Speculative Constraint Processing with Iterative Revision for Disjunctive Answers

Martine Ceberio[1], Hiroshi Hosobe[2], and Ken Satoh[2]

[1] University of Texas at El Paso
500 West University Avenue, El Paso, Texas 79968-0518, USA
mceberio@cs.utep.edu
[2] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
{ ksatoh, hosobe }@nii.ac.jp

Abstract. In multi-agents systems, incompleteness, due to either communication failure or response delay, is a major problem to handle. To face incompleteness, frameworks for speculative computation were proposed (see [5, 6, 4]). The idea developed in such frameworks is to allow the asking agent, while waiting for slave agents to reply, to reason using default belief until replies are sent.
In [6] in particular, a framework is proposed, that allows an agent not only to perform speculative computation but also to accept iterative answer revision, in the case of yes/no questions. In this paper, we present an extension of the framework in the case of more general types of questions using constraint logic programming (CLP).

1 Introduction

Multi-agent systems are very fashionable and convenient, for they make it possible, for instance, to take advantage of multi-processor machines, and for they also make it possible to design human-like efficient organizations of agents. The main limitation to such an approach is that, as arises in human organizations, communication may be an issue: delayed or broken, it leads to incompleteness of the information in the reasoning structure.

This is a concrete concern when we consider distributed systems such as the Internet, in which communication is indeed not guaranteed, and even if we could guarantee it, communication may either take time, or agents themselves may delay their sending information.

In the case of such unideal, but as we believe, practical situations, when problem-solving is at stake, frameworks for speculative computations were proposed: first for yes/no questions only [5], and then for general questions [4] using constraints.

In [5] and [4], they only provide the possibility for the master agent to perform speculations and a returned answer from the slave agent is final and there is no possibility of change of answers. However, if we let every agent perform speculative computation, the asked agent may revise his answer since the previous answer sometimes depends on the asked agent's belief, which might turn out to be false. Therefore, a chain reaction of belief revision among agents might occur which was firstly observed in [6], and Satoh and Yamamoto provide a revisable speculative computation method for yes/no questions. Essential part of their work is a dynamic iterative belief revision mechanism which can handle a revision of an answer for query even during the execution.

Belief revision is indeed very important for both the sake of flexibility (information is processed before it is complete), and speed of computation (time is saved in case prior information is later entailed).

In this paper, we combine the methods proposed in [4] and [6], and extend them, so that we can handle iterative answer revision for a query with constraints. We also complete these methods with the ability to corporate disjunctive answers. So, the main contribution of this paper is the definition of a framework that enables to perform speculative computations on constraints while handling belief revision, and that handles as well disjunctive answers. In particular, the main challenges dealt with in this work are the following.

– First, processing speculative constraints, as shown in [4], is manageable when belief revision is not considered. In this paper, belief revision is made possible because it enables more speculative computation in multi-agent systems. This hardens the problem a lot: the process management needs to be modified so as to enable changes in the computation at any time, while maintaining a reasonable balance between not being too much space-consuming, and not loosing too much time (*i.e.*, we don't want to start from scratch all the time). The process management is presented in detail in this paper, as well as results on the space complexity of our operational model.

– The second challenging point described in this paper is the way disjunction is now handled in the framework we propose. Indeed, considering the situation where each agent's behavior is specified as a CLP program, we need to handle alternative answers, since these answers may come from different derivations in CLP. By manipulating such alternative answers, we face another complication, in that we need to distinguish a revised answer of a previous answer, from an answer derived from an alternative derivation path[3]. To solve this problem, we devise an answer entry which keeps track of the usage status of the answer in processes. This new feature impacts the way processes are managed, as described in Section 3, and therefore makes the problem more complicated.

For an iterative belief revision, many proposals have been described. As far as we know, existing frameworks separate reasoning and belief revision, except [5, 6, 4]. And this work is along the line of the works of Satoh et al. in a more general setting.

There are works on a formalization of an agent in terms of logic programming such as [3]. Although these research are important in their own right, our paper

[3] Indeed, in particular, a contradictory answer should be considered as contradictory only if it is a revision of a former answer, not if it is an alternative answer.

pursues another branch of investigation in the context of speculative computation.

Most related research would be constraint programming language such as AKL(Andorra Kernel Language) [2] and Oz [7] which perform a kind of speculative computation. AKL allows local speculative variable bindings in a guard of each clauses until one of guards is succeeded and Oz can control multiple computation spaces each of which represents alternative path of constraint processing. As far as we understand, however, speculative computation used in these languages are mainly motivated for or-parallel computing where all alternative paths of computation are executed in parallel until one of paths are succeeded eventually. On the other hand, we regard a speculative computation as a default computation where most plausible paths of computation are executed. Moreover, they do not consider any revision of the answers.

The structure of the paper is as follows. We firstly define a framework for speculative constraint processing and a semantics of the framework. Then, we describe an operational model and show an example of execution and state correctness of our model. Finally, we discuss space complexity issues, before to conclude.

2 Speculative Constraint Processing

In this section, we provide a framework of speculative constraint computation based on the CLP framework [1]. This framework is designed so that an agent not only performs speculative constraint processing but also accepts revised answers and alternative answers. We then define a semantics of this framework, in Subsection 2.2.

2.1 Framework Definition

Definition 1. *Let Σ be a finite set of constants. We call an element in Σ a* slave agent identifier*. An* atom *is of the form either $p(t_1, ..., t_n)$ or $p(t_1, ..., t_n)@S$ where p is a predicate, $t_i (1 \leq i \leq n)$ is a term, and S is in Σ.*

We call an atom with an agent identifier an "*askable atom*", and an atom without an identifier a "*non-askable atom*".

Definition 2. *A* framework for speculative constraint computation, in a master-slave system *is a triple $\langle \Sigma, \Delta, \mathcal{P} \rangle$ where:*

- *Σ is a finite set of constants;*
- *Δ is a set of rules of the following form called* default rule *w.r.t. $Q@S$:*

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

where $Q@S$ is an askable atom, each of whose arguments is a variable, and C is a set of constraints, called default constraint for *$Q@S$;*

– *$\mathcal{P}$ is a constraint logic program, that is, a set of rules of the form:*

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

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

Note that a default is not necessarily specified for every askable atom. Moreover, we allow multiple defaults for the same askable atom.

Example 1. We consider the following example of hotel room reservation. There is a master agent m: m asks travellers a and b. If both travel, m reserves a twin room. If one of them travels, m reserves a single room. Agent m has default information about the status of a and b for days 1, 2 and 3, but the real status will be obtained directly from a and b, and the status is therefore likely to be changed.

This example can be represented as the following multi-agent system $\langle \Sigma, \Delta, \mathcal{P} \rangle$[4]:

– Σ is the set of slave agents. Here, there is one master agent, m, and two slave agents, a and b. Therefore $\Sigma = \{a, b\}$.

– Δ is the set of default information (default rules), assumed by the master agent. In particular, let us suppose that m assume that a is free on days 1, and 2, and busy on day 3, and that b is free on day 2, and busy on day 1. Then the corresponding set Δ is as follows:

$$\begin{aligned} \Delta = \{ \; & d_1: \; fr(D)@a \leftarrow D=1 \|., \\ & d_2: \; fr(D)@a \leftarrow D=2 \|., \quad d_3: \; bs(D)@a \leftarrow D=3 \|., \\ & d_4: \; fr(D)@b \leftarrow D=2 \|., \quad d_5: \; bs(D)@b \leftarrow D=1 \|. \} \end{aligned}$$

Let us remark that it is not necessary that a default information exist for all cases. In particular, m has no default information concerning the status of b on day 3.

– $\mathcal{P}$ is a constraint logic program, to be solved by agent m. In our case of hotel room reservation with two travelers, it is made of the following set of rules:

$$\begin{aligned} rsv(R, L, D) &\leftarrow R=tr, L=[a, b] \| fr(D)@a, fr(D)@b. \\ rsv(R, L, D) &\leftarrow R=sr, L=[a] \| fr(D)@a, bs(D)@b. \\ rsv(R, L, D) &\leftarrow R=sr, L=[b] \| bs(D)@a, fr(D)@b. \end{aligned}$$

In order to solve this constraint satisfaction problem, agent m will have to ask agents a and b about $fr(D)@a$, $bs(D)@a$, $fr(D)@b$, $bs(D)@b$.

[4] A string beginning with an upper case letter represents a variable and a string beginning with a lower case letter represents a constant. We abbreviate "free" as fr, "busy" as bs, "travel" as $trvl$, "reserve" as rsv, "twin room" as tr, and "single room" as sr.

2.2 Semantics of Speculative Constraint Processing

For a semantics of the above framework, we index the semantics of constraint logic program by a *reply set* which specifies a reply for an askable atom.

Definition 3. *A* reply set *is a set of rules of the form:*

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

where $Q@S$ is an askable atom, each of whose arguments is a variable, and C is a constraint over these variables.

Let $\langle \Sigma, \Delta, \mathcal{P} \rangle$ be a framework for speculative constraint computation, and $\mathcal{R}$ be a reply set. A belief state *w.r.t. $\mathcal{R}$ and Δ is a reply set defined as:*

$$\mathcal{R} \cup \{\text{“}Q@S \leftarrow C\|\text{”} \in \Delta \mid \neg\exists\, C' \ s.t.\ \text{“}Q@S \leftarrow C'\|\text{”} \in \mathcal{R}\}$$

and denoted as $BEL(\mathcal{R}, \Delta)$.

We introduce the above belief state, since if the answer is not returned, we use a default rule for an unreplied askable atom.

Definition 4. *A goal is of the form $\leftarrow C\|B_1, ..., B_n$ where C is a set of constraints and B_i's are atoms. We call C the* constraint of the goal *and $B_1, ..., B_n$ the* body of the goal.

Definition 5. *A reduction of a goal $\leftarrow C\|B_1, ..., B_n$ w.r.t. a constraint logic program $\mathcal{P}$, a reply set $\mathcal{R}$ and an atom B_i, is a goal $\leftarrow C'\|B'$ such that:*

- *there is a rule R in $\mathcal{P} \cup \mathcal{R}$ s.t. $C \wedge (B_i = head(R)) \wedge const(R)$ is consistent*[5].
- *$C' = C \wedge (B_i = head(R)) \wedge const(R)$*
- *$B' = \{B_1, ...B_{i-1}, B_{i+1}, ..., B_n\} \cup body(R)$*

Definition 6. *A derivation of a goal $G = \leftarrow C\|Bs$ w.r.t. a framework for speculative constraint computation $\mathcal{F} = \langle \Sigma, \Delta, \mathcal{P} \rangle$ and a reply set $\mathcal{R}$ is a sequence of reductions “$\leftarrow C\|Bs$”,...,“$\leftarrow C'\|\emptyset$”*[6] *w.r.t. $\mathcal{P}$ and $BEL(\mathcal{R}, \Delta)$ where in each reduction step, an atom in the body of the goal in each step is selected. C' is called an* answer constraint *w.r.t. G, $\mathcal{F}$ and $\mathcal{R}$. We call a set of all answer constraints w.r.t. G, $\mathcal{F}$ and $\mathcal{R}$ the* semantics of G *w.r.t. $\mathcal{F}$ and $\mathcal{R}$.*

In the above definition, we only consider the most recent reply set, whereas a reply set might be varied during execution according to the slave agent's answer revision. We use the most recent reply set because it reflects the current situation of the slave agents.

[5] A notation $B_i = head(R)$ represents a conjunction of constraints equating the arguments of atoms B_i and $head(R)$.
[6] $\emptyset$ denotes an empty goal.

3 An Operational Model for Speculative Computation with Iterative Answer Revision

3.1 Overview of Operational Model

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

For the operational model, we use the following two kinds of objects: a *process* and an *answer entry*.

Each *process* represents an alternative way of computation. Processes are created when a choice point of computation is encountered, such as case splitting, default handling and answer arrival. A process becomes a finished process when the body of the associated goal with the process becomes empty. A process fails when some used default constraints are found to contradict the newly returned answer.

An *answer entry* is used to distinguish alternative answers and to detect which old answer corresponds to the newly revised answer. This detection is done by attaching an ID to each answer. If a new answer with an ID different from any existing answer comes, it is an alternative answer. Otherwise, the new answer is considered as a revised answer for the old answer with the same ID.

Figures 1∼4 intuitively explain how processes are updated according to askable atoms. In the tree, each node represents a process, but we only show constraints associated with the process. The top node represents a constraint for the original process, and the other nodes represent added constraints for the reduced processes. The leaves of the process tree represent the current processes. Therefore, the processes which are not in the leaves are deleted processes.

Fig. 1 shows a situation of the processes represented as a tree when an askable atom, whose reply has not arrived yet, is executed in the process reduction phase. In this case, the current process, represented by the processed constraints C, is splitted into two different kinds of processes: the first one is a process using default information, C_d, and is called *default process* [7]; and the other one is the current process C itself, called *original process*, suspended at this point.

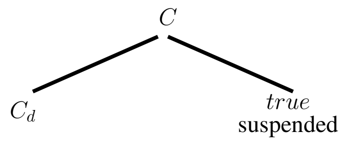

Fig. 1. When $Q@S$ is processed, during the process reduction phase

Note that, if there are multiple definitions of defaults, we will have more than one default process, but still only one suspended process. In addition, let us note

[7] In this figure, we assume that there is only one default for brevity.

that the reason for suspending processes (which is, keeping them in memory), is that in case of a contradictory revision of the default, or later alternative answers coming, it is essential to keep memory of the original processes to be able to restore them.

When, after some reduction of the default processes (represented on Fig. 2 by dashed lines), the first answer comes from a slave agent, expressing constraint C_f for this askable literal, we update default processes as well as the original suspended process as follows:

– Default process(es) are reduced into two different kinds of processes: the first kind is a process adding C_f to the problem to solve, and the other is the current process itself which is suspended at this point[8].
– The original process is reduced into two different kinds of processes as well: the first kind is a process adding $\neg C_d \wedge C_f$, and the other is the original process, suspended at this point.

Let us remark that although the tree of processes grows, only leaves are kept in memory.

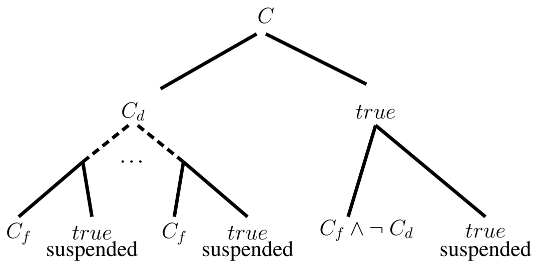

Fig. 2. When the first answer C_f for $Q@S$ arrives

To explain the correctness of the above process update intuitively, we define a *frontier* which represents the computation status of all alternative derivations. A *frontier* w.r.t. a goal $\leftarrow C \| Bs$, a framework for speculative constraint computation $\langle \Sigma, \Delta, \mathcal{P} \rangle$ and a reply set $\mathcal{R}$, is a set of goals defined as follows.

1. The set consisting of the initial goal, $\{\leftarrow C \| Bs\}$ is a frontier.
2. Let F be a frontier w.r.t. the above initial goal, the framework and the reply set. If a goal G is in F, B is an atom in G, and $RGs = \{G' | \ G'$ is a reduction of G w.r.t. $\mathcal{P}$, $BEL(\mathcal{R}, \Delta)$ and $B\}$, then $F \backslash \{G\} \cup RGs$ is a frontier.

Then we have the following properties.

Lemma 1. *Let $\leftarrow C \| Bs$ be a goal, F be a frontier of this goal, and C' be a constraint. If we add C' to the constraints of every goal in F, then the disjunctions of all answer constraints of these modified goals is logically equivalent to the disjunction of all answer constraints of the goal $\leftarrow C \wedge C' \| Bs$.*

[8] Let us remark that this splitting process is similar to the splitting process above-described for the case of a first default used.

Lemma 2. *Let $\leftarrow C \| Bs$ be a goal, $\mathcal{R}$ be a reply set, and C' be a constraint. Then, the disjunction of answer constraints of $\leftarrow C \wedge C' \| Bs$ and $\leftarrow C \wedge \neg C' \| Bs$ is logically equivalent to the disjunction of all answer constraints of $\leftarrow C \| Bs$.*

Let $\leftarrow C \| Bs$ be a goal containing $Q@S$, suppose that it is reduced into $\leftarrow C \wedge C_d \| Bs \backslash \{Q@S\}$ by a default rule "$Q@S \leftarrow C_d \|$". Let F be a frontier of $\leftarrow C \wedge C_d \| Bs \backslash \{Q@S\}$ when the first reply "$Q@S \leftarrow C_f \|$" is returned. Since our semantics considers the most recent replies, at this point, we should consider:

$$\leftarrow C \wedge C_f \| Bs \backslash \{Q@S\}$$

instead of:

$$\leftarrow C \wedge C_d \| Bs \backslash \{Q@S\}.$$

One possibility to implement this change is that we just discard F and invoke a new goal $\leftarrow C \wedge C_f \| Bs \backslash \{Q@S\}$. However, in this case, we throw every computation away before F is obtained. To retain the previous computation as much as possible, we propose the following execution.

1. We add C_f to the constraint of every goal in F.
   Let us remark that the disjunction of all answer constraints from this new frontier is logically equivalent to the disjunction of all answer constraints of $\leftarrow C \wedge C_d \wedge C_f \| Bs \backslash \{Q@S\}$ as Lemma 1 states. This computation keeps the previous computation which is consistent with the new reply (C_f).

2. In addition to the above computation, we also start computing a new goal:

$$\leftarrow C \wedge \neg C_d \wedge C_f \| Bs \backslash \{Q@S\}$$

   to guarantee completeness. It is because the disjunction of all answer constraints derived from $\leftarrow C \wedge C_d \wedge C_f \| Bs \backslash \{Q@S\}$ and $\leftarrow C \wedge \neg C_d \wedge C_f \| Bs \backslash \{Q@S\}$ is logically equivalent to the disjunction of all answer constraints derived from $\leftarrow C \wedge C_f \| Bs \backslash \{Q@S\}$ as Lemma 2 states.

When an alternative answer, with the constraint C_a, comes from a slave agent (as shown on Fig. 3), we need to follow the same procedure as when the first answer comes (*cf.* Fig. 2), except that now the processes handling only default information are suspended. So, this is done by splitting the suspended default process(es), in order to obtain the answer constraints which are logically equivalent to the answer constraints of:

$$\leftarrow C \wedge C_d \wedge C_a \| Bs \backslash \{Q@S\},$$

as well as by splitting the suspended original process, in order to obtain the answer constraints which are logically equivalent to the answer constraints of $\leftarrow C \wedge \neg C_d \wedge C_a \| Bs \backslash \{Q@S\}$ (Fig. 3). By gathering these answer constraints, we can compute all answer constraints for the alternative reply.

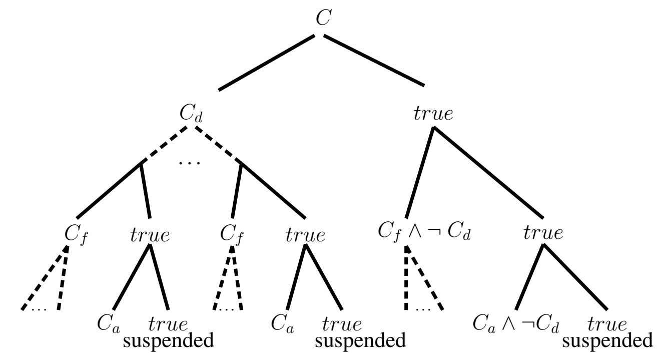

Fig. 3. When the alternative answer C_a for $Q@S$ arrives

On the other hand, when a revised answer, with the constraint C_r, comes, all processes using the first (or current) answer are splitted, in order to obtain the answer constraints which are logically equivalent to the answer constraints of:

$$\leftarrow C \wedge C_f \wedge C_r \| Bs \backslash \{Q@S\},$$

and the suspended original process is splitted as well, in order to obtain the answer constraints which are logically equivalent to the answer constraints of $\leftarrow C \wedge \neg C_f \wedge C_r \| Bs \backslash \{Q@S\}$ (Fig. 4). By gathering these answer constraints, we can override the previous reply by the revised reply.

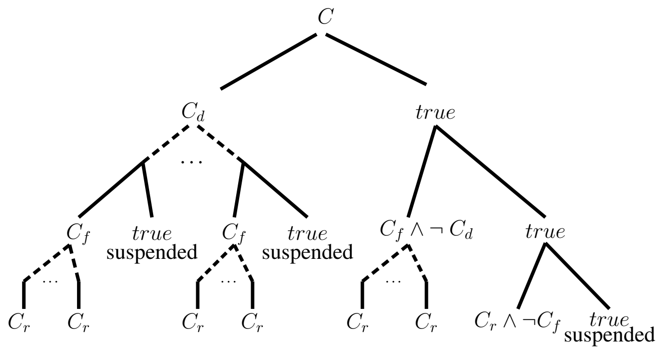

Fig. 4. When the revised answer C_r for $Q@S$ arrives

3.2 Preliminary Definitions

A process is either an *ordinary process* or a *finished process*. An *ordinary process* P is an expression of the form $\langle PID, C, GS, WA, AA \rangle$ where:

– PID: the ID for a process denoted as $pid(P)$;

- C: the current constraint in the goal denoted as $pconst(P)$;
- GS: the body in the goal denoted as $gs(P)$;
- WA: a set of pairs $\langle Q@S, WAID\rangle$ where $Q@S$ is an askable atom and $WAID$ is the ID of an answer entry whose answer is waited for by the process. We denote WA as $wa(P)$.
- AA: a set of pairs $\langle Q@S, AAID\rangle$ where $Q@S$ is an askable atom and $AAID$ is the ID of an answer entry whose answer is used in the process. We denote AA as $aa(P)$.

A *finished process* FP is an expression of the form $\langle Query, FPID, C\rangle$ where:

- $Query$: an initial query for this process. It is used to send an answer to the asking agent;
- $FPID$: the ID for a process. This is also used when this answer is returned to the asking agent;
- C: the current constraint in the process.

For simplicity, an ordinary process is sometimes just called a process.

An *answer entry* A is an expression of the form $\langle Q@S, AID, C, UPIDs\rangle$ where:

- $Q@S$: the query given to the other agent denoted as $aq(A)$;
- AID: the ID for an answer entry denoted as $aid(A)$. We have the special IDs, "o" for the answer entry created when this query is firstly asked, and "$d_1, ...$" for default answers. We call an answer entry with the ID "o" an *original answer entry* for $Q@S$, an answer entry with an ID of "$d_1, ...$" a *default answer entry*, and other answer entries *ordinary answer entries*;
- C: the most recent answer constraint for $Q@S$ for answer entry A denoted as $aconst(A)$. The constraint of the original answer entry is defined as $true$;
- $UPIDs$: the set of IDs of processes using an answer in A denoted as $ups(A)$.

3.3 Process Reduction Phase

In the process reduction phase, we process the constraints we have, in a regular CP way. The only difference is that we may have to consider default information, or answers. In this subsection, we describe how we manage processes, following the above-given definitions.
We do the following until no more process can be processed.

- When a query $Q_{init}@S_{self}$ is asked from another agent S' where S_{self} is the ID for this agent, we record Q_{init} as the initial query and S' as the asking agent. We then create a new process $\langle PID, \{\}, Q_{init}, \{\}, \{\}\rangle$ where PID is a new process ID.
- If there is an ordinary process P such that $gs(P) = wa(P) = \emptyset$,
  1. Send an answer to the asking agent S' which is of the form: $\langle Q_{init}@S_{self}, pid(P), pconst(P)\rangle$.
  2. We change this process into a finished process of the form: $\langle Q_{init}@S_{self}, pid(P), pconst(P)\rangle$.

– Else if there is a process P such that $gs(P) \neq \emptyset$ and $wa(P) = \emptyset$, then we select an atom L in $gs(P)$ and reduce L as follows.
  - If L is a non-askable atom,
    1. For every rule R such that $pconst(P) \wedge (L = head(R)) \wedge const(R)$ is consistent, we do the following:
       (a) We create the following process $\langle newPID, newC, GS, \{\}, AA\rangle$ where
         - ∗ $newPID$ is a new process ID;
         - ∗ $newC := pconst(P) \wedge (L = head(R)) \wedge const(R)$;
         - ∗ $GS := body(R) \cup gs(P) \backslash \{L\}$;
         - ∗ $AA := aa(P)$.

       (b) For every answer entry A s.t. $\langle aq(A), aid(A)\rangle$ in $aa(P)$, $ups(A) := ups(A) \cup \{newPID\}$.
    2. For every answer entry A s.t. $\langle aq(A), aid(A)\rangle$ in $aa(P)$, $ups(A) := ups(A) \backslash \{pid(P)\}$.
    3. We delete P.
  - If L is an askable atom $Q@S$,
    1. We do either of the following according to non-arrival/arrival of the answer.
       - ∗ If there is no ordinary answer entry of the form $\langle Q@S, AID, C, UPIDs\rangle$, then for each default "$Q@S \leftarrow C_d \|$." such that $pconst(P) \wedge C_d$ is consistent, we do the following:

       (a) We create a new process $\langle newPID, newC, GS, \{\}, AA\rangle$ where
         - · $newPID$ is a new process ID.
         - · $newC := pconst(P) \wedge C_d$
         - · $GS := gs(P) \backslash \{Q@S\}$
         - · $AA := aa(P) \cup \{\langle Q@S, d\rangle\}$ where d is an ID for this default.

       (b) We associate the newly created process with a default d of $Q@S$ as follows.
         - · If there is a default answer entry $A_d = \langle Q@S, d, C_d, UPIDs_d\rangle$, then $ups(A_d) := UPIDs_d \cup \{newPID\}$.
         - · Else if there is no default answer of the form $\langle Q@S, d, C_d, UPIDs_d\rangle$, we create the answer entry $\langle Q@S, d, C_d, \{newPID\}\rangle$.

       (c) For every answer entry A s.t. $\langle aq(A), aid(A)\rangle$ in $aa(P)$, $ups(A) := ups(A) \cup \{newPID\}$.
       - ∗ Else if there exists an ordinary answer entry of the form $\langle Q@S, AID, C, UPIDs\rangle$, then for each ordinary answer entry $\langle Q@S, AID, C_a, UPIDs\rangle$ s.t. $pconst(P) \wedge C_a$ is consistent, we do the following:

       (a) We create a new process $\langle newPID, newC, GS, \{\}, AA\rangle$ where
         - · $newPID$ is a new process ID.
         - · $newC := pconst(P) \wedge C_a$
         - · $GS := GS \backslash \{Q@S\}$

        · $AA := aa(P) \cup \{\langle Q@S, AID \rangle\}$.
    (b) For every answer entry A s.t. $\langle aq(A), aid(A) \rangle$ in $aa(P)$,
        $ups(A) := ups(A) \cup \{pid(P)\}$.
2. We associate P with $Q@S$ as follows.
   ∗ If there is an original answer entry $A_o = \langle Q@S, o, true, UPIDs_o \rangle$, then $ups(A_o) := UPIDs_o \cup \{pid(P)\}$.
   ∗ Else if there is no original answer entry of the form $\langle Q@S, o, true, UPIDs \rangle$, we create an answer entry $\langle Q@S, o, true, \{pid(P)\} \rangle$, and send a question Q to S.
3. $wa(P) := \{\langle Q@S, o \rangle\}$

3.4 Fact Arrival Phase

Suppose that an answer is returned from an agent S for a question $Q@S$ of the form $\langle Q@S, AID, C \rangle$. Then, we do the following after one step of process reduction is finished.

– If there is no answer entry of the form $\langle Q@S, AID, C_f, UPIDs' \rangle$[9],
  1. We create an answer entry $\langle Q@S, AID, C, UPIDs \rangle$ where $UPIDs$ is set to $\emptyset$ initially, but will be incremented as shown below.
  2. For every default answer entry for a default d of the form $\langle Q@S, d, C_d, UPIDs_d \rangle$ and for every process P_d such that $pid(P_d) \in UPIDs_d$, we do the following:
     • If P_d is a finished process of the form $\langle Q_{init}@S_{self}, PID, C_{Final} \rangle$ s.t. $C \wedge C_{Final} \neq C_{Final}$, we send an answer of the form $\langle Q_{init}@S_{self}, PID, C \wedge C_{Final} \rangle$ to the asking agent S'.
     • If P_d is an ordinary process,
       (a) $wa(P_d) := wa(P_d) \cup \{\langle Q@S, d \rangle\}$
       (b) $aa(P_d) := aa(P_d) \backslash \{\langle Q@S, d \rangle\}$
       (c) If $C \wedge pconst(P_d)$ is consistent, we do the following.
          i. We create the following process
             $\langle newPID, newC, GS, WA, AA \rangle$ where
             ∗ $newPID$ is a new process ID.
             ∗ $newC := C \wedge pconst(P_d)$
             ∗ $GS := gs(P_d)$.
             ∗ $WA = wa(P_d)$
             ∗ $AA = aa(P_d) \cup \{\langle Q@S, AID \rangle\} \backslash \{\langle Q@S, d \rangle\}$
          ii. $UPIDs := UPIDs \cup \{newPID\}$.
  3. Pick up the original answer entry of the form $\langle Q@S, o, true, UPIDs_o \rangle$.
  4. For every process P_o such that $pid(P_o) \in UPIDs_o$ and $C \wedge pconst(P_o) \wedge \bigwedge_{(Q@S \leftarrow C_d \|) \in \Delta} \neg C_d$ is consistent, do the following:
     (a) We create the following process
         $\langle newPID, newC, GS, WA, AA \rangle$ where
         • $newPID$ is a new process ID.

[9] This means that the arriving answer is an alternative answer to the query $Q@S$.

        - $newC := C \wedge pconst(P_o) \wedge \bigwedge_{(Q@S \leftarrow C_d \|) \in \Delta} \neg C_d$
        - $GS := gs(P_o)$.
        - $WA := wa(P_o) \backslash \{\langle Q@S, o\rangle\}$
        - $AA := aa(P_o) \cup \{\langle Q@S, AID\rangle\}$

     (b) $UPIDs := UPIDs \cup \{newPID\}$.

– Else if there is an answer entry of the form $\langle Q@S, AID, C_f, UPIDs'\rangle$[10],

1. We change $\langle Q@S, AID, C_f, UPIDs'\rangle$ into $\langle Q@S, AID, C, UPIDs\rangle$ where $UPIDs := UPIDs'$ initially but will be incremented/decremented as shown below.
2. For every process P such that $pid(P) \in UPIDs'$ do the following:
    - If P is a finished process of the form $\langle Q_{init}@S_{self}, PID, C_{Final}\rangle$ s.t. $C \wedge C_{Final} \neq C_{Final}$, we send an answer of the form $\langle Q_{init}@S_{self}, PID, C \wedge C_{Final}\rangle$ to the asking agent S'.
    - If P is an ordinary process,
        - ∗ If $C \wedge pconst(P)$ is consistent, $pconst(P) := C \wedge pconst(P)$.
        - ∗ Otherwise, delete P and $UPIDs := UPIDs \backslash \{pid(P)\}$.
3. Pick up the original answer entry of the form $\langle Q@S, o, true, UPIDs_o\rangle$.
4. For every process P_o such that $pid(P_o) \in UPIDs_o$ and $C \wedge pconst(P_o) \wedge \neg C_f$ is consistent, we do the following:

    (a) We create the following process $\langle newPID, newC, GS, WA, AA\rangle$ where
        - $newPID$ is a new process ID.
        - $newC := C \wedge pconst(P_o) \wedge \neg C_f$
        - $GS := gs(P_o)$.
        - $WA := wa(P_o) \backslash \{\langle Q@S, o\rangle\}$
        - $AA := aa(P_o) \cup \{\langle Q@S, AID\rangle\}$

    (b) $UPIDs := UPIDs \cup \{newPID\}$.

3.5 Execution Trace Example

We show a part of an execution trace for a question $rsv(R, L, D)$ in Example 1. In this trace, we consider a scenario which highlights process updates upon arrivals of an alternative answer and revised answer. We firstly give the initial process $\langle p_0, \{\}, \{rsv(R, L, D)\}, \{\}, \{\}\rangle$.

1. Select process p_0 and reduce it to p_1, p_2, p_3.
   Processes:
   $\langle p_1, \{R = tr, L = [a, b]\}, \{fr(D)@a, fr(D)@b\}, \{\}, \{\}\rangle$,
   $\langle p_2, \{R = sr, L = [a]\}, \{fr(D)@a, bs(D)@b\}, \{\}, \{\}\rangle$,
   $\langle p_3, \{R = sr, L = [b]\}, \{bs(D)@a, fr(D)@b\}, \{\}, \{\}\rangle$

[10] This means that the arriving answer is a revised answer of one of the previous answer to the query $Q@S$.

2. Select p_1, and ask a question $fr(D)@a$, and create answer entries for $fr(D)@a$ and new processes p_4, p_5 for default answers.
   Answer entries:
   $\langle fr(D)@a, o, true, \{p_1\}\rangle$,
   $\langle fr(D)@a, d_1, \{D = 1\}, \{p_4\}\rangle$,
   $\langle fr(D)@a, d_2, \{D = 2\}, \{p_5\}\rangle$
   Processes: p_2, p_3,
   $\langle p_4, \theta_{tr} \cup \{D = 1\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, d_1\rangle\}\rangle$[11],
   $\langle p_5, \theta_{tr} \cup \{D = 2\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, d_2\rangle\}\rangle$,
   $\langle p_1, \theta_{tr}, \{fr(D)@b\}, \{\langle fr(D)@a, o\rangle\}, \{\}\rangle$

3. Suppose that $\langle fr(d)@a, a_1, \{D=2\}\rangle$ is returned from the agent a. We suspend p_4 and p_5 since they use a default answer and then create new processes p_6 from p_5 since the default answer used in p_5 is consistent with the returned answer. Note that we create no new process from p_1 since the returned answer contradicts one of negations of default answers.
   Answer entries: fra_o, fra_{d_1}, fra_{d_2}[12],
   $\langle fr(D)@a, a_1, \{D = 2\}, \{p_6\}\rangle$
   Processes: p_1, p_2, p_3,
   $\langle p_6, \theta_{tr2}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, a_1\rangle\}\rangle$,
   $\langle p_4, \theta_{tr1}, \{fr(D)@b\}, \{\langle fr(D)@a, d_1\rangle\}, \{\}\rangle$,
   $\langle p_5, \theta_{tr2}, \{fr(D)@b\}, \{\langle fr(D)@a, d_2\rangle\}, \{\}\rangle$[13]

4. Suppose that $\langle fr(D)@a, a_2, \{D = 3\}\rangle$ is returned from the agent a. Since this has the different answer ID from the previous answer in the last step, this answer is an alternative answer. Then, we create a new process from p_1 which is the original process for query $fr(D)@a$. Note that we create no new process from the processes created by default answers for $fr(D)@a$ since this answer contradicts the defaults.
   Answer entries: fra_o, fra_{d_1}, fra_{d_2}, fra_{a_1}[14],
   $\langle fr(D)@a, a_2, \{D = 3\}, \{p_7\}\rangle$
   Processes: $p_1, p_2, p_3, p_4, p_5, p_6$,
   $\langle p_7, \theta_{tr} \cup \{D = 3, D \neq 1, D \neq 2\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, a_2\rangle\}\rangle$

5. Suppose that $\langle fr(D)@a, a_1, \{D = 1\}\rangle$ is returned from the agent a. The ID a_1 for the returned answer indicates that this answer is a revised answer for "$D = 2$". Therefore, we revise every process using a_1 which is recorded in the answer entry fra_{a_1}. This is p_6, but its associated constraint is contradictory with the returned answer, and therefore we kill this process. Then, we create a new process p_8 from p_1.

[11] $\theta_{tr} = \{R = tr, L = [a, b]\}$.
[12] $fra_o = \langle fr(D)@a, o, true, \{p_1\}\rangle$,
$fra_{d_1} = \langle fr(D)@a, d_1, \{D = 1\}, \{p_4\}\rangle$,
$fra_{d_2} = \langle fr(D)@a, d_2, \{D = 2\}, \{p_5\}\rangle$.
[13] $\theta_{tr2} = \theta_{tr} \cup \{D = 2\}$ and $\theta_{tr1} = \theta_{tr} \cup \{D = 1\}$.
[14] $fra_{a_1} = \langle fr(D)@a, a_1, \{D = 2\}, \{p_6\}\rangle$.

Answer entries: fra_o, fra_{d_1}, fra_{d_2}, fra_{a_2}[15],
$\langle fr(D)@a, a_1, \{D = 1\}, \{p_8\}\rangle$
Processes: $p_1, p_2, p_3, p_4, p_5, p_7$,
$\langle p_8, \theta_{tr} \cup \{D = 1, D \neq 2\}, \{fr(D)@b\}, \{\}, \{\langle fr(D)@a, a_1\rangle\}\rangle$

4 Correctness of the Operational Model

We guarantee that the above operational model gives a correct answer w.r.t. the most recent replies. Let us note that we assume that the order of reply messages is preserved.

Theorem 1. *Let $\langle \Sigma, \Delta, \mathcal{P}\rangle$ be a framework for speculative constraint computation. Suppose that there is an ordinary process P such that $gs(P) = wa(P) = \emptyset$ for the initial query Q_{init}. Let*

$$\mathcal{R} = \{\text{“}Q@S \leftarrow C\|\text{”} \mid \textit{there exists an answer entry } \langle Q@S, AID, C, UPIDs\rangle \ \textit{s.t.}\ \langle Q@S, AID\rangle \in aa(P)\}.$$

Then, there exists an answer constraint C' w.r.t. Q_{init}, the framework and $\mathcal{R}$ s.t. $\pi_V(pconst(P))$ entails $\pi_V(C')$, where V is the set of the variables that occur in Q_{init}, and π_V is the projection of constraints onto V.

5 Space complexity of our approach

Our approach, compared to traditional approaches (no belief revision), generates an additional cost in terms of space. In this section, we briefly show that the additional cost in space is linear. This cost is observed based on the size of the set PS of processes related to the revised or alternative answer to handle.

When a revised answer comes, say C_r, as shown in Fig. 4:

- if C_r entails the previous answer, say C_f, PS either remains the same size, or reduces (because some processes in PS may now have inconsistent constraints and therefore be killed);
- if C_r is inconsistent with C_f, then all the processes using C_f in PS are killed, the original suspended processes are duplicated and resumed with C_r, and therefore PS grows by at most the number of original suspended processes;
- if C_r is consistent with C_f but does not entail it, PS grows by at most the number of original suspended processes.

These three cases exhibit only linear (or less) behavior.

When an alternative answer comes, say C_a, as shown in Fig. 3, all the suspended processes created on the arrival of the first answer, as well as the original suspended processes, are duplicated and resumed with C_a. Therefore, PS grows by at most the number of these suspended processes.

As briefly covered here, the growth of the set of processes on the arrival of revised and alternative answers follows a linear behavior.

[15] $fra_{a_2} = \langle fr(D)@a, a_2, \{D = 3\}, \{p_7\}\rangle$.

6 Conclusion

In this paper, we presented an operational model for speculative constraint processing with iterative revision for alternative answers. This paper is a generalization of two previous works; the work of revisable speculative computation for yes/no questions [6] and the work of non-revisable speculative computation for queries with constraints [4].

As future work, we will prove correctness and completeness for more general forms of multi-agent systems, where every agent can perform speculative computation. Our current framework is focused on master-slave multi-agent systems, and defines the operational model of master agents. To handle a more general multi-agent system, we need to guarantee the appropriate computation of the overall system by additionally considering communication paths among agents. As another direction, we will also consider applications for this framework.

References

1. J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.
2. S. Janson and S. Haridi. Programming paradigms of the andorra kernel language. In *Proc. of ISLP'91*, pages 167–186, 1991.
3. R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. In *Annals of Mathematics and Artificial Intelligence*, volume 25, pages 391–419, 1999.
4. K. Satoh, P. Codognet, and H. Hosobe. Speculative constraint processing in multi-agent systems. In *Proc. of PRIMA2003, LNCS 2891*, pages 133 – 144, 2003.
5. K. Satoh, K. Inoue, K. Iwanuma, and C. Sakama. Speculative computation by abduction under incomplete communication environments. In *Proc. of ICMAS2000*, pages 263–270, 2000.
6. K. Satoh and K. Yamamoto. Speculative computation with multi-agent belief revision. In *Proc. of AAMAS2002*, pages 897 – 904, 2002.
7. C. Schulte. Programming constraint services: High-level programming of standard and new constraint services. In *LNCS*, volume 2302. Springer Verlag, 2002.