leftarrow C_1 \wedge C_p \wedge C_2 \, \| \{Q'@S' \mid \langle Q'@S', o_{\mathrm{s}}\rangle \in \mathrm{wa}(P) \vee \langle Q'@S', o_{\mathrm{n}}\rangle \in \mathrm{wa}(P)\} \cup \mathrm{gs}(P)$" w.r.t. $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}^{(n+1)}_{P}$. Then $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P))$ entails $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C_1 \wedge C_p \wedge C_2)$ since $\mathrm{pconst}(P) = C_p \wedge \mathrm{pconst}(P_d)$ and $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P_d))$ entails $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C_1 \wedge C_d \wedge C_2)$. Thus this lemma holds for P.

If there exists no ordinary answer entry A_p such that $\mathrm{aq}(A_p) = Q@S$, this step changes P_d by setting $\mathrm{wa}(P_d) := \mathrm{wa}(P_d) \cup \{\langle Q@S, d\rangle\}$ and $\mathrm{aa}(P_d) := \mathrm{aa}(P_d) \setminus \{\langle Q@S, d\rangle\}$, and hence we have $\mathcal{A}^{(n+1)}_{P_d} = \mathcal{A}^{(n)}_{P_d} \setminus \{\text{"}Q@S \leftarrow C_d \|\text{"}\}$. Otherwise, P_d is unchanged since $\langle Q@S, d\rangle \in \mathrm{wa}(P_d)$ and $\langle Q@S, d\rangle \notin \mathrm{aa}(P_d)$ hold for the original P_d, and thus we have $\mathcal{A}^{(n+1)}_{P_d} = \mathcal{A}^{(n)}_{P_d}$. Therefore, this lemma is kept satisfied for P_d.

Next, let A_o be any original answer entry and P_o be any ordinary process such that $\mathrm{aq}(A_o) = Q@S$ and $\mathrm{pid}(P_o) \in \mathrm{ups}(A_o)$. By the induction hypothesis, P_o satisfies this lemma for some C'' and $\mathcal{A}^{(n)}_{P_o}$; i.e. there is a sequence of reductions "$\leftarrow \| \, Q_{\mathrm{init}}@M$", …, "$\leftarrow C'' \, \| \{Q@S\} \cup \mathrm{gs}(P_o)$" w.r.t. $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}^{(n)}_{P_o}$ such that $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P_o))$ entails $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C'')$. Since this step does not change P_o, this lemma is kept satisfied for P_o.

Assume that $\mathrm{aid}(A_o) = o_{\mathrm{s}}$ and $\mathrm{pconst}(P_o) \wedge C_p \wedge (\bigwedge_{\text{"}Q@S \leftarrow C_d \|\text{"} \in \Delta_M} \neg C_d)$ is consistent. Then a process $P = \langle \mathit{PID}, \mathrm{pconst}(P_o) \wedge C_p \wedge \bigwedge_{\text{"}Q@S \leftarrow C_d \|\text{"} \in \Delta_M} \neg C_d, \mathrm{gs}(P_o), \mathrm{wa}(P_o) \backslash \{\langle Q@S, o_{\mathrm{s}}\rangle\}, \mathrm{aa}(P_o) \cup \{\langle Q@S, \mathit{AID}\rangle\}\rangle$ is created, and we have $\mathcal{A}^{(n+1)}_{P} = \mathcal{A}^{(n)}_{P_o} \cup \{\text{"}Q@S \leftarrow C_p \|\text{"}\}$. Then we can consider the sequence of reductions "$\leftarrow \| \, Q_{\mathrm{init}}@M$", …, "$\leftarrow C'' \, \| \{Q@S\} \cup \mathrm{gs}(P)$", "$\leftarrow C'' \wedge C_p \, \| \, \mathrm{gs}(P)$" w.r.t. $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}^{(n+1)}_{P}$. Then $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P))$ entails $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C'' \wedge C_p)$ since $\mathrm{pconst}(P) = \mathrm{pconst}(P_o) \wedge C_p \wedge \bigwedge_{\text{"}Q@S \leftarrow C_d \|\text{"} \in \Delta_M} \neg C_d$ and $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(\mathrm{pconst}(P_o))$ entails $\pi_{\mathrm{var}(Q_{\mathrm{init}})}(C'')$. Therefore, this lemma holds for P. Also, if $\mathrm{aid}(A_o) = o_{\mathrm{n}}$, we can similarly show that this lemma holds for the created process.

This lemma is kept satisfied for the other processes that are not handled in this case, since those processes and their most recent answer sets are unchanged. Therefore, this lemma holds for any processes after processing a new answer in the fact arrival phase.

Similarly, we can show that this lemma holds for the case that the fact arrival phase treats a revised answer. Thus this lemma holds in all the cases. □

Proof of Lemma 2. We prove this lemma by induction on the number of steps for the execution of M.

Induction base. When $Q_{\text{init}}@M$ is asked at the initial step, a process $P = \langle \mathit{PID}, \mathsf{true}, \{Q_{\text{init}}@M\}, \emptyset, \emptyset\rangle$ is created, and therefore this lemma holds.

Induction step. Assume that this lemma holds for any execution with n steps.

Consider any execution with $n+1$ steps. We can construct a "reduced" execution by skipping the last process reduction phase while keeping the other process reduction phases and all the fact arrival phases. Note that this reduced execution consists of n steps and has the same set of most recent answers as the original execution that consists of $n+1$ steps. Then we have the following cases: (A) the skipped process reduction phase treats a non-askable atom; (B) the skipped process reduction phase treats an askable atom $Q@S$.

Consider case (A). By the induction hypothesis and the completeness of the CLP scheme, it is straightforward to show that this lemma holds.

Next, consider case (B). If there exists an active process P' of M such that P' remains both in the original and reduced executions, and that θ is derived from P', this lemma clearly holds. Next, assume that there exists no such active process P' of M. Let P be the process in the original execution for which the process reduction phase is skipped in the reduced execution. We assume for simplicity that $\langle Q@S, \mathit{AID}\rangle \notin \text{aa}(P)$ for any AID, and also that there exists no ordinary answer entry A_p such that $\text{aq}(A_p) = Q@S$. Consider any sequence of active processes successively derived from P in the original execution by using a default rule "$Q@S \leftarrow C_d^i||$". Since the skipped process reduction phase is the last one, these processes except the first one come from the remaining fact arrival phases. Let $C_d^i, C_1^0, \ldots, C_{l_0}^0, C_p^1, C_1^1, \ldots, C_{l_1}^1, C_p^2, C_1^2, \ldots, C_{l_2}^2, \ldots, C_p^m, C_1^m, \ldots, C_{l_m}^m$ be the constraints successively added in this order to these processes, where C_p^1 is a new answer w.r.t. $Q@S$, each C_p^j $(2 \leq j \leq m)$ is a revised answer whose previous answer is C_p^{j-1}, and each C_k^j $(0 \leq j \leq m$ and $1 \leq k \leq l_j)$ is an answer w.r.t. another askable atom than $Q@S$. By contrast, in the reduced execution, because of the skipped process reduction phase, the constraints $C_1^0, \ldots, C_{l_0}^0, C_1^1, \ldots, C_{l_1}^1, C_1^2, \ldots, C_{l_2}^2, \ldots, C_1^m, \ldots, C_{l_m}^m$ are added, and $Q@S$ remains in the goal of the resulting process. By the induction hypothesis, θ must be derived from such a resulting process in the reduced execution. If θ is derived from the corresponding process in the original execution that uses some C_d^i, this lemma holds. Otherwise, we can choose an alternative sequence of active processes successively derived from P that use the following constraints: $C_1^0, \ldots, C_{l_0}^0, C_p^1 \wedge (\bigwedge_{\text{“}Q@S \leftarrow C_d^i||\text{”} \in \Delta_M} \neg C_d^i), C_1^1, \ldots, C_{l_1}^1, C_p^2, C_1^2, \ldots, C_{l_2}^2, \ldots, C_p^m, C_1^m, \ldots, C_{l_m}^m$. Thus θ is derived from the resulting process in the original execution. Therefore, this lemma holds. □

Proof of Theorem 1. We prove this theorem by induction on the tree structure of $\langle H, \mathcal{F}\rangle$. For this purpose, we introduce hierarchical restrictions of $\langle H, \mathcal{F}\rangle$. Let $\mathcal{F} = \{\langle \Delta_{M'}, \mathcal{P}_{M'}\rangle\}_{M' \in \text{int}(H)}$. The hierarchical restriction of $\langle H, \mathcal{F}\rangle$ to an agent $M \in \text{int}(H)$, written $\text{hres}(M, \langle H, \mathcal{F}\rangle)$, is defined as the the multi-agent system $\langle H_M, \mathcal{F}_M\rangle$, where H_M is the agent hierarchy consisting of M as the root and all the descendant agents of M in H, and $\mathcal{F}_M = \{\langle \Delta_{M'}, \mathcal{P}_{M'}\rangle\}_{M' \in \text{int}(H_M)}$. We also define $\text{height}(H_M)$ as the height of H_M, i.e. the number of the agents along the longest path from the root to an external agent of H_M. Below we prove the proposition $(*)$ that, for any $M \in \text{int}(H)$ and any active finished process P_M of M, there exists a derivation "$\leftarrow || \, Q_{P_M}@M$", …, "$\leftarrow C_{P_M}||$" w.r.t. $\langle H_M, \mathcal{F}_M\rangle = \text{hres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}_H$ such that $\pi_{\text{var}(Q_{P_M})}(\text{pconst}(P_M))$ entails $\pi_{\text{var}(Q_{P_M})}(C_{P_M})$.

Induction base. Consider any $M \in \text{int}(H)$ such that $\text{height}(H_M) = 2$ (which is the minimum). Since all the children of M are external agents, $(*)$ holds by Lemma 1.

Induction step. Assume that, for any $M \in \text{int}(H)$ such that $\text{height}(H_M) \leq n$, $(*)$ holds. Consider any $M \in \text{int}(H)$ such that $\text{height}(H_M) = n+1$. Let P_M be an arbitrary active finished process of M. Then, for any $\langle Q@S, \mathit{AID}\rangle \in \text{aa}(P_M)$, we have the following cases: (A) $S \in \text{ext}(H)$; (B) $S \in \text{int}(H)$.

Consider case (A). There exists an answer entry $\langle Q@S, \mathit{AID}, C_S, \mathit{UPS}\rangle$ for M. If it is an ordinary answer entry, there exists a most recent answer "$Q@S \leftarrow C_S||$" in $\mathcal{A}_H$; otherwise, it is a default answer entry, and there exists a default rule "$Q@S \leftarrow C_S||$" in Δ_M. Thus "$Q@S \leftarrow C_S||$" is in $\text{bel}(\mathcal{A}_H, \langle H_M, \mathcal{F}_M\rangle)$.

Next, consider case (B). There exist an ordinary answer entry $\langle Q@S, \mathit{AID}, C_S, \mathit{UPS}\rangle$ for M and an active finished process P_S of S such that $\mathit{AID} =$

$\mathrm{pid}(P_S)$ and $C_S = \mathrm{pconst}(P_S)$. By the induction hypothesis, there exists a derivation "$\leftarrow || \, Q@S$", ..., "$\leftarrow C_{P_S}||$" w.r.t. $\mathrm{hres}(S, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}_H$ such that $\pi_{\mathrm{var}(Q)}(C_S)$ entails $\pi_{\mathrm{var}(Q)}(C_{P_S})$.

Let "$\leftarrow || \, Q_{P_M}@M$" be the initial goal of P_M. By Lemma 1, there exists a sequence of reductions "$\leftarrow || \, Q_{P_M}@M$", ..., "$\leftarrow C''||$" w.r.t. $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}_{P_M}$ such that $\pi_{\mathrm{var}(Q_{P_M})}(\mathrm{pconst}(P_M))$ entails $\pi_{\mathrm{var}(Q_{P_M})}(C'')$, where $\mathcal{A}_{P_M} = \{$"$Q@S \leftarrow C'||$" $|$ there exists an answer entry $\langle Q@S, AID, C', UPS\rangle$ for M such that $\langle Q@S, AID\rangle \in \mathrm{aa}(P_M)\}$. Then, replacing each reduction using "$Q@S \leftarrow C_S||$" for $S \in \mathrm{int}(H)$ with the same reductions as in "$\leftarrow || \, Q@S$", ..., "$\leftarrow C_{P_S}||$" above, we can construct a derivation "$\leftarrow || \, Q_{P_M}@M$", ..., "$\leftarrow C_{P_M}||$" w.r.t. $\langle H_M, \mathcal{F}_M\rangle$ and $\mathcal{A}_H$. Also, since $\pi_{\mathrm{var}(Q)}(C_S)$ entails $\pi_{\mathrm{var}(Q)}(C_{P_S})$ for any C_S, $\pi_{\mathrm{var}(Q_{P_M})}(\mathrm{pconst}(P_M))$ entails $\pi_{\mathrm{var}(Q_{P_M})}(C_{P_M})$. □

Proof of Theorem 2. We prove this theorem by induction on the tree structure of $\langle H, \mathcal{F}\rangle$. As in the proof of Theorem 1, we use hierarchical restrictions of $\langle H, \mathcal{F}\rangle$ and their heights. Below we prove the proposition $(*)$ that, for any $M \in \mathrm{int}(H)$ and any valuation θ_M of $\mathrm{var}(Q_M)$ that satisfies the answer obtained from some derivation of "$\leftarrow || \, Q_M@M$" w.r.t. $\langle H_M, \mathcal{F}_M\rangle = \mathrm{hres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}_H$, there exists an active finished process P_M of M such that θ_M satisfies $\mathrm{pconst}(P_M)$.

Induction base. Consider any $M \in \mathrm{int}(H)$ such that $\mathrm{height}(H_M) = 2$. Since all the children of M are external agents, $(*)$ holds by Lemma 2.

Induction step. Assume that, for any $M \in \mathrm{int}(H)$ such that $\mathrm{height}(H_M) \le n$, $(*)$ holds. Consider any $M \in \mathrm{int}(H)$ such that $\mathrm{height}(H_M) = n+1$. Define $\mathcal{A}_M$ as in Lemma 2. Let θ_M be an arbitrary valuation as defined for $(*)$. Then, for any "$Q@S \leftarrow C_S||$" with $S \in \mathrm{chi}(M, H)$ that is used in the derivation of "$\leftarrow || \, Q_M@M$", we have the following cases: (A) $S \in \mathrm{ext}(H)$; (B) $S \in \mathrm{int}(H)$.

Consider case (A). Then any "$Q@S \leftarrow C_S||$" in $\mathcal{A}_H$ is also in $\mathcal{A}_M$.

Next, consider case (B). By the induction hypothesis, for any valuation θ_S of $\mathrm{var}(Q)$ that satisfies the answer obtained from some derivation of "$Q@S \leftarrow C_S||$", there exists an active finished process P_S of S such that θ_S satisfies $\mathrm{pconst}(P_S)$. Because of the hierarchical stability, "$Q@S \leftarrow \mathrm{pconst}(P_S)||$" is in $\mathcal{A}_M$.

Thus we can construct a derivation "$\leftarrow || \, Q_M@M$", ..., "$\leftarrow C_M||$" w.r.t. $\mathrm{msres}(M, \langle H, \mathcal{F}\rangle)$ and $\mathcal{A}_M$ such that θ_M satisfies C_M. Then, by Lemma 2, there exists an active finished process P_M of M such that θ_M satisfies $\mathrm{pconst}(P_M)$. □

References

[1] F. W. Burton. Speculative computation, parallelism, and functional programming. *IEEE Trans. Comput.*, 34(12):1190–1193, 1985.

[2] M. Ceberio, H. Hosobe, and K. Satoh. Speculative constraint processing with iterative revision for disjunctive answers. In *Post-proc. Intl. Workshop on Computational Logic in Multi-Agent Systems (CLIMA-VI)*, volume 3900 of *LNAI*, pages 340–357, 2006.

[3] S. Gregory. Experiments with speculative parallelism in Parlog. In *Proc. Intl. Symp. on Logic Programming (ILPS'93)*, pages 370–387, 1993.

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

[5] H. Hosobe, K. Satoh, and P. Codognet. Agent-based speculative constraint processing. *IEICE Trans. Inf. & Syst.*, E90-D(9):1354–1362, 2007.

[6] K. Inoue and K. Iwanuma. Speculative computation through consequence-finding in multi-agent environments. *Ann. Math. Artif. Intell.*, 42(1–3):255–291, 2004.

[7] J. Jaffar, M. Maher, K. Marriott, and P. Stuckey. The semantics of constraint logic programs. *J. Log. Program.*, 37(1–3):1–46, 1998.

[8] S. Janson and S. Haridi. Programming paradigms of the Andorra Kernel Language. In *Proc. Intl. Symp. on Logic Programming (ISLP'91)*, pages 167–183, 1991.

[9] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 2006.

[10] R. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Ann. Math. Artif. Intell.*, 25(3–4):391–419, 1999.

[11] J. Ma, A. Russo, K. Broda, H. Hosobe, and K. Satoh. On the implementation of speculative constraint processing. In *Proc. Intl. Workshop on Computational Logic in Multi-Agent Systems (CLIMA-X)*, pages 105–120, 2009.

[12] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1–2):149–180, 2005.

[13] L. Morgenstern. A formal theory of multiple agent nonmonotonic reasoning. In *Proc. Natl. Conf. on Artificial Intelligence (AAAI-90)*, pages 538–544, 1990.

[14] R. Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1–2):81–132, 1980.

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

[16] K. Satoh. Speculative computation and abduction for an autonomous agent. *IEICE Trans. Inf. & Syst.*, E88-D(9):2031–2038, 2005.

[17] K. Satoh, P. Codognet, and H. Hosobe. Speculative constraint processing in multi-agent systems. In *Proc. Pac. Rim Intl. Workshop on Multi-Agents (PRIMA2003)*, volume 2891 of *LNAI*, pages 133–144, 2003.

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

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

[20] G. Smolka. The Oz programming model. In *Computer Science Today: Recent Trends and Developments*, volume 1000 of *LNCS*, pages 324–343, 1995.

[21] R. J. Wallace and E. C. Freuder. Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artif. Intell.*, 161(1–2):209–227, 2005.

[22] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Knowl. Data Eng.*, 10(5):673–685, 1998.