Speculative Constraint Processing for Hierarchical Agents

Hiroshi Hosobe\textsuperscript{1}, Ken Satoh\textsuperscript{1}, Jiefei Ma\textsuperscript{2}, Alessandra Russo\textsuperscript{2}, and Krysia Broda\textsuperscript{2}

\textsuperscript{1} National Institute of Informatics, Japan \{hosobe,ksatoh\}@nii.ac.jp
\textsuperscript{2} Imperial College London, United Kingdom \{jm103,ar3,kb\}@doc.ic.ac.uk

Abstract. 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 information about agents even if communications between agents delayed or failed. 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.

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 insufficiently reliable network such as the Internet, or if a multi-agent system requires involvement of real human agents, communications might largely delay or even fail.

Speculative computation is an effective means for coping with such problems in multi-agent systems \cite{Belg14,Belg15,Belg16,Belg17}. It allows a multi-agent system to compute tentative solutions if communications between agents delayed or failed. This speculative computation is done by using default information about other agents instead of waiting for their answers.

Previously we have proposed a logical framework for speculative constraint processing for multi-agent systems \cite{Hos16}. 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., real human agents. 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 personalized agents also speculatively perform the management tasks of the corresponding human agents. It should be noted that such a system with multiple speculative computation agents cannot be modeled as a master-slave multi-agent system. Thus this extension marks an advance toward our ultimate goal that is to provide a powerful framework for speculative constraint processing. In this paper, we provide an operational model for the extended framework, and also 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. An example in the field of logic programming is the use of speculative parallelism in the Parlog language [2]. 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 [9], 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 [10]; 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 [7]. Inoue and Iwanuma proposed a different approach to speculative computation that used a consequence-finding procedure [3]. 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 [8]. In the framework, constraints facilitated the modeling of more general problems. Later we extended the constraint-based framework to support the revision of answers [1]. However, both of these frameworks were limited to master-slave multi-agent systems.

There have been studies on the extension of logic programming to multi-agent systems (e.g. [5]). Unlike those studies, our work is focused on speculative computation for 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 \(\text{root}(H)\) be the root node of \(H\), called the root agent. Let \(\text{int}(H)\) be the set of all the non-leaf nodes of \(H\), each called an internal agent. Let \(\text{ext}(H)\) be the set of all the leaf nodes of \(H\), each called an external agent. Given an internal agent \(M\), let \(\chi(M, H)\) be the set of all the child nodes of \(M\), each called a child agent of \(M\). Given a non-root agent \(S\), let \(\text{par}(S, H)\) be the parent node of \(S\), called the parent agent of \(S\).

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

Each internal agent is associated with a specification that is a constraint logic program [4] 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 asked of one of its children, and a non-askable atom as a knowledge that should be acquired from its program.

Definition 2 (askable/non-askable atom) Given an agent hierarchy \(H\) and an agent \(M \in \text{int}(H)\), 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 \(S \in \chi(M, H)\), \(p\) 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 \(\text{vars}(Q)\).

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

- \(\Delta_M\) 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.
P is a constraint logic program that is a set of rules R in the form

\[ H \leftarrow C \mid B_1, B_2, \ldots, B_n \]

where:
- H is a non-askable atom called the head of R and written head(R);
- C is a set of constraints written 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 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 \( \langle H, F \rangle \)
where H is an agent hierarchy, and \( F \) is a set of specifications \( F_M = \{ \langle \Delta_M, P_M \rangle \}_{M \in \text{int}(H)} \).

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

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

- \( F_r = \langle \Delta_r, P_r \rangle \) where
  - \( \Delta_r \) contains the following default rules:
    \[ \text{available}(D) @ a \leftarrow D \in \{1, 2, 3\} \mid \]
    \[ \text{available}(D) @ b \leftarrow D \in \{1, 2, 3\} \mid \]
  - \( P_r \) is the following constraint logic program:
    \[ \text{reserve}(R, L, D) @ r \leftarrow R = \text{twin\_room}, L = [a, b] \mid \]
    \[ \text{available}(D) @ a, \text{available}(D) @ b \]
    \[ \text{reserve}(R, L, D) @ r \leftarrow R = \text{single\_room}, L = [a] \mid \]
    \[ \text{available}(D) @ a, \text{unavailable}(D) @ b \]
    \[ \text{reserve}(R, L, D) @ r \leftarrow R = \text{single\_room}, L = [b] \mid \]
    \[ \text{unavailable}(D) @ a, \text{available}(D) @ b \]

- \( F_a = \langle \Delta_a, P_a \rangle \) where
  - \( \Delta_a \) contains the following default rules:
    \[ \text{free}(D) @ a' \leftarrow D \in \{1, 2\} \mid \]
    \[ \text{busy}(D) @ a' \leftarrow D \in \{3\} \mid \]
  - \( P_a \) is the following constraint logic program:
    \[ \text{available}(D) @ a \leftarrow \mid \text{free}(D) @ a' \]
    \[ \text{unavailable}(D) @ a \leftarrow \mid \text{busy}(D) @ a' \]

- \( F_b = \langle \Delta_b, P_b \rangle \) where
  - \( \Delta_b \) contains the following default rules:
    \[ \text{free}(D) @ b' \leftarrow D \in \{2\} \mid \]
  - \( P_b \) is the following constraint logic program:
    \[ \text{available}(D) @ b \leftarrow \mid \text{free}(D) @ b' \]
    \[ \text{unavailable}(D) @ b \leftarrow \mid \text{busy}(D) @ b' \]

Then \( \langle H, F \rangle \) is a multi-agent system.
In this example, there are two human agents 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 [1], which is based on the CLP scheme [4].

We first define a goal that is a question to ask of a multi-agent system.

**Definition 5 (goal)** Let $\langle H, F \rangle$ 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 $\langle H, F \rangle$ be a multi-agent system with $F = \{ (\Delta_M, \mathcal{P}_M) \}_{M \in \text{int}(H)}$. Let $\mathcal{A}_{\text{ext}}$ be a set of most recent answers of the external agents, each of which is a rule “$Q@S \leftarrow C ||$” with $S \in \text{ext}(H)$. The belief state of $\langle H, F \rangle$ w.r.t. $\mathcal{A}_{\text{ext}}$, written $\text{bel}(\mathcal{A}_{\text{ext}}, \langle H, F \rangle)$, is

$$\text{bel}(\mathcal{A}_{\text{ext}}, \langle H, F \rangle) = \mathcal{A}_{\text{ext}} \cup \{ "Q@S \leftarrow C ||" \mid S \in \text{ext}(H) \land "Q@S \leftarrow C ||" \in \Delta_{\text{par}(S,H)} \land \exists C, "Q@S \leftarrow C ||" \in \mathcal{A}_{\text{ext}} \}.$$

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 $\langle H, F \rangle$ be a multi-agent system with $F = \{ (\Delta_M, \mathcal{P}_M) \}_{M \in \text{int}(H)}$, and $\mathcal{A}_{\text{ext}}$ 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. $\langle H, F \rangle$, $\mathcal{A}_{\text{ext}}$, and $B_i$ is a goal “$\leftarrow C" || GS"$” such that:

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

---

3 If $B_i$ is unifiable with \text{head}(R), $B_i = \text{head}(R)$ represents the conjunction of the constraints that equate the arguments of $B_i$ with those of \text{head}(R); otherwise, $B_i = \text{head}(R)$ represents false.
**Definition 8 (derivation)** Let \( \langle H, \mathcal{F} \rangle \) be a multi-agent system with \( \mathcal{F} = \{ (\Delta_M, \mathcal{P}_M) \} \) and \( M_{\text{root}} = \text{root}(H) \), and \( \mathcal{A}_{\text{ext}} \) be a set of most recent answers of the external agents. A derivation of a goal \( G = "j Q \text{@init}@M_{\text{root}}" \) w.r.t. \( \langle H, \mathcal{F} \rangle \) and \( \mathcal{A}_{\text{ext}} \) is a sequence of reductions \( "j Q \text{@init}@M_{\text{root}}", \ldots, "C j Q" \) w.r.t. \( \langle H, \mathcal{F} \rangle \) and \( \mathcal{A}_{\text{ext}} \), where an atom in the body of the current goal is selected in each reduction. \( C \) is called an answer w.r.t. \( H, \mathcal{F}, \mathcal{A}_{\text{ext}}, \) and \( G \).

### 4 Operational Model

This section provides the operational model of hierarchical multi-agent systems defined in the previous section.

#### 4.1 Data Structures

We first 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 \( \{ o_s, o_n \} \cup \{ d_1, d_2, \ldots \} \cup \{ p_1, p_2, \ldots \} \), where \( \{ d_1, d_2, \ldots \} \) is a countably infinite set, and \( \{ o_s, o_n \} \), \( \{ d_1, d_2, \ldots \} \), and \( \{ p_1, p_2, \ldots \} \) are disjoint.

A **labeled askable atom** is a pair \( \langle Q @ S, o_s \rangle, \langle Q @ S, o_n \rangle, \langle Q @ S, d_i \rangle, \text{ or } \langle Q @ S, p_i \rangle \), where \( Q @ S \) is an askable atom, and \( o_s, o_n, d_i, \text{ and } p_i \) 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 \( M \in \text{int}(H) \), an answer to \( M \) is a quadruple \( \langle Q @ S, AID, C, AID_{\text{prev}} \rangle \), where \( S \in \text{chi}(M, H) \), \( Q @ S \) is an askable atom, \( AID \) is an answer identifier, \( C \) is a set of constraints, and \( AID_{\text{prev}} \) is either \( \text{nil} \) or a different answer identifier from \( AID \). If \( AID_{\text{prev}} = \text{nil} \), this answer is called a **new answer**; otherwise, it is called a **revised answer**.

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 \( M \in \text{int}(H) \), a process \( P \) of \( M \) is a quintuple \( \langle \text{pid}(P), C, GS, WA, AA \rangle \), where \( \text{pid}(P) \) is a process identifier written \( \text{pid}(P) \), \( C \) is a set of constraints written \( \text{pconst}(P) \), \( GS \) is a set of atoms written \( \text{gs}(P) \), \( WA \) and \( AA \) are sets of labeled askable atoms written \( \text{wa}(P) \) and \( \text{aa}(P) \) respectively. If \( \text{wa}(P) \neq \emptyset \), \( P \) is said to be **suspended**.

An **answer entry** is a data structure that keeps an answer and the identifiers of the processes using the answer.
Definition 11 (answer entry) Given a multi-agent system $\langle H, \mathcal{F} \rangle$ and an agent $M \in \text{int}(H)$, an answer entry $A$ for $M$ is a quadruple $\langle Q@S, \text{AID}, \text{C}, \text{UPS} \rangle$, where $S \in \chi(M, H)$, $Q@S$ is an askable atom written $aq(A)$, $\text{AID}$ is an answer identifier written $\text{aid}(A)$, $\text{C}$ is a set of constraints written $\text{aconst}(A)$, and $\text{UPS}$ is a set of process identifiers written $\text{ups}(A)$. If $\text{aid}(A)$ is either $o_s$ or $o_n$, $A$ is called an original answer entry; if $\text{aid}(A)$ 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.

4.2 Procedure

Now we present the procedure of the operational model. The main routine shown in Figure 1 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 the fact arrival phase and the process reduction phase respectively.

The process reduction phase presented in Figure 2 treats a non-suspended 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 3 updates the related answer entries and processes when the agent $M$ receives an answer $\langle Q@S, \text{AID}, \text{C}_r, \text{AID}_{\text{prev}} \rangle$ 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 $\text{C}_r$ 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 $\text{C}_r$ to the processes using the previous answer whose identifier is $\text{AID}_{\text{prev}}$.

4.3 Correctness

We show the correctness of the operational model presented in the previous subsection. For this purpose, we first present a lemma that holds for local parts of hierarchical multi-agent systems.

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

**Lemma 1** For any multi-agent system \( \langle H, \mathcal{F} \rangle \), any agent \( M \in \text{int}(H) \), and any ordinary/finished process \( P \) of \( M \) whose initial goal is \( \leftarrow || Q_{\text{init}} \circ M \)”, there exists a sequence of reductions \( \leftarrow || Q_{\text{init}} \circ M \), \ldots, \( \leftarrow || Q_{\text{init}} \circ M \)” w.r.t. msres\( (M, \langle H, \mathcal{F} \rangle) \) and \( A_P \) such that \( \pi_{\text{vars}(Q_{\text{init}})}(\text{pconst}(P)) \) entails \( \pi_{\text{vars}(Q_{\text{init}})}(C) \), where

\[ A_P = \{ "Q \circ S \leftarrow C' || \" | there exists an answer entry \( \langle Q \circ S, AID, C', UPS \rangle \) for \( M \) such that \( \langle Q \circ S, AID \rangle \in \text{aa}(P) \}\} \]

**Proof:** Similar to the proof of Theorem 1 in [1]. \( \square \)

We prove that multi-agent systems based on the operational model eventually compute correct solutions in the sense of the semantics.

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

- for any agent \( M \in \text{int}(H) \), there exists no ordinary process \( P \) of \( M \) such that \( \text{wa}(P) = \emptyset \);
- for any agent \( M \in \text{int}(H) \) and any finished process \( P \) of \( M \) such that \( \text{wa}(P) = \emptyset \), there exists no default answer entry \( A_d \) for \( M \) such that \( \langle Q \circ S, \text{aid}(A_d) \rangle \in \text{aa}(P) \) and \( S \in \text{int}(H) \), where \( Q = \text{aq}(A_d) \);
- for any agent \( S \in \text{int}(H) \setminus \{ \text{root}(H) \} \), there exists an ordinary answer entry \( \langle Q \circ S, AID, C, UPS \rangle \) for \( \text{par}(S) \) if and only if there exists a finished process \( P \) of \( S \) such that \( \text{wa}(P) = \emptyset \), \( AID = \text{pid}(P) \), \( C = \text{pconst}(P) \), and \( \leftarrow || Q \circ S \) is the initial goal of \( P \);
if $gs(P) = \emptyset$ then
  change $P$ into a finished process with the same data;
  answer $(Q_{init} \odot M, pid(P), pconst(P), nil)$ to the parent;
else
  select an atom $L$ from $gs(P)$;
  if $L$ is a non-askable atom then
    foreach rule $R \in Pr$ do
      $C = pconst(P) \land (L = \text{head}(R)) \land const(R)$;
      if $C$ is consistent then
        $PID \leftarrow$ a new process ID; $GS \leftarrow$ $\text{body}(R) \cup gs(P) \setminus \{L\}$;
        create an ordinary process ($PID, C, GS, \emptyset, aa(P)$);
      foreach answer entry $A$ s.t. $(aq(A), aid(A)) \in aa(P)$ do
        $ups(A) \leftarrow ups(A) \cup \{PID\}$;
      end
    foreach answer entry $A$ s.t. $(aq(A), aid(A)) \in aa(P)$ do
      $ups(A) \leftarrow ups(A) \setminus \{pid(P)\}$;
      kill $P$;
  else // $L$ is an askable atom $Q@S$.
    $Q@S \leftarrow L$;
    $AS_r \leftarrow \{A_r \mid A_r$ is an ordinary answer entry s.t. $aq(A_r) = Q@S\}$;
    if $AS_r \neq \emptyset$ then
      $AS \leftarrow AS_r$; $AID_o \leftarrow o_i$;
      foreach answer entry $A$ s.t. $aq(A) = Q@S$ do
        $C = pconst(P) \land \text{aconst}(A)$;
        if $C$ is consistent then
          $PID \leftarrow$ a new process ID; $GS \leftarrow gs(P) \setminus \{Q@S\}$;
          $AA \leftarrow aa(P) \cup \{(Q@S, aid(A))\}$;
          create an ordinary process ($PID, C, GS, \emptyset, AA$);
          $ups(A) \leftarrow ups(A) \cup \{PID\}$;
          foreach answer entry $A'$ s.t. $(aq(A'), aid(A')) \in aa(P)$ do
            $ups(A') \leftarrow ups(A') \cup \{PID\}$;
          end
        end
      end
    else // $L$ is a variable in root($H$).
      $Q@S \leftarrow L$;
      $AS_r \leftarrow \{A_r \mid A_r$ is an ordinary answer entry s.t. $aq(A_r) = Q@S\}$;
      if $AS_r \neq \emptyset$ then
        $AS \leftarrow AS_r$; $AID_o \leftarrow o_i$;
        foreach answer entry $A$ s.t. $aq(A) = Q@S$ do
          $C = pconst(P) \land \text{aconst}(A)$;
          if $C$ is consistent then
            $PID \leftarrow$ a new process ID; $GS \leftarrow gs(P) \setminus \{Q@S\}$;
            $AA \leftarrow aa(P) \cup \{(Q@S, aid(A))\}$;
            create an ordinary process ($PID, C, GS, \emptyset, AA$);
            $ups(A) \leftarrow ups(A) \cup \{PID\}$;
          end
        end
      end
    end
  end
end

Fig. 2. Process reduction phase for a non-suspended ordinary process $P$.

- for any agent $S \in \text{ext}(H)$, there exists an ordinary answer entry $(Q@S, AID, C, UPS)$ for $par(S)$ if and only if there exists a most recent answer $\overline{Q@S \leftarrow C}$ of $S$.

**Theorem 1** Let $(H,F)$ be an arbitrary multi-agent system with $M_{root} = \text{root}(H)$, $\overline{\mid Q_{init} \odot M_{root} \mid}$ be an arbitrary goal, and $A_{ext}$ be an arbitrary set of most recent answers of the external agents. Assume that the operation of $(H,F)$ is hierarchically stable. Then, for any finished process $P$ of $M_{root}$ such that $wa(P) = \emptyset$, there exists a derivation $\overline{\mid Q_{init} \odot M_{root} \mid, \ldots, \overline{\mid C \mid}}$ w.r.t. $(H,F)$ and $A_{ext}$ such that $\pi_{vars(Q_{init})}(pconst(P))$ entails $\pi_{vars(Q_{init})}(C)$.

**Proof:** We prove this theorem by induction on the tree structure of $(H,F)$. For this purpose, we introduce hierarchical restrictions of $(H,F)$. Let $F = \{(\Delta_M, \mathcal{P}_M')\}_{M' \in \text{int}(M)}$. The hierarchical restriction of $(H,F)$ to an agent $M \in \text{int}(H)$, written $\text{hres}(M, (H,F))$, is defined as the multi-agent system $(H_M, F_M)$, where $H_M$ is the agent hierarchy consisting of $M$ as the root and all the descendant agents of $M$ in $H$, and $F_M = \{(\Delta_M, \mathcal{P}_M')\}_{M' \in \text{int}(H_M)}$. 


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 $(\ast)$ that, if the operation of $\langle H, F \rangle$ is hierarchically stable, for any $M \in \text{int}(H)$ and any finished process $P_M$ of $M$ such that $\text{wa}(P_M) = \emptyset$, there exists a derivation "$\lhd \text{qresh}M", \ldots, \text{qresh}C_{P_M} \rhd$" w.r.t. $\langle H_M, F_M \rangle = \text{hres}(M, \langle H, F \rangle)$ and $A_\text{ext}$ such that $\pi_{\text{vars}(Q_{P_M})}(\text{pconst}(P_M))$ entails $\pi_{\text{vars}(Q_{P_M})}(C_{P_M})$.

**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, $(\ast)$ holds by Lemma 1.

**Induction step.** Assume that, for any $M \in \text{int}(H)$ such that $\text{height}(H_M) < n$, $(\ast)$ holds. Consider any $M \in \text{int}(H)$ such that $\text{height}(H_M) = n$. Assume that the
operation of \((H,F)\) is hierarchically stable, and let \(P_M\) be an arbitrary finished process of \(M\) such that \(wa(P_M) = \emptyset\). Then, for any \(\langle Q@S,AID\rangle \in aa(P_M)\), we have the following.

- **Case \(S \in ext(H)\).** There exists an answer entry \(\langle Q@S,AID,C_S,UPS\rangle\) for \(M\). If it is an ordinary answer entry, there exists a most recent answer “\(Q@S \leftarrow C_S\)” in \(A_{\text{ext}}\); otherwise, it is a default answer entry, and there exists a default rule “\(Q@S \leftarrow C_S\)” in \(\Delta_M\). Thus “\(Q@S \leftarrow C_S\)” is in \(\text{bel}(A_{\text{ext}}, \langle H_M, F_M \rangle)\).

- **Case \(S \in int(H)\).** There exists an ordinary answer entry \(\langle Q@S,AID,C_S,UPS\rangle\) for \(M\) and a finished process \(P_S\) of \(S\) such that \(wa(P_S) = \emptyset\), \(AID = \text{pid}(P_S)\), \(C_S = \text{pconst}(P_S)\), and “\(\leftarrow ||Q@S\)” is the initial goal of \(P_S\). By the induction hypothesis, there exists a derivation “\(\leftarrow ||Q@S||, \ldots, \leftarrow C_{P_S}||\)” w.r.t. \(\text{hres}(S, \langle H,F \rangle)\) and \(A_{\text{ext}}\) such that \(\pi_{\text{vars}(Q)}(C_S)\) entails \(\pi_{\text{vars}(Q)}(C_{P_S})\).

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

5 Implementation

Using the operational model proposed in the previous section, we have developed a prototype system for speculative constraint processing for hierarchical multi-agent systems. Our current implementation is written in the Objective Caml\(^4\) 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.\(^5\)

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.

\(^4\) [http://caml.inria.fr/ocaml/](http://caml.inria.fr/ocaml/)

\(^5\) 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.
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.

We start with a goal “reserve(R, L, D)@r” by asking it of the root agent r, and then obtain the following state of r.

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
Receiveable 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 asks available(D) of its child agent a.

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

Answer entries:

6 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.

7 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.
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}, {}, {(av(D)@a, d(1))})

Finished processes:

After further two reductions (during which agent $r$ asks available($D$) of agent $b$), the system yields the following.

Agent $r$ returned to Root_caller the new answer:
(rsv(R,L,D)@r, r(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 $R = \text{twin\_room}$, $L = [a,b]$, and $D \in \{1,2,3\}$. Note that it is a tentative answer speculatively computed from the default rules of $r$.

Since agent $r$ asked available($D$) of 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, r(3), \{D@r#2:{1,2}\}, nil)

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

Agent $r$ returned to Root_caller the revised answer:
(rsv(R,L,D)@r, r(7), {R=tr, L=[a,b], D:{1,2}}, r(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, r(3), \{D@r#2:{2}\}, nil)

Then agent $r$ returns a further revised answer $R = \text{twin\_room}$, $L = [a,b]$, and $D \in \{2\}$. 
Agent r returned to Root_caller the revised answer:
(rsv(R,L,D)@r, r(9), {R=tr, L=[a,b], D:{2}}, r(7))

Next, switch our attention to the ordinary process 4 of agent r. Its reduction causes r to ask unavailable(D) of agent a.

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

Then agent a returns an answer that it computes from its default rule “busy(D)@a’ ← D ∈ {3}”.

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

Next, suppose that the external agent b returns an answer “free(D)@b ← D ∈ {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, r(5), {D@r#2:{3}}, nil)

Then r returns an answer R = single_room, L = [b], and D ∈ {3}.

Agent r returned to Root_caller the new answer:
(rsv(R,L,D)@r, r(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 = twin_room, L = [a,b], and D ∈ {2} is still valid.

7 Discussion

Speculative constraint processing requires appropriate default rules to obtain good results based on speculative computation. 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 [9, 10] that allow only yes/no questions. However, speculative constraint processing currently does not support negation as failure 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 as failure.
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 [6]. We will extend the existing multi-threaded implementation to cover hierarchical multi-agent systems. We are also interested in supporting negation as failure in speculative constraint processing, since it is useful for modeling various problems as discussed in Section 7.

Acknowledgement

This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology, Japan, Grant-in-Aid for Scientific Research (B), 19300053.

References