# Stabilization of Information Sharing for Queries Answering in Multiagent Systems

Phan Minh Dung, Do Duc Hanh, and Phan Minh Thang

Computer Science and Information Management
Asian Institute of Technology, Thailand
{dung,hanh,thangfm}@cs.ait.ac.th

**Abstract.** We consider multiagent systems situated in unpredictable environments. Agents viewed as abductive logic programs with abducibles being literals the agent could sense or receive from other agents, must cooperate to provide answers to users as they may not have the knowledge or the capabilities to sense relevant changes in their environment. As their surroundings may change unpredictably, agents may provide wrong answers to queries. Stabilization refers to a capability of the agents to eventually answer queries correctly despite unpredictable environment changes and the incapability of many agents to sense such changes.It could be viewed as the correctness criterium of communicating cooperative multiagent systems.

For efficiency, a piece of information obtained from other agents may be used to answer many queries. Surprisingly, this natural form of "information sharing" may be a cause of non–stabilization of multiagent systems. We formulate postulates and present a formal framework for studying stabilization with information sharing and give sufficient conditions to ensure it.

**Keywords:** Stabilization, Information Sharing, Abductive Logic Programs, Cooperative Multiagent Systems.

## 1 Introduction

Cooperative agents are entities that work together to achieve common objectives. To operate effectively in a changing and unpredictable environment, agents need correct information about their surroundings. Due to limited knowledge and sensing capability, agents need to cooperate with each other to get such information by sending requests and receiving replies. Stabilization, a key characteristics of cooperative multiagent systems, represents the capability of the agents to eventually get correct information ([4]).

*Example 1.* Consider a system of two agents $A$ and $B$, where the knowledge base of $A$ consists of the clause:

$$p \leftarrow q$$

while the knowledge base of agent $B$ consists of the clause:

$$q \leftarrow f$$

where $f$ is a fluent about the environment and could be sensed by only $B$. We assume $B$ always notices any change on $f$ instantly.

Suppose there is a user query from an external agent $E$ to $A$ on $p$. To answer it, $A$ sends a query on $q$ to $B$ and waits for a reply. Suppose that $f$ is true. $B$ will send a reply *"$q$!" ($q$ is true)* to $A$ informing that $q$ is true. In turn, after receiving "$q$!" from $B$, $A$ sends an answer "$p$!" to $E$. $E$ gets a correct answer for the query on $p$ from $A$ in this case.

The environment can change unpredictably. Suppose immediately after $B$ sends reply "$q$!" to $A$, but before $A$ sends reply "$p$!" to $E$, $f$ changes to $false$. The information on $q$ that $A$ has received from $B$, becomes incorrect and so does the answer "$p$!" that $E$ receives from $A$.

Of course if the environment (i.e. $f$) does not change anymore afterwards, a new query on $p$ to $A$ would be answered correctly because $A$ would send a new request on $q$ to $B$, then $B$ would reply with "$\neg q$!" (*$q$ is false*), and at last $A$ would send "$\neg p$!" to $E$. Our system in this scenario is said to be **stabilizing**, i.e. **even though environment changes could cause temporarily incorrect answers to some queries, but once the environment stops changing, all queries will be answered correctly after some delay**.

*Example 2 (Continuation of Example 1)*

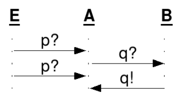


**Fig. 1.** Message Exchanges in Example 2

Consider the message exchanges in Figure 1. Should $A$ send a new query "$q$?" to $B$ to answer the second query "$p$?" from $E$?.

Naturally, $A$ should not send another query "$q$?" to $B$. On receiving information about $q$ from $B$ in the fourth message, $A$ uses it to answer both user queries. This **information sharing** is a natural common mechanism in real world multiagent systems (including humans) for efficiency. It is captured by the following postulate:

**Postulate 1: If a request for some information has been made and an answer to it is being expected, then no request for the same information should be made now**.

The environment can change unpredictably and information an agent has obtained from others may become incorrect without the agent being aware of it. To deal with this problem, the agent should not use information obtained from other agents forever. This motivates Postulate 2.

**Postulate 2: No information obtained from other agents should be used forever.**

When should input information be deleted? Ideally when it becomes false. The problem is that an agent may not know when the environment changes and so the agent may not know whether information obtained from other agents is false or not. For instance in the above examples there is no way agent $A$ could know whether the information about $q$ she has received from $B$ is true or false.

The answer to the question could only be: "Information should be deleted some time after it has been received". Though this "some time" is domain-dependent, we could ask whether there is a "lower bound" for that time.

Come back to the scenario in Figure 1, we expect that the reply "$q!$" from $B$ to $A$ should be used to answer the two queries "$p?$" that have been received by $A$ before. This motivates Postulate 3.

**Postulate 3: Information obtained from other agents should be deleted only after all queries received before the information is obtained and to which the information is relevant, are answered**.

*The question we ask now is: "Would a multiagent system stabilize if Postulates 1 to 3 are satisfied?".*

Unfortunately, in general multiagent systems do not stabilize even if Postulates 1 to 3 are all satisfied as the following example shows.

*Example 3* Consider a system of two agents $A$ and $B$ where the knowledge base of $A$ consists of the clause:

$$q \leftarrow \neg r(x)$$

and the knowledge base of $B$ consists of the clauses:

$$r(x+1) \leftarrow r(x) \text{ and } r(0) \leftarrow \ .$$

Note that there is no environment change in this case.

Suppose $A$ receives a query "$q?$" from some external agent. Obviously the correct answer to this query is "$\neg q!$". To answer the query "$q?$", $A$ may first send a query "$\neg r(0)?$" to $B$ and $B$ replies with "$r(0)!$". $A$ hence has to send another query e.g. "$\neg r(1)?$" to $B$ and $B$ replies with "$r(1)!$". The exchanges will continue that $A$ will send a query "$\neg r(n+1)?$" after receiving a reply "$r(n)!$" from $B$ and the information obtained by $A$ is never sufficient to answer "$q?$". Hence $A$ could never be able to answer the query "$q?$". The system is not stabilizing.

The purpose of this paper is to formalize the problem of stabilization in multiagent systems and study general conditions under which the stabilization with information sharing following Postulates 1 to 3 is guaranteed.

The rest of this paper is organized as follows. In section 2 we briefly introduce the basic notations, definitions and lemmas of acyclic and abductive logic programs and admissibility semantics that are needed through this paper. Problem formalization and results are presented in section 3. We summarize related works and conclude in section 4. Due to space limitation, proofs of lemmas and theorems are skipped in this paper.

## 2 Acyclic Logic Programs and Admissibility Semantics

We assume the existence of a Herbrand base $HB$. A normal logic program is a set of ground clauses of the form:

$a \leftarrow l_1, \ldots, l_m$

where $a$ is an atom from $HB$, and $l_1, \ldots, l_m$ are literals (i.e. atoms or negations of atoms) over $HB$, $m \geq 0$. $a$ is called the head, and $l_1, \ldots, l_m$ the body of the clause. Note that clauses with variables are considered as a shorthand for the set of all their ground instantiations.

Given a logic program $P$, $head(P)$ and $body(P)$ denote the sets of atoms occuring in the heads and the bodies of clauses of $P$ respectively.

For each atom $a$, the **definition** of $a$ in $P$ is the set of all clauses in $P$ whose head is $a$. A logic program is **bounded** if the definition of every atom is finite.

The **atom dependency graph** of a logic program $P$ is a graph, whose nodes are atoms in $HB$ and there is an edge from $a$ to $b$ in the graph iff there is a clause in $P$ whose head is $a$ and whose body contains $b$ or $\neg b$.

A logic program $P$ is **acyclic** iff there is no infinite directed path in its atom dependency graph.

An atom $b$ is said to be **relevant** to an atom $a$ in $P$ if there is a path from $a$ to $b$ in the atom dependency graph of $P$. Further a literal $l$ is relevant to another literal $l'$ if the atom of $l$ is relevant to the atom of $l'$.

Abusing notation we write $\neg l$ for complement of $l$, i.e. $\neg l$ is $a$ if $l$ is $\neg a$ and $\neg l = \neg a$ if $l$ is $a$. Given a set $S$ of literals, $\neg S = \{\neg l \mid l \in S\}$. A set of literals is *consistent* if it does not contain any pair of a literal and its complement.

Given a logic program $P$ and a consistent set of literals $S$. We write $P \cup S \vdash l$ iff there is a sequence of literals $l_1, \ldots, l_n = l$, $n \geq 1$ such that for all $m = 1 \ldots n$: $l_m \in S$ or there exists a clause $l_m \leftarrow l'_1, \ldots, l'_k$ in $P$ s.t. $l'_1, \ldots, l'_k \in S \cup \{l_1, \ldots, l_{m-1}\}$.

An **abductive logic program** is a tuple $\langle P, Ab \rangle$ where $P$ is a logic program, $Ab$ is a set of atoms in $body(P)$ called abducible atoms such that no element of $Ab$ occurs in the head of any clause in $P$, i.e. $Ab \cap head(P) = \emptyset$. Literals over $Ab$ called abducible literals or **abducibles** for short ( [11], [10], [1] ). $\langle P, Ab \rangle$ is **acyclic** if $P$ is acyclic.

Let $\langle P, Ab \rangle$ be an abductive program. The set of **assumptions** $\mathbb{A}$ of $\langle P, Ab \rangle$ comprises $Ab$ and the set of all negative literals including negative abducibles.

A set of assumptions $S$ **attacks** a set of assumptions $R$ if there is $\alpha \in R$ such that $P \cup S \vdash \neg \alpha$.

A set of assumptions is **admissible** if it does not attack itself and attacks every set of assumptions attacking it. It is not difficult to see that admissible sets of assumptions are consistent.

A **preferred extension** of $\langle P, Ab \rangle$ is a maximal (wrt set inclusion) admissible set of assumptions. Note that in difference to a normal acyclic logic program, an acyclic abductive logic program may have more than one preferred extensions.

**Lemma 1.** *Let $\langle P, Ab \rangle$ be an acyclic program. For each maximal consistent set $S$ of abducibles there is a unique preferred extension $E$ of $\langle P, Ab \rangle$ such that $S \subseteq E$.*

A set of assumptions $S \subseteq \mathbb{A}$ is a stable extension of $\langle P, Ab \rangle$ iff for every $a \in Ab$, either $a \in S$ or $\neg a \in S$ and for every $a \notin Ab$, either $P \cup S \vdash a$ or $\neg a \in S$

([3],[9]). Similarly to [3] we can show that each preferred extension of an acyclic abductive program is also stable.

A set $S$ of abducibles is an **abductive solution (or explanation)** for a literal $l$ wrt $\langle P, Ab\rangle$ iff there exists an admissible set of assumptions $S' \subseteq \mathbb{A}$ such that $S = S' \cap (Ab \cup \neg Ab)$ and $P \cup S' \vdash l$. It is not difficult to show that if there exists an abductive solution $S$ for $l$ wrt $\langle P, Ab\rangle$ then there exists a preferred extension $E$ of $\langle P, Ab\rangle$ such that $S \subseteq E$ and $P \cup E \vdash l$.

An abduction solution for $l$ wrt $\langle P, Ab\rangle$ is **non-redundant** if it contains only abducibles relevant to $l$ in $P$.

As for every abductive solution $S$ for a literal $l$ wrt $\langle P, Ab\rangle$ there is a non-redundant abductive solution $R$ for $l$ wrt $\langle P, Ab\rangle$ such that $R \subseteq S$, we restrict our attention on non-redundant abductive solutions.

*Example 4.* Consider an abductive logic program $\langle P, Ab\rangle$ where

$$P = \{q \leftarrow r \quad q \leftarrow \neg p \quad p \leftarrow t\} \text{ and } Ab = \{r, t\}.$$

$S_0 = \{\neg t\}$ is an abductive solution for $q$ wrt $\langle P, Ab\rangle$ as $S_0' = \{\neg p, \neg t\}$ is an admissible set and $P \cup S_0' \vdash q$. Note that $P \cup S_0 \not\vdash q$.

There are four preferred extensions $E_1 = \{r, t\}$, $E_2 = \{r, \neg p, \neg t\}$, $E_3 = \{\neg r, \neg p, \neg t\}$, $E_4 = \{\neg q, \neg r, t\}$ of $\langle P, Ab\rangle$ where $P \cup E_i \vdash q$, $i = 1 \ldots 3$.

In general an admissible set of assumptions is determined largely by its subset of abducibles as shown in the following lemmas.

**Lemma 2.** *If $E$ is a preferred extension of an acyclic abductive program $\langle P, Ab\rangle$ and $R$ is an admissible set of assumptions such that all abducibles in $R$ are in $E$, i.e. $R \cap (Ab \cup \neg Ab) \subseteq E$, then $R \subseteq E$.*

**Lemma 3.** *Given an acyclic logic program $\langle P, Ab\rangle$ and a consistent set $S$ of abducibles ($S \subseteq Ab \cup \neg Ab$). There is no abductive solution for a literal $l$ wrt $\langle P, Ab\rangle$ consistent with $S$ iff for every preferred extension $E \supseteq S$ of $\langle P, Ab\rangle$: $P \cup E \vdash \neg l$.*

There are many ALP systems and abductive proof procedures proposed in the literature (e.g. [5], [3], [7], [10], [1]). In this paper we do not consider complexity of ALP systems. We simply assume the availability of abductive solution generation algorithms.

## 3 Problem Formalization

Let $l$ be a literal. A **query** whether $l$ is true or a **reply** that $l$ is true has a form $l?$ or $l!$ respectively.

### 3.1 Agent and Multiagent System

Agents are situated in environments. An agent could sense some of the changes of her surroundings though not all of them. Let $ENV$ be a set of ground atoms representing the fluents of the environments.

A **multiagent system** is a pair $(\mathcal{A}, ENV)$ where $\mathcal{A}$ is a set of agents situated in an environment characterized by fluent atoms in $ENV$.

**Definition 1 (Agent).** *An* ***agent*** *situated in an environment $ENV$ is represented by a quadtuple $A = (P, HBE, HBI, \Lambda)$ where*

- *$P$, is an acyclic logic program, representing the knowledge base of the agent.*
- *$HBE \subseteq ENV$, representing the sensing capability of the agent, is a set of environment atoms whose truth values the agent could sense. Atoms in $HBE$ do not occur in the head of any clause of $P$.*
- *$HBI$ is the set of input atoms, that occur in the body of some clause in $P$ but not in the head of any clause of $P$ and not in $HBE$, i.e. $HBI = body(P) \setminus (head(P) \cup HBE)$.*
- *$\Lambda$ is the initial state of the agent and will be defined shortly.*

It is not difficult to see that $\langle P, HBE \cup HBI \rangle$ is an abductive logic program.

**Definition 2 (Cooperative Multiagent System)**
*A multiagent system $(\mathcal{A}, ENV)$ with $\mathcal{A} = (A_1, \dots, A_n)$, $A_i = (P_i, HBE_i, HBI_i, \Lambda_i)$ is cooperative iff the following conditions are satisfied:*

- *$ENV = \bigcup_{i=1\dots n} HBE_i$, i.e. each environment change is sensed by some agent.*
- *For each atom $a$, if $a \in head(P_i) \cap head(P_j)$ then $a$ has the same definition in $P_i$ and $P_j$. In other words, agents' domain knowledge bases are consistent.*
- *For every agent $A_i$, for each $a \in HBI_i$, there is an agent $A_j$ such that $a \in head(P_j) \cup HBE_j$, i.e. $A_i$ can get the value of $a$ or $\neg a$ from $A_j$.*
- *No environment atom appears in the head of clauses in the knowledge base of any agent, i.e. for all $i$: $ENV \cap head(P_i) = \emptyset$.*
- *A state of $(\mathcal{A}, ENV)$ is the collection of states of its agents and $(\Lambda_1, \dots, \Lambda_n)$ is the initial state of $(\mathcal{A}, ENV)$.*

*From now on, we focus solely on cooperative multiagent systems. Hence whenever we say "multiagent systems", we mean cooperative ones.*

**Definition 3 (Agent State).** *A state of agent $A = (P, HBE, HBI, \Lambda)$ is a quintuple $\sigma = (EDB,\ RDB,\ SDB, IDB, t)$ where*

- *$EDB \subseteq HBE \cup \neg HBE$ is a maximal consistent set of environment literals containing information the agent has sensed from the environment.*
- *$SDB$ is a database containing the send–out requests whose replies have not been received. An input literal $l$ is inserted into $SDB$ when $A$ sends out a query "$l$?" and is removed from $SDB$ when $A$ receives a reply "$l$!" or "$\neg l$!".*
- *$RDB$ is a database of received queries of the form* $(\textbf{sender}, \textbf{query}, S, id)$ *where*
  - **sender** *is the sender of the query;*
  - **query** *is of the form $l$? where $l$ is a literal;*
  - *$S$ is a non-redundant abductive solution for $l$ wrt $\langle P, HBE \cup HBI \rangle$ or $\perp$ representing non-existence of abductive solutions;*
  - *$id$ is a nonnegative integer used as the identification of the query. Each received query/reply is assigned a unique identification that is also served as a timestamp. The greater is the identification of a query/reply, the more recent the query/reply is received.*

– *$IDB$ is a database containing input information $A$ has obtained from other agents. It is a consistent set of input literals associated with identifications. When $A$ receives a reply "$l$!", $\langle l, t\rangle$ is inserted into $IDB$ and the timestamp counter $t$ of $A$ is increased by* 1.
– *$t$, a nonnegative integer, holds the current timestamp counter. In the initial state $t = 0$.*

*Example 5.* The multiagent system in Example 1 is represented by $(\mathcal{A}, ENV)$ where $\mathcal{A} = (A, B)$, $ENV = \{f\}$, $A = (P_A, HBE_A, HBI_A, \Lambda_A)$, $B = (P_B, HBE_B, HBI_B, \Lambda_B)$ and

$$\begin{array}{llll} P_A = \{p \leftarrow q\} & HBE_A = \emptyset & HBI_A = \{q\} & \Lambda_A = (\emptyset, \emptyset, \emptyset, \emptyset, 0) \\ P_B = \{q \leftarrow f\} & HBE_B = \{f\} & HBI_B = \emptyset & \Lambda_B = (\{f\}, \emptyset, \emptyset, \emptyset, 0) \end{array}$$

Let $S$ be a set of literals and $IDB$ be the input database in an agent state. Abusing the notation, we often say that $S \cup IDB$ is consistent meaning that $S \cup \{l \mid \langle l, id\rangle \in IDB\}$ is consistent.

**Definition 4.** *Given a state $\sigma = (EDB, RDB, SDB, IDB, t)$, let $\Theta = (X, l?, S, id)$, $S \neq \bot$, be a query form in RDB. $\Theta$ is **consistent** wrt $\sigma$ if $S \cup EDB \cup IDB$ is consistent. Otherwise it is inconsistent wrt $\sigma$. $\Theta$ is **verified** wrt $\sigma$ if $S \subseteq EDB \cup IDB$.*

## 3.2 Agent Actions and Environment Changes

Let $\sigma = (EDB, RDB, SDB, IDB, t)$ be the current state of an agent $A = (P, HBE, HBI, \Lambda)$. A state of $A$ changes when the environment changes or $A$ receives/sends a query/reply from/to another agent or deletes some inputs from $IDB$.

1. **Environment change**
   An environment change is represented by a pair $C = (T, F)$ where $T$ (resp. $F$) contains the atoms whose truth values have changed from false (resp. true) to true (resp. false) and $T \cap F = \emptyset$. Given an environment change $C = (T, F)$, what agent $A$ could sense of this change is a pair $(T_A, F_A)$ where $T_A = T \cap HBE$ and $F_A = F \cap HBE$. Hence if a change $C = (T, F)$ occurs then $A$ will update her environment database $EDB$ to
   $$EDB' = (EDB \setminus (F_A \cup \neg T_A)) \cup T_A \cup \neg F_A$$
   The new state of $A$ is denoted by
   $$Upe_A(\sigma, C) = (EDB', RDB, SDB, IDB, t).$$
2. **Receiving a query**
   When $A$ receives a query "$l$?" from some agent $X$ ($X \neq A$), $A$ will generate a query form $\Theta = (X, l?, S, t)$ where $S$ is an abductive solution for $l$ wrt $\langle P, HBE \cup HBI\rangle$ consistent with $EDB \cup IDB$ and insert $\Theta$ into $RDB$. If no such abductive solution exists, $A$ will insert a query form $(X, l?, \bot, t)$ into $RDB$. The new received query database is denoted by $RDB'$.
   The new state of $A$ is denoted by
   $$Upi_A(\sigma, l?, X) = (EDB, RDB', SDB, IDB, t + 1).$$

**3. Receiving a reply**

When receiving a reply "$l!$" from some agent, $A$ updates $IDB$ by deleting any input of the form $\langle l, id \rangle$ and $\langle \neg l, id \rangle$ from it and inserting $\langle l, t \rangle$ into it. The new input database is denoted by $IDB'$. $A$ also removes $l$ and $\neg l$ from $SDB$. The new sent–out database is denoted by $SDB'$.
The new state of $A$ is denoted by

$$Upi_A(\sigma, l!) = (EDB, RDB, SDB', IDB', t+1).$$

**4. Sending out a query**

**Definition 5.** *Let $\Theta = (X, l'?, S, id)$ be a query form in RDB.*
*We say that $A$ **is ready to request information** $l?$ from $B$ for $\Theta$ wrt $\sigma$, where $l$ is an input literal, iff the following conditions are satisfied:*

1. *$\Theta$ is not verified wrt $\sigma$ and*
   ***either** $S \cup EDB \cup IDB$ is consistent and $l \in S$*
   ***or** $\Theta$ is inconsistent wrt $\sigma$ and there is a nonredundant abductive solution $S'$ for $l'$ wrt $\langle P, HBE \cup HBI \rangle$ consistent with $EDB \cup IDB$, $S' \not\subseteq EDB \cup IDB$ and $l \in S'$,*
2. 
   a. *$l \notin SDB$ and $\neg l \notin SDB$ i.e. $A$ is not waiting for replies for queries "$l?$" or "$\neg l?$" (**Postulate 1**).*
   b. *if $\langle l, id' \rangle$ or $\langle \neg l, id' \rangle$ occurs in $IDB$ then $id' < id$ (**Postulate 1**).*[1]
3. *The atom of $l$ is in $head(P_B) \cup HBE_B$ (queries are only sent to agents that can answer them).*

If $A$ sends a request "$l?$" to $B$ ($B \neq A$) for a query form $\Theta = (X, l'?, S, id) \in RDB$ in state $\sigma$ then the following conditions are satisfied:

1. $A$ is ready to request information $l?$ from $B$ for $\Theta$ wrt $\sigma$.
2. If $\Theta$ is inconsistent wrt $\sigma$ then $\Theta$ is replaced in $RDB$ by a new query form $(X, l'?, S', id)$ where $S'$ is a new generated abductive solution for $l'$ wrt $\langle P, HBE \cup HBI \rangle$ consistent with $EDB \cup IDB$, $S' \not\subseteq EDB \cup IDB$ and $l \in S'$,

After sending out "$l?$" to $B$, $A$ will insert $l$ into $SDB$. The new received query and sent–out databases are denoted by $RDB'$ and $SDB'$ respectively.
The new state of $A$ is denoted by

$$Upo_A(\sigma, l?) = (EDB, RDB', SDB', IDB, t).$$

**5. Sending out a reply**

**Definition 6.** *Let $\Theta = (X, l?, S, id)$ be a query form in RDB.*
*We say that $A$ **is ready to answer** $\Theta$ by "$l!$" wrt $\sigma$ iff either $\Theta$ is verified wrt $\sigma$ or if $S \cup EDB \cup IDB$ is inconsistent then there must be an abductive solution $S'$ for $l$ wrt $\langle P, HBE \cup HBI \rangle$ and $S' \subseteq EDB \cup IDB$.*

[1] Postulate 1 states that if $A$ has been waiting for a reply $l!$ or $\neg l!$ then $A$ should not send a query $l?$ or $\neg l?$. It implicitly implies that queries receiving before $id'$ (with identification less than $id'$) should use $\langle l, id' \rangle$ or $\langle \neg l, id' \rangle$ in their answers. Therefore, if a new request for $l$ is made, it should come from queries receiving after $id'$ (with identification greater than $id'$).

*$A$ **is ready to answer** $\Theta$ by "$\neg l!$" wrt $\sigma$ iff either $S = \bot$ or there is no abductive solution for $l$ wrt $\langle P, HBE \cup HBI\rangle$ consistent with $EDB \cup IDB$.*[2]

If $A$ sends "$l!$" or "$\neg l!$" to $X$ ($X \neq A$) in state $\sigma$ then there must be a query form $\Theta = (X, l?, S, id)$ in $RDB$ such that $A$ is ready to answer $\Theta$ by $l!$ or $\neg l!$ wrt $\sigma$ respectively.

After sending out reply "$l!$" or "$\neg l!$", $A$ will remove $\Theta$ from $RDB$. The new received query database is denoted by $RDB'$.

The new state of $A$ is denoted by

$$Upo_A(\sigma, l!, X) = (EDB, RDB', SDB, IDB, t).$$

**6. Deleting possibly stale inputs**

If $A$ deletes an input $\langle l, id\rangle$ from $IDB$, then there is no query form $\Theta = (X, l'?, S, id')$ in $RDB$ where $l$ is relevant to $l'$ and $id' < id$ (**Postulate 3**). $A$ updates $IDB$ to $IDB' = IDB \setminus \{\langle l, id\rangle\}$.

The new state of $A$ is denoted by

$$Upd_A(\sigma, del(l)) = (EDB, RDB, SDB, IDB', t).$$

*Example 6 (Continuation of Example 5).* Consider the system in Example 1, 2 and 5 and the following table of changes in states of agents[3].

| | Event | $A$ | | | | | $B$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $EDB$ | $RDB$ | $SDB$ | $IDB$ | $t$ | $EDB$ | $RDB$ | $SDB$ | $IDB$ | $t$ |
| 0 | | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | 0 | $f$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | 0 |
| 1 | $E, A, p?$ | - | $(E, p?, \{q\}, 0)$ | - | - | 1 | - | - | - | - | - |
| 2 | $A, B, q?$ | - | - | $q$ | - | - | - | $(A, q?, \{f\}, 0)$ | - | - | 1 |
| 3 | $B, A, q!$ | - | - | $\emptyset$ | $\langle q, 1\rangle$ | 2 | - | $\emptyset$ | - | - | - |

0. The initial states of $A$ and $B$ are shown in row 0.
1. $A$ receives a query "$p?$" from an external agent $E$. $A$ generates and adds query form $(E, p?, \{q\}, 0)$ into her received query database, and increases her timestamp counter by 1.
2. $A$ sends out a query "$q?$" to $B$ and adds $q$ into her sent–out database. Receiving the query "$q?$", $B$ generates and adds query form $(A, q?, \{f\}, 0)$ into her received query database, and increases her timestamp counter by 1.
3. $B$ sends out a reply "$q!$" to $A$ and removes the only query form from her received query database. Receiving the reply "$q!$", $A$ inserts input $\langle q, 1\rangle$ into her input database and removes $q$ from her sent–out database.

## 3.3 Runs

The semantics of a multiagent system is defined in terms of runs. A run of a multiagent system is an infinite sequence of transitions that occur when the environment changes or agents send out/receive queries/replies or delete possibly stale inputs from their input databases.

[2] Because by Lemma 3, for every preferred extension $E$ of $\langle P, HBE \cup HBI\rangle$ consistent with $EDB \cup IDB$: $P \cup E \vdash \neg l$.

[3] "-" means unchanged and event "$A, B, \pi$" means that $A$ sends $\pi$ to $B$.

**Definition 7 (Transitions).** *Let $\Sigma = (\sigma_1, \ldots, \sigma_n)$ and $\Sigma' = (\sigma'_1, \ldots, \sigma'_n)$ be states of a multiagent system $(\mathcal{A}, ENV)$.*

1. *An environment transition $\Sigma \xrightarrow{C} \Sigma'$ happens when there is an environment change $C = (T, F)$ and the following conditions are satisfied:*
   – *For every $A_k \notin S_C$: $\sigma'_k = \sigma_k$ and*
   – *For each agent $A_i \in S_C$: $\sigma'_i = Upe_i(\sigma_i, C)$,*[4] *where $S_C$ denotes the set of agents which could sense parts of $C$, i.e. $S_C = \{A_i \mid HBE_i \cap (T \cup F) \neq \emptyset\}$.*
2. *A query transition $\Sigma \xrightarrow{(X, A_i, l?)} \Sigma'$ happens when agent $X$ sends a query on $l$ to agent $A_i$ and the following conditions are satisfied:*
   – *For every $A_k \notin \{X, A_i\}$: $\sigma'_k = \sigma_k$.*
   – *$\sigma'_i = Upi_i(\sigma_i, l?, X)$.*
   – *If $X = A_j$ then $\sigma'_j = Upo_j(\sigma_j, l?)$.*
3. *A reply transition $\Sigma \xrightarrow{(A_i, X, l!)} \Sigma'$ happens when $A_i$ sends "$l!$" to $X$ and the following conditions are satisfied:*
   – *For every $A_k \notin \{A_i, X\}$: $\sigma'_k = \sigma_k$.*
   – *If $X = A_j$ then $\sigma'_j = Upi_j(\sigma_j, l!)$.*
   – *$\sigma'_i = Upo_i(\sigma_i, l!, X)$.*
4. *An input delete transition $\Sigma \xrightarrow{(A_i, del(l))} \Sigma'$ happens when $A_i$ deletes input $\langle l, id \rangle$ from $IDB_i$ and the following conditions are satisfied:*
   – *For every $A_k$, $k \neq i$: $\sigma'_k = \sigma_k$.*
   – *$\sigma'_i = Upd_i(\sigma_i, del(l))$.*
5. *An empty transition $\Sigma \xrightarrow{nil} \Sigma'$ denotes that there is no change in the states of agents.*[5]

*We often simply write $\Sigma \rightarrow \Sigma'$ if there is a transition from $\Sigma$ to $\Sigma'$.*

**Definition 8 (Run).** *Let $(\mathcal{A}, ENV)$ be a multiagent system. A run $\mathcal{R}$ of $(\mathcal{A}, ENV)$ is an infinite sequence of transitions*

$$\mathcal{R} = \Sigma_0 \rightarrow \Sigma_1 \rightarrow \cdots \rightarrow \Sigma_m \rightarrow \ldots$$

*where $\Sigma_k = (\sigma_{1,k}, \ldots, \sigma_{n,k})$, $\sigma_{i,k} = (EDB_{i,k}, RDB_{i,k}, SDB_{i,k}, IDB_{i,k}, t_{i,k})$ such that*

1. *$\Sigma_0$ is the initial state of $(\mathcal{A}, ENV)$.*
2. *There is a point $h$ such that at every $k \geq h$ in the run, there is no more environment change.*
3. *$\mathcal{R}$ satisfies the following fairness condition:*[6] *For every agent $A_i$, for each $k$: there is no query form $\Theta = (X, l?, S, id)$ such that*

[4] We write $Upe_i(\sigma_i, C)$ for $Upe_{A_i}(\sigma_i, C)$.
[5] This transition is introduced to ensure that runs are infinite (See Definition 8).
[6] The fairness condition ensures that actions of sending out request/reply are not delayed indefinitely if they are ready.

  - *for all $m \geq k$, $\Theta \in RDB_{i,m}$ and $A_i$ is ready to request information from other agents for $\Theta$ wrt $\sigma_{i,m}$*[7] *or*
  - *for all $m \geq k$, $\Theta \in RDB_{i,m}$ and $A_i$ is ready to answer $\Theta$ by $l!$ or $\neg l!$ wrt $\sigma_{i,m}$.*

4. *For every $k$, for every input $\langle l, id\rangle \in IDB_{i,k}$: there is a $m \geq k$ such that $\langle l, id\rangle$ is deleted at $m$ (**Postulate 2**)*
   - *either explicitly by transition $\Sigma_m \xrightarrow{(A_i,del(l))} \Sigma_{m+1}$*
   - *or implicitly and replaced by $\langle l, t_{i,m}\rangle$ or $\langle \neg l, t_{i,m}\rangle$ by transition $\Sigma_m \xrightarrow{(A_j,A_i,l!)} \Sigma_{m+1}$ or $\Sigma_m \xrightarrow{(A_j,A_i,\neg l!)} \Sigma_{m+1}$ for some $A_j$ in $\mathcal{R}$.*
5. *If $\Sigma_k \xrightarrow{nil} \Sigma_{k+1}$ then for all $m \geq k$: $\Sigma_m \xrightarrow{nil} \Sigma_{m+1}$.*

*Example 7 (Continuation of Example 6).*
The sequence in Example 6 is a part of the following run:

$$\Sigma_0 \xrightarrow{(E,A,p?)} \Sigma_1 \xrightarrow{(A,B,q?)} \Sigma_2 \xrightarrow{(B,A,q!)} \Sigma_3 \xrightarrow{(\emptyset,\{f\})} \Sigma_4 \xrightarrow{(A,E,p!)} \Sigma_5 \rightarrow \ldots$$

It is easy to see from Definitions 7, 8 and conditions for sending out queries and deleting inputs in section 3.2 that the following lemma holds.

**Lemma 4.** *Each run of a multiagent system satisfies Postulates 1 to 3 of information sharing.*

### 3.4 Superagent

The superagent of a multiagent system represents the combined capacity (both reasoning and sensing) of the multiagent system as the whole in the ideal case where all agents are instantly provided all necessary information (e.g. located at one place).

Let $(\mathcal{A}, ENV)$ be a multiagent system with $\mathcal{A} = (A_1, \ldots, A_n)$ and $A_i = (P_i,\ HBE_i, HBI_i, \Lambda_i)$, $\Lambda_i = (EDB_i, RDB_i, SDB_i, IDB_i, 0)$. The **superagent** of $(\mathcal{A}, ENV)$ is the agent $\mathbf{S}_{\mathcal{A}} = (P_{\mathcal{A}}, ENV, \emptyset, \Lambda_{\mathcal{A}})$, where $P_{\mathcal{A}} = P_1 \cup \cdots \cup P_n$ and $\Lambda_{\mathcal{A}} = (EDB, \emptyset, \emptyset, \emptyset, 0)$, $EDB = EDB_1 \cup \cdots \cup EDB_n$.

Note that as $\mathbf{S}_{\mathcal{A}}$ can answer all queries by herself without the need to send requests to other agents, her database of received queries, database of sent–out requests and input database are all empty. Her timestamp is always 0 too. Her state is therefore represented by $EDB$, a maximal consistent set of literals over $ENV$.

The answer of the superagent $\mathbf{S}_{\mathcal{A}}$ to a query "$l?$" at a state $EDB$ is "$l!$" (resp. "$\neg l!$") iff $E \cup P_{\mathcal{A}} \vdash l$ (resp. $E \cup P_{\mathcal{A}} \vdash \neg l$) where $E$ is the preferred extension of $\langle P_{\mathcal{A}}, ENV\rangle$ such that $EDB \subseteq E$.

*Intuitively, an answer of an agent in a multiagent system is correct if it coincides with the answer of the superagent.* Hence stabilization refers to the convergence of agents' answers to the answers of the superagent.

[7] We say that $A$ is ready to request information from other agents for a query form $\Theta \in RDB$ wrt her state $\sigma = (EDB, RDB, SDB, IDB, t)$ if there is an input literal $l$ such that $A$ is ready to request information $l?$ from some agent $B$ for $\Theta$ wrt $\sigma$.

### 3.5 Stabilization

Let $\mathcal{R} = \Sigma_0 \rightarrow \cdots \rightarrow \Sigma_h \rightarrow \ldots$ be a run of a multiagent system $(\mathcal{A}, ENV)$.

**Definition 9.**

– *A query "$l?$" from agent $X$ to agent $A_i$ at point $k$ in $\mathcal{R}$* [8] *has an answer $l!$ or $\neg l!$ at $m$ $(m > k)$ iff there is a reply transition of the form*

$$\Sigma_m \xrightarrow{(A_i,X,l!)} \Sigma_{m+1} \text{ or } \Sigma_m \xrightarrow{(A_i,X,\neg l!)} \Sigma_{m+1}$$

*in $\mathcal{R}$ and there exist query forms $\Theta = (X, l?, S, id)$, $\Theta' = (X, l?, S', id)$ such that $RDB_{i,k+1} \setminus RDB_{i,k} = \{\Theta\}$ and $RDB_{i,m} \setminus RDB_{i,m+1} = \{\Theta'\}$* [9].

– *A query "$l?$" from $X$ to $A_i$ at $k$ in $\mathcal{R}$ is said to be* ***correctly answered*** *iff*
  - *it has the answer $l!$ or $\neg l!$ at some $m > k$ and*
  - *the superagent provides the same answer at state*
$$EDB_m = EDB_{1,m} \cup \cdots \cup EDB_{n,m}.$$

– *$\mathcal{R}$ is* ***convergent*** *if there is a point $h$ such that every query appearing in $\mathcal{R}$ at any point $k \geq h$ is answered correctly.*

**Definition 10.** *A multiagent system is said to be* **stabilizing** *iff each of its runs is convergent.*

Example 3 shows that stabilization is not guaranteed in general. The following example illustrates that even if the program of each agent is finite, stabilization is not guaranteed.

*Example 8.* Consider a multiagent system $(\mathcal{A}, ENV)$ where $\mathcal{A} = (A, B)$, $ENV = \{f\}$ and

$$\begin{array}{llll} P_A = \{p \leftarrow q\} & HBE_A = \emptyset & HBI_A = \{q\} & \Lambda_A = (\emptyset, \emptyset, \emptyset, \emptyset, 0) \\ P_B = \{q \leftarrow p \quad q \leftarrow f\} & HBE_B = \{f\} & HBI_B = \{p\} & \Lambda_B = (\{\neg f\}, \emptyset, \emptyset, \emptyset, 0) \end{array}$$

Consider the following run where $A$ receives a query "$p?$" from an external agent $E$ and there is no environment change.

$$\Sigma_0 \xrightarrow{(E,A,p?)} \Sigma_1 \xrightarrow{(A,B,q?)} \Sigma_2 \xrightarrow{(B,A,p?)} \Sigma_3 \xrightarrow{nil} \Sigma_4 \xrightarrow{nil} \ldots$$

To answer the query "$p?$" from $E$, $A$ sends out a query "$q?$" to $B$. To answer the query "$q?$" from $A$ , $B$ sends out a query "$p?$" to $A$. According to Postulate 1, to answer the query "$p?$" from $B$, $A$ should not send another query on $q$ to $B$. As $B$ does not receive any new query, $B$ will not send or receive anything from $A$. Similarly $A$ will not send or receive anything from $B$. Thus there is a deadlock and both $A$ and $B$ would never get information about $q$ and $p$ respectively. So the query "$p?$" will never be answered.

We introduce now sufficient conditions for stabilization.

[8] i.e. there is a query transition $\Sigma_k \xrightarrow{(X,A_i,l?)} \Sigma_{k+1}$ in $\mathcal{R}$.

[9] i.e. $\Theta$ is generated when $A_i$ receives "$l?$" from $X$ at $k$ and $\Theta'$ is deleted when $A_i$ sends "$l!$" at $m$. $\Theta$, $\Theta'$ have the same identification.

**Definition 11.** *Let $(\mathcal{A}, ENV)$ be a multiagent system and $P_{\mathcal{A}}$ be its superagent's program. The* ***I/O graph*** *of $(\mathcal{A}, ENV)$ is a graph obtained from the atom dependency graph of $P_{\mathcal{A}}$ by removing all nodes that are not relevant to any input atom of agents. $(\mathcal{A}, ENV)$ is* ***IO-acyclic*** *if there is no infinite path in its I/O graph. $(\mathcal{A}, ENV)$ is* ***bounded*** *if $P_{\mathcal{A}}$ is bounded. $(\mathcal{A}, ENV)$ is* ***IO-finite*** *if its I/O graph is finite.*

**Theorem 1.** *IO–acyclic and IO–finite multiagent systems are stabilizing.*

Theorem 1 introduces sufficient conditions for stabilization. Unfortunately these conditions are rather strong. Could we weaken them? Are IO-acyclicity and boundedness sufficient to guarantee the stabilization of a multiagent system?

**Theorem 2.** *IO-acyclicity and boundedness are not sufficient to guarantee the stabilization of a multiagent system.*

*Proof.* We give a counterexample in Example 9.

*Example 9.* Consider a multiagent system $(\mathcal{A}, ENV)$ with $\mathcal{A} = (A, B)$, $ENV = \{p, q\}$ and

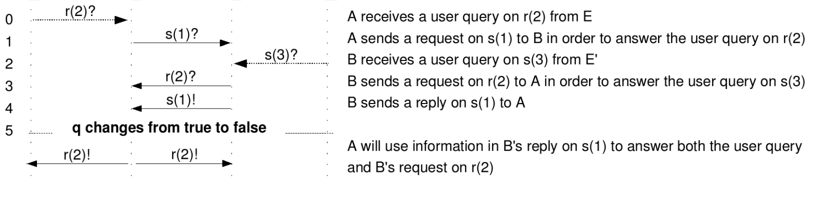
$$\begin{array}{ll}
P_A = \{r(1) \leftarrow p \quad r(n+1) \leftarrow s(n)\} & P_B = \{s(1) \leftarrow q \quad s(n+1) \leftarrow r(n)\} \\
HBE_A = \{p\} & HBE_B = \{q\} \\
HBI_A = \{s(n) \,|\, n \geq 1\} & HBI_B = \{r(n) \,|\, n \geq 1\} \\
\Lambda_A = (\{\neg p\}, \emptyset, \emptyset, \emptyset, 0) & \Lambda_B = (\{q\}, \emptyset, \emptyset, \emptyset, 0)
\end{array}$$

Obviously, $(\mathcal{A}, ENV)$ is bounded and IO–acyclic. It is easy to see that the semantics of agents' programs are as follows: If $p$ is true (resp. false) then all $r(2n+1)$ and $s(2n+2)$, $n \geq 0$, are true (resp. false). Similarly, if $q$ is true (resp. false) then all $s(2n+1)$ and $r(2n+2)$, $n \geq 0$, are true (resp. false).

Suppose that at the beginning $p$ is false, $q$ is true. Consider the following infinite sequence $\mathcal{S}$ of message exchanges between agents:

1. Steps 0 to 5 are given in Figure 2.
2. For every $n \geq 2$, steps $3n$ to $3n+5$ in $\mathcal{S}$ follow the patterns in Figure 3.

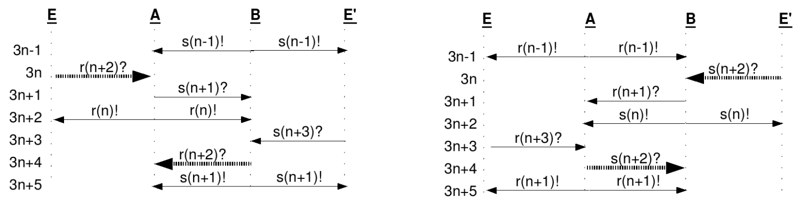
In Figure 2 $A$ receives two queries on $r(2)$ and sends only one request on $s(1)$ to $B$ at step 1 (following **Postulate 1**). $A$ uses the information in $B$'s reply "$s(1)$!" to answer both queries on $r(2)$. Because $q$ is true, $B$'s answer to $A$'s



A receives a user query on r(2) from E
A sends a request on s(1) to B in order to answer the user query on r(2)
B receives a user query on s(3) from E'
B sends a request on r(2) to A in order to answer the user query on s(3)
B sends a reply on s(1) to A

A will use information in B's reply on s(1) to answer both the user query and B's request on r(2)

**Fig. 2.** Sequence $\mathcal{S}$: Steps 0-5



(a) Sharing on $s(n+1)$ (Even $n$) (b) Sharing on $r(n+1)$ (Odd $n$)

**Fig. 3.** Sequence $\mathcal{S}$: Steps $3n$ to $3n+5$

request on $s(1)$ is "$s(1)!$" ($s(1)$ is true) and $A$'s answers to both queries on $r(2)$ are "$r(2)!$" ($r(2)$ is true).

In Figure 3(a), there are two queries on $r(n+2)$ to $A$ (at steps $3n$ and $3n+4$) but only one query on $s(n+1)$ to $B$ (at step $3n+1$). Similarly in Figure 3(b), there are two queries on $s(n+2)$ to $B$ (at steps $3n$ and $3n+4$) but only one query on $r(n+1)$ to $A$ (at step $3n+1$).

To answer all user queries, $A$ needs to request $B$ to provide the value of each $s(n)$, $n$ odd, only once (at step 1 if $n = 1$ and $3n-2$ if $n > 1$) and uses the information in $B$'s reply on $s(n)$ (at step 4 if $n = 1$ and $3n+2$ if $n > 1$) to answer both queries on $r(n+1)$ from $E$ and $B$ (at step $3n+5$). Similarly, $B$ needs to request $A$ to provide the value of each $r(n)$, $n$ even, only once (at step 3 if $n = 2$ and $3n-2$ if $n > 2$) and uses the information in $A$'s reply on $r(n)$ (at step $3n+2$) to answer both the queries on $s(n+1)$ from $E'$ and $A$ (at step $3n+5$). As a result, the answers to queries on $r(2)$, $r(4)$, $\ldots$ and $s(3)$, $s(5)$,$\ldots$ by $A$ and $B$ are all *true*.

As $q$ is false at step 5 and there is no change in the environment after that, the correct answers to queries on $r(2)$, $r(4)$, $\ldots$, and $s(3)$, $s(5)$, $\ldots$ are all $false$.

Because of sharings, the wrong information in $B$'s reply "$s(1)!''$ propagates to $A$'s reply on $r(2)$, the wrong information in $A$'s reply "$r(2)!''$ propagates to $B$'s reply on $s(3)$. This propagation continues upward to replies on $r(4)$, $s(5)$, $\ldots$ and never stops. Consequently, all these replies are incorrect. There is no point in $\mathcal{S}$ where after it the user queries could be answered correctly again. Hence the system is not stabilizing.

## 4 Related Works and Conclusions

Stabilization of distributed protocols has been studied extensively in the literature ([2],[6],[12]) where agents are defined operationally as automata. Dijkstra ([2]) defined a system as stabilizing if it is guaranteed to reach a legitimate state after a finite number of steps regardless of the initial state. The definition of what constitutes a legitimate state is left to individual algorithms.

There are many research works on multiagent systems where logic programming is used to model agent interaction and/or dialogs/negotiations (e.g. [8],

[11]). But until now research on multiagent systems has not considered the question of stabilization.

Agent communications are either push-based or pull-based. In the push-based communication, agents periodically send information to specific recipients without being requested. Push–based communications are common in internet systems like routing systems. On the other hand, in the pull-based communication, agents have to send requests for information to other agents and wait for replies. Dung et al. ([4]) for the first time studies the stabilization of cooperative information multiagent systems for the push-based communication mode.

In this paper we study the stabilization of multiagent systems based on pull–based communication with information sharing.

**Acknowledgments.** We thank the referees for constructive comments and criticisms. This work was partially funded by the Sixth Framework IST program of the European Commission under the 035200 ARGUGRID project.

## References

1. Denecker, M., Kakas, A.C.: Abduction in logic programming. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS, vol. 2407, pp. 402–436. Springer, Heidelberg (2002)
2. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (1974)
3. Dung, P.M.: An argumentation-theoretic foundations for logic programming. Journal of Logic Programming 22(2), 151–171 (1995)
4. Dung, P.M., Hanh, D.D., Thang, P.M.: Stabilization of cooperative information agents in unpredictable environment: a logic programming approach. TPLP 6(1-2), 1–22 (2006)
5. Eshghi, K., Kowalski, R.A.: Abduction compared with negation by failure. In: ICLP, pp. 234–254 (1989)
6. Flatebo, M., Datta, A.K.: Self-stabilization in distributed systems. In: Flatebo, M., Datta, A.K. (eds.) Readings in Distributed Computer Systems, ch. 2, pp. 100–114. IEEE Computer Society Press, Los Alamitos (1994)
7. Fung, T., Kowalski, R.A.: The Iff proof procedure for abductive logic programming. Journal of Logic Programming 33(2), 151–165 (1997)
8. Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: An abductive framework for information exchange in multi-agent systems. In: Dix, J., Leite, J. (eds.) CLIMA 2004. LNCS, vol. 3259, pp. 34–52. Springer, Heidelberg (2004)
9. Gelfond, M., Vladimir Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, pp. 1070–1080 (1988)
10. Kakas, A.C., Robert, A., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. Handbook of Logic in AI and Logic Programming, OUP 5, 235–324 (1998)
11. Satoh, K., Yamamoto, K.: Speculative computation with multi-agent belief revision. In: AAMAS, pp. 897–904 (2002)
12. Schneider, M.: Self-stabilization. ACM Computing Survey 25(1), 45–67 (1993)