Multi-agent Confidential Abductive Reasoning*

Jiefei Ma[1], Alessandra Russo[1], Krysia Broda[1], and Emil C. Lupu[1]

1 **Department of Computing, Imperial College London**
180 Queen's Gate, London, United Kingdom
{j.ma,a.russo,k.broda,e.c.lupu}@imperial.ac.uk

Abstract

In the context of multi-agent hypothetical reasoning, agents typically have partial knowledge about their environments, and the union of such knowledge is still incomplete to represent the whole world. Thus, given a global query they collaborate with each other to make correct inferences and hypothesis, whilst maintaining global constraints. Most collaborative reasoning systems operate on the assumption that agents can share or communicate any information they have. However, in application domains like multi-agent systems for healthcare or distributed software agents for security policies in coalition networks, confidentiality of knowledge is an additional primary concern. These agents are required to collaborately compute consistent answers for a query whilst preserving their own private information. This paper addresses this issue showing how this dichotomy between "open communication" in collaborative reasoning and protection of confidentiality can be accommodated. We present a general-purpose distributed abductive logic programming system for multi-agent hypothetical reasoning with confidentiality. Specifically, the system computes consistent conditional answers for a query over a set of distributed normal logic programs with possibly unbound domains and arithmetic constraints, preserving the private information within the logic programs. A case study on security policy analysis in distributed coalition networks is described, as an example of many applications of this system.

1998 ACM Subject Classification I.2.11 Distributed Artificial Intelligence

Keywords and phrases Abductive Logic Programming, Coordination, Agents

Digital Object Identifier 10.4230/LIPIcs.ICLP.2011.175

1 Introduction

In the context of multi-agent reasoning, each agent has its own *partial knowledge* about the world together with local and/or global constraints. Given a reasoning task, agents interact and compute answers that are consistent with respect to the global constraints. When the union of all the agent knowledge is still incomplete to represent the whole world, hypothetical reasoning is needed, and agents need to collaborate to make correct inferences and hypotheses given a global query. Previously, a general-purpose system called DAREC has been developed, which combines distributed problem solving and abductive logic programming, for multi-agent hypothetical reasoning. In DAREC, agent knowledge is represented as a normal logic program, and a distributed abductive logic programming algorithm is used to coordinate the agents' local reasoning tasks. Through this algorithm, agents compute local conditional answers, by assuming undefined knowledge needed to maintain their (global) constraints, and coordinate their proofs through consistency checks over their respective assumptions. DAREC is the first

* This research is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence.

Technical Communications of the 27th International Conference on Logic Programming (ICLP'11).
Editors: John P. Gallagher, Michael Gelfond; pp. 175–186

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

distributed abductive system that can compute non-ground answers and handle arithmetic constraints. In DAREC, all knowledge is considered public, so, during collaboration, agents can communicate any information they may have. However, in application domains such as simulation of policy-based distributed systems for decentralised policy analysis, confidentiality is an additional primary concern – agents may contain private information that cannot be shared with others during, or after, the collaborative reasoning. This concern imposes an extra challenge to agent interactions during the reasoning process, as agents must decide what to disclose between their communications.

This paper addresses the new challenge by extending the DAREC system to support multi-agent hypothetical reasoning with confidentiality. It provides two main contributions. At knowledge representation level the logical language and the distributed abductive framework have been extended to allow modelling of private agent knowledge. At algorithmic level, the distributed proof procedure has been customised with a *safe* yet efficient agent interaction protocol, which prevents private knowledge being passed between agents and allows some degree of concurrent computation. From the operational point of view, our new distributed abductive algorithm, called DAREC^2 is a coordinated state rewriting process, consisting of *local abductive inferences* by the agents and *coordination* of these local inferences. The local inference is a goal-directed reasoning process, where a current agent (i) solves as many sub-goals of the query as possible, using its own knowledge, and (ii) collects those sub-goals solvable only by other agents, and the constraints that must be satisfied by all agents to guarantee *global consistency* of the final answer. The latter are generated from *constructive negations* and *arithmetic constraints* during the local inference process. They can be reduced to a set of inequalities and arithmetic constraints and be handled by external Constraint Logic Programming (CLP) solvers, enabling also reasoning over unbounded domains. The collected sub-goals and constraints, together with the hypotheses made during the local inference, are encapsulated into a *token state*, which is then passed around to other agents for further processing once all private sub-goals of the current agent have been solved by the agent. This guarantees that confidential information is not included in the token state and not passed to other agents. The coordination of state-passing implements *synchronised backtracking*, whilst enabling concurrent computation between local inferences. The coordination allows two types of agent interaction: *positive* and *negative*. In the case of a positive interaction, the token state is directed to a suitable agent (i.e. who may help to solve some pending sub-goals), whereas for negative interactions, the token state is passed among all agents enforcing each to check the pending constraints. Application dependent strategies may be adopted to interleave/combine such interactions in order to reduce communication overheads. The new system is implemented in Prolog together with a benchmarking test-bed environment. The main intended use of DAREC^2 is as a decoupleable multi-purpose tool for larger multi-agent systems (MAS). For example, each DAREC^2 agent could be implemented as a reactive reasoning module of an agent in a larger MAS to support other agent/system functionalities (e.g., distributed reasoning over BDI agents' belief stores [12]). Alternatively, the whole DAREC^2 system could be implemented as a "simulator" to verify properties of a target MAS, such as the case study example described in this paper for decentralised policy analysis.

The paper is organised as follows. Section 2 discusses related work, Section 3 formalises the notion of a multi-agent abductive reasoning problem, and Section 4 describes the DAREC^2 algorithm. A case study in the context of distributed policy analysis to exemplify the confidentiality aspect in real-world applications is described in Section 5. Finally, conclusion and future work are given in Section 6.

2 Related Work

Distributed abductive reasoning has previously been studied, such as in the ALIAS system [2], the DARE system [7], MARS [1] and DAREC [8]. ALIAS focuses on *local consistency*, i.e., the global answer is consistent with each agent's knowledge individually, where the other three systems focus on *global consistency*, i.e, the global answer is consistent with the union of all the agent knowledge. Both ALIAS and DARE deploy distributed abductive algorithms that are based on the Kakas-Mancarella proof procedure [5], and can only compute ground proofs/answers. MARS uses the consequence finding algorithm [9] for local computation, and is mainly for computing a consistent superset of the agents' existing hypotheses. DAREC, whose distributed abductive algorithm is based on ASystem [10], is the first system that can compute non-ground proofs/answers for possibly unbound domains and with arithmetic constraint satisfaction support. None of these systems considers the confidential aspect of agent knowledge. Our system extends DAREC. In addition to the inherited features, it allows private and public agent knowledge to be explicitly expressed, and guarantees confidentiality during collaborative reasoning. Specifically, the new system introduces *askable literals* that can be shared between agents (i.e., to represent public knowledge), and new local inference rules for solving askable sub-goals. An agent interaction protocol, consisting of special goal selection and agent selection strategies, is enforced to preserve confidentiality while reducing communications and increasing concurrent computation.

In DAREC^2, an askable literal is $A@S$ where A is an atom and S is an agent identifier indicating where the askable sub-goal should be solved. This language feature is most closed to one proposed in *speculative computation* [11]. However, whereas in speculative computation when an askable sub-goal is selected, the agent identifier must be ground, in our system it can also be a variable with quantifier.

3 Knowledge Representation

Standard logic programming notations are employed throughout the paper. We use $\vec{t}$ to represent a vector of arguments. Constraint atoms are those formed with constraint predicates from CLP for a particular domain, such as $\{\in, <, \leq, >, \geq\}$. A clause is either a *rule* $\phi \leftarrow \phi_1, \ldots, \phi_n$ with $n \geq 0$, or a *denial* $\leftarrow \phi_1, \ldots, \phi_n$ with $n > 0$, where $\phi_1, \ldots, \phi_n$ is a conjunction of literals called the *body*, and in the case of a rule ϕ is an atom called the *head*. All variables appearing in a clause are *universally quantified* with the scope the whole clause implicitly, unless stated otherwise. A *query* is a conjunction of literals, whose variables are *existentially quantified* with the scope the whole conjunction implicitly.

In abductive logic programming, predicates of atoms that are not equality or constraint are divided into *abducible predicate* and *non-abducible predicate*. An atom with (non-)abducible predicate is called an *(non-)abducible (atom)*. An abductive (logic programming) framework is a tuple $\langle \Pi, \mathcal{AB}, \mathcal{IC} \rangle$, where Π is a finite set of rules called the *background knowledge*, $\mathcal{AB}$ is a set of abducible predicates, and $\mathcal{IC}$ is a finite set of denials, each of which contains at least one positive abducible, called the *integrity constraints*. Sometimes $\mathcal{AB}$ represents the set of all abducible atoms. Without loss of generality, it is often assumed that no abducible appears in the head of a rule in Π. Therefore, non-abducibles are also called *defined atoms* (or *defined* in brief). Given a query $\mathcal{Q}$, an abductive logic programming task consists of computing an abductive answer $\langle \Delta, \theta \rangle$, where Δ is a set of abducibles, θ is a set of variable substitutions, such that (1) $\Pi \cup \Delta\theta \models \mathcal{Q}$, (2) $\Pi \cup \Delta\theta \models \mathcal{IC}$, where $\models$ is logical entailment under a selected semantics. Δ is called the *hypotheses*, or *explanation*, for the given query (i.e. *observation*).

3.1 Multi-agent Confidential Abductive Framework

In this work, we focus on MAS's with a *fixed* set of agents satisfying the following assumptions:

- Each agent has a *unique ID* and has an *abductive framework*.
- Agents have the same set of abducible predicates, which ensures they can only generate hypotheses that are not defined or *provable* by others.
- Agents can send peer-to-peer messages, i.e., the MAS communication channels form a *fully connected* graph, eliminating the need for considering message routing.

As our current main focus is on the correctness of multi-agent hypothetical reasoning, we further assume that the agents and the communication channels are reliable (i.e. no corruption or loss of messages).

Let's denote an agent's abductive framework with the tuple $\mathcal{F} = \langle \Pi, \mathcal{AB}, \mathcal{IC} \rangle$, where Π and $\mathcal{IC}$ constitute together the agent's *local knowledge*. We may use an agent's identifier, say i, to suffix the agent's abductive framework and its components, i.e., $\mathcal{F}_i$, Π_i, $\mathcal{AB}_i$ and $\mathcal{IC}_i$. Non-abducible predicates of an agent can be *private* or *public (askable)*. The former are defined only in the agent background knowledge and *cannot* appear in the body of a rule of any other agent[1]. Public predicates can only be defined in the agent background knowledge, but *can* appear in the body of rules of other agents.

▶ **Definition 1.** An **askable** atom is $p(\vec{u})@ID$ where $p(\vec{u})$ is a non-abducible atom and ID is the agent identifier where the predicate is defined.

Askable atoms $p(\vec{u})@ID$ that appear as the head of a rule in an agent's background knowledge have their ID ground with the agent's identifier. Askable atoms that appear in the body of a rule may have an unground agent identifier. The ID of an askable atom is therefore more than just a syntactic alias. It denotes a variable that can be (appropriately) existentially and universally quantified. Syntactically, an askable can be seen as a non-abducible with an extra agent identifier argument. The negation of an askable $\neg p(\vec{u})@ag$ is read as $\neg(p(\vec{u})@ag)$, meaning "$p(\vec{u})$ should not be provable by agent ag".

▶ **Definition 2.** The **global abductive framework** is $\langle \Sigma, \widehat{\mathcal{F}} \rangle$, where Σ is the set of all agent identifiers and $\widehat{\mathcal{F}}$ is the set of agent abductive frameworks, i.e. $\{\mathcal{F}_i \mid i \in \Sigma\}$. For any pair of agents $i, j \in \Sigma$, $\mathcal{AB}_i = \mathcal{AB}_j$.

▶ **Definition 3. Global Abductive Answer** Given a global abductive framework $\langle \Sigma, \widehat{\mathcal{F}} \rangle$ and a query $\mathcal{Q}$, let $\widehat{\Pi} = \bigcup_{i \in \Sigma} \Pi_i$, let $\widehat{\mathcal{IC}} = \bigcup_{i \in \Sigma} \mathcal{IC}_i$, and let $\widehat{\mathcal{AB}} = \bigcup_{i \in \Sigma} \mathcal{AB}_i$. A pair $\langle \Delta, \theta \rangle$ is a global abductive answer for $\mathcal{Q}$ if and only if:

$$(1)\ \Delta\theta \subseteq \mathcal{AB};\ (2)\ \widehat{\Pi} \cup \Delta\theta \models \mathcal{Q}\theta;\ (3)\ \widehat{\Pi} \cup \Delta\theta \models \widehat{\mathcal{IC}}$$

where θ is the variable substitutions over the variables in $\mathcal{Q}$, and $\models$ is the logical entailment of a selected semantics for the logic program formed by $\widehat{\Pi} \cup \widehat{\mathcal{IC}}$.

4 Distributed Algorithm

Given a global abductive framework (of a MAS) and a query, agents must collaborate to compute global abductive answers, while keeping there private information confidential.

[1] Name clashes between private predicates of different agents are assumed to have been resolved.

▶ **Definition 4.** The collaboration of agents to compute global abductive answers for a given query is a *confidential reasoning* if and only if no private predicate and its definitions of any agent can be seen or inferred by another agent during or after the collaboration.

The DAREC[2] distributed abductive algorithm satisfies this property. Operationally, the distributed computation is a sequence of coordinated local abductive computations, i.e., *distributed abduction = local abduction + coordination.*

4.1 Local Abduction

Local abduction is a top-down (goal-directed) abductive inference, which can be described as a state rewriting process. A *state* contains intermediate computational results, and has four components: (1) a *remaining goals store* – a goal is either a literal or a denial "$\forall\vec{X} \leftarrow \phi_1, \ldots, \phi_n$" ($n > 0$), where $\vec{X}$ is the set of variables in the $\phi_1, \ldots, \phi_n$ that are universally quantified with the scope the whole denial, (2) a *hypotheses store* containing a set of collected abducibles, (3) a *constraints store* containing a set of collected and consistent inequalities and arithmetic constraints, and (4) a *denials store* containing collected denial goals whose left-most literal in the body is an abducible. These denials represent the *conditions* for collecting the instances of their left-most abducibles and are collected during the inference. A state also contains *abducible tagging* information – an abducible in the hypotheses store is *tagged* by an agent's identifier if the agent **has not** checked it against its integrity constraints. All free variables in a state are existentially quantified with the scope the whole state implicitly. A *solved* state is one that has *no remaining goal* and *no tagged abducible*. At each inference step, a goal is selected from a non-solved state, and a literal is selected from the goal if the goal is a denial. *Safe* goal selection strategies are adopted, in which the *current agent* (i.e. the one performing the local abduction) can select:

- an askable goal, only if its agent ID is the current agent's ID or a variable;
- an askable from a denial goal, only if its agent ID is the current agent's ID or a variable;
- an abducible from a denial goal, only if there is no private non-abducible literal in the denial goal;
- inequalities, arithmetic constraints and negative literals, only if they do not contain universally quantified variables;

The first three requirements guarantee the *confidentiality* (see later), whereas the last requirement avoids *floundering*. The next state is obtained after applying a *local inference rule* to the selected goal. These rules are based on the ASystem [6] inference rules, with extensions to handle askables and tagging information, where ag is the current agent's ID:

- if an askable goal $p(\vec{u})@ID$ is selected:
  - if ID is ag, it is resolved with a rule in the agent's background knowledge;
  - otherwise, ID must be an existentially quantified variable, and the goal can be replaced with either $p(\vec{u})@ag$, if $ID = ag$ is satisfiable, or with $ID \neq ag, p(\vec{u})@ID$. Semantically, this means that the askable can be either solved by the current agent, or by a different agent (later).
- if an askable $p(\vec{u})@ID$ in a denial goal is selected:
  - if ID is ag, it is reduced as being non-abducible in a denial goal;
  - if ID is an existentially quantified variable, then either the denial goal is replaced with one obtained by replacing the askable with $p(\vec{u})@ag$ if $ID = ag$ is satisfiable, or the denial goal is kept but an inequality goal $ID \neq ag$ is added. Semantically, this means that the denial can be either solved by the current agent on $p(\vec{u})@ag$, or by a different agent ag' on $p(\vec{u})@ag'$ (later);

  - if ID is a universally quantified variable with the scope the denial, then the denial goal is replaced by the set of denial goals obtained by binding ID to all the agent identifiers. Semantically, this means the denial goal must be solved by all the agents.
- if an abducible is collected to the hypotheses store by the current agent, it is tagged with all other agents' identifiers;
- if there is an abducible in the hypotheses store that is tagged by the current agent's identifier, the set of denial goals generated by resolving the abducible with the current agent's integrity constraints is added to the remaining goals store, and the tag is removed.

Each inference step may result in zero or more next states. A state is called a *transitional state* if any of the following conditions is satisfied: (1) it contains only askable goals, or denial goals consist of only askables, where none of the askables has an agent ID equal to the current agent's identifier; (2) it has no remaining goal but at least one collected abducible is tagged. A state is called a *failure state* if it is not a solved or transitional state, and no local inference rule is applicable to a selected goal or no goal can be selected. Thanks to the safe goal selection strategies, the whole local state rewriting process can be visualised as a *local abductive derivation tree*, whose leaf states can only be failure, transitional and solved states.

4.2 Coordination

When a query is received by an agent, the agent starts a local abduction with an *initial state*, whose pending goals are the literals in the query and all other stores are empty. If a transitional state is derived, it can be passed to a *suitable* agent for further processing (i.e., the recipient agent will start another local abduction with the state). If a solved state is derived, an answer $\langle \Delta, \theta \rangle$, where Δ is the set of collected abducibles and θ is the set of variable substitutions induced by the constraints store, is returned to the query issuer.

4.2.1 Transitional States and Confidentiality

A transitional state is one that can be passed between agents. It can be seen as a specification of *agent collaboration* – intuitively, the hypotheses store and the constraints store record the partial answer, the pending goals are the remaining reasoning tasks to be solved by relevant agents, the denials store includes the *global integrity constraints* that must not be violated by the agents during their reasoning, and finally the abducible tagging information is used to ensure consistency checks on the abducibles by all the agents, which may themselves result in new global integrity constraints. Under the safe goal selection strategies (1) an agent is forced to process a state as much as possible, and (2) no private non-abducible predicates can appear in transitional states, so preserving confidentiality.

4.2.2 State Recipient Agent Selection

When a transitional state is derived by an agent, a recipient agent is identified so that the state can be passed on for further processing. This uses a *helpfulness ranking* algorithm. We say "an agent ag may help with an askable $p(\vec{u})@ID$" if ID is ag's identifier or ID is a variable and $ID = ag$ is satisfiable; and we say "an agent ag may help with a (collected) abducible" if the abducible is tagged with ag's identifier. Given a transitional state, we compute the *helpfulness* of each agent in the system by summing up the total number of askable goals, denial goals (containing an askable) and abducibles that the agent may help with, and sort the agents according to the their helpfulness. One of the agents with highest value of helpfulness is then selected as the state recipient. However, it may occur that no

agent can help with a transitional state (i.e., agents' helpfulness value is 0). This may happen when the state contains an askable goal, $p(\vec{u})@ID$, with ID being a variable and has been passed around all agents once but no agent is able to solve it by bounding ID to its own identifier. In this case, the transitional state is treated as a failure state.

4.2.3 Backtracking with Concurrent Computation

Each local abduction constructs a local abductive derivation tree in search for solved states. If a transitional state is derived, this is sent out and another local abduction is initiated by a different agent. It can be shown that merging all local derivation trees would give a *global abductive derivation tree* equivalent to the one that would be constructed by the ASystem algorithm for the same query but over the merged agents' background knowledge and integrity constraints. This is, in fact, the formal basis for the correctness of our distributed algorithm. However, in practice such centralised reasoning is not always feasible and from an operational perspective we are interested in coordinations of local abductions that strike a balance between performance and communication. In our system, after an agent sends a transitional state to another agent, it can continue its local abduction while the recipient is working on a different local abductive reasoning task. Agents then essentially construct and explore different parts of the global derivation tree concurrently, providing a better performance than a fully sequential reasoning process. Each agent may derive more than one transitional state during its local abduction. If transitional states are sent as soon as they are derived, the communication channels between agents may quickly "flood" and agents become overloaded, as they may receive several states and perform several unfinished local abductions at the same time. To address these issues without sacrificing concurrent computation, a token based *synchronised backtracking* coordination strategy is adopted:

1. when the query issuer sends a query to an agent, it also sends a *token*. The agent creates the initial state and starts a local abduction;
2. during the local abduction, if the agent derives a transitional state,
   a. if it has the token, then it sends out the transitional state with the token (i.e., it will no longer have the token);
   b. otherwise, it buffers the transitional state;

   and in both cases it continues the local abduction;
3. once an agent receives a transitional state and the token, it initiates a local abduction and keeps the token;
4. if an agent derives a solved state, it sends the extracted answer to the query issuer regardless if it has the token or not, and continues the local abduction (i.e., to search for more solutions);
5. if an agent finishes a local abduction (i.e., finishes constructing and exploring a local abductive derivation tree), then
   a. if it has the token, it returns the token as a *backtracking* signal to the agent who passed the transitional state;
   b. otherwise, it waits for the backtracking signal;
6. after an agent receives a backtracking signal (i.e., it regains the token),
   a. if there are buffered transitional states, then one of them is sent out with the token (i.e., the agent loses the token again);
   b. otherwise, the agent stores the token and continues the local abduction if it has not yet completed, or the agent sends a backtracking signal as in 5(a) if finished;
7. if the query issuer no longer needs further answers, it sends a *discard* message to all the agents, so they will stop all relevant local abductions and remove relevant buffered states.

Note that with such a coordination strategy, an agent may still receive a transitional state while it still has some unfinished local abduction. Because of the *synchronised backtracking*, the current local abduction computation can be interrupted until a new local abduction is finished, and then resumed.

4.2.4 Soundness and Completeness

The distributed algorithm is *sound* and *complete* only with respect to a three-valued semantics [13] for which, given a global query $\mathcal{Q}$ and a global abductive answer $\langle\theta, \Delta\rangle$, the interpretation (completion) of all abducibles (i.e. $\mathcal{AB}$) is defined as $I_{\Delta\theta} = \{A^{\mathbf{t}} \mid A \in \Delta\theta \wedge A \in \mathcal{AB}\} \cup \{A^{\mathbf{f}} \mid A \notin \Delta\theta \wedge A \in \mathcal{AB}\}$.

▶ **Theorem 5. (Soundness)** *Given a global abductive framework $\langle\Sigma, \widehat{\mathcal{F}}\rangle$ and a global query $\mathcal{Q}$, if there is a successful global derivation for $\mathcal{Q}$ with answer $\langle\Delta, \theta\rangle$, then 1. $\bigcup_{i\in\Sigma}\Pi_i \cup I_{\Delta\theta} \models_3 \mathcal{Q}\theta$, and 2. $\bigcup_{i\in\Sigma}\Pi_i \cup I_{\Delta\theta} \models_3 \bigcup_{i\in\Sigma}\mathcal{IC}_i$, where $\models_3$ is the logical entailment under the three-valued semantics for abductive logic programs [13].*

▶ **Theorem 6. (Completeness)** *Let $\langle\Sigma, \widehat{\mathcal{F}}\rangle$ be a global abductive framework and $\mathcal{Q}$ a global query, suppose there is a finite global derivation tree T for $\mathcal{Q}$. If $\bigcup_{i\in\Sigma}\Pi_i \cup \bigcup_{i\in\Sigma}\mathcal{IC}_i \cup \exists\mathcal{Q}$ is satisfiable under the three-valued semantics, then T contains a successful branch.*

Our DAREC[2] system is a customisation of the DAREC system, i.e., special goal selection and agent interaction strategies are adopted. With these strategies, askable atoms can be seen as normal non-abducible atoms where the agent ID argument has the set of (the identifiers) the agents as domain. The DAREC system is sound and complete with respect to any goal selection and agent interaction strategies, and therefore our system inherits these properties (the reader is referred to [8] for outline proofs). The choice of three-valued completion semantics instead of other stronger semantics (e.g. stable model semantics [4]), is because top-down inference procedures, like abduction, may suffer from looping. In practice, looping can be avoided either by implementing a depth-bounded search strategy or by ensuring that the overall logic program satisfies certain properties (e.g., abductive non-recursive [14]).

4.2.5 Implementation

Our system has been implemented with YAP Prolog 6 [2]. An inequality solver extending the standard unification algorithm has also been implemented. For example, given $f(1, p(X)) \neq f(Y, p(2))$, the solver will answer $1 \neq Y$ or $X \neq 2$. The system uses the finite domain constraint solver (`CLP(FD)`) by YAP and the inequality solver for handling the arithmetic constraints and inequalities during the local inferences.

Agents in the system use TCP messages for peer-to-peer communications. For optimisation purposes, the system allows the option of using a "yellow page directory". When enabled, each agent maintains a copy of the directory to record *agent advertisements*. An advertisement contains an agent's identifier, the set of askables defined by the agent, and the set of abducibles *regulated* by the agent (i.e., the abducible that appears in an integrity constraint of the agent). When a new agent joins the system, it broadcasts its advertisements, so they are added to all other agents' directory. When an agent leaves, its advertisements are removed from everyone's directory. The directory can be used to further reduce communications and local computations: (1) abducibles no longer need to be checked (tagged) by agents that do

[2] `http://www.dcc.fc.up.pt/~vsc/Yap/`

not regulate them; (2) denials, with an askable whose agent ID is a universally quantified variable, no longer need to be checked by agents that do not know the askable; (3) in deciding recipient agents for a transitional state, agents that do not define the askable in question, are no longer considered. Note that the directory does not contain private atoms of any agent, and therefore its use does not affect the confidentiality property of the system.

5 Distributed Policy Analysis

A policy analysis framework has been proposed in [3]. The framework provides a specification language to represent security policies (*authorisations* and *obligations*), and system domain, as normal logic programs. Various policy analysis tasks can be solved using abduction. Modality conflict, for instance, can be defined as an abductive reasoning task where the goal is the negation of the property to analyse and the conditional answer is an example of system execution that proves this goal, i.e. a counterexample to the property. An example of analysis of *separation of duty* (SoD) in the context of security policies distributed over networks is given below.

▶ **Example 7.** In a Role-based Access Control (RBAC) system, the permission of an action depends on the role(s) assigned to the subject. Assume a RBAC corporate network with one team agent ($team1$) and two administrator agents ($admin1$ and $admin2$). The team1 is responsible for taking orders which can only be completed by an *initiator* and a *verifier* together, and who cannot be the same person, (i.e. the SoD concept). Before team1 can accept an order, it needs to check whether it can have clerks with suitable roles available to perform the two actions. The role assignments can only be performed by independent administrators, whose duties/privileges are also separated – they manage different role assignments and have without centralising the knowledge of the agents.

A distributed network, such as in the example, often consists of multiple nodes, each of which may have its own policies that are not shareable with others, but which may depend on each other. Centralised policy analysis for the whole network is not possible because of the confidentiality concern. Our system can be applied directly to this class of problems as a simulator of the distributed network.

Let's elaborate the example further. For simplicity, we only describe the policies relevant to the aforementioned analysis tasks. The team agent has local knowledge about team members and administrators, as well as the effects of roles assignments. It also has a rule specifying that role assignments are decided by the administrators. The system domain is *dynamic*, as executed actions may change its properties, and it is modelled using a set of domain independent Event Calculus axioms. Thus, Π_{team1} contains at least the following rules (by convention, Su, Ta, Ac and F are variables of the sorts *subjects*, *targets*, *actions* and *fluents* respectively, and $T, T_1, T_2, \ldots$ are variables of the sort *time*):

$$
\begin{aligned}
&holds(F,T) \leftarrow initially(F), 0 < T, \neg clipped(0,F,T).\\
&holds(F,T) \leftarrow do(Su,Ta,Ac,T1), initiates(Su,Ta,Ac,F,T1),\\
&\qquad\qquad 0 < T1, T1 < T, \neg clipped(T1,F,T).\\
&clipped(T1,F,T) \leftarrow do(Su,Ta,Ac,T2), terminates(Su,Ta,Ac,F,T2),\\
&\qquad\qquad T1 < T2, T2 < T.\\
&do(Su,Ta,Ac,T) \leftarrow Ac = assign(Role),\\
&\qquad\qquad request(Su,Ta,Ac,T), permitted(Su,Ta,Ac,T)@Su.\\
&initiates(Su,Ta,assign(Role),hasRole(Ta,Role),T) \leftarrow\\
&\qquad\qquad holds(clerk(Ta),T), holds(admin(Su),T).\\
&holds(clerk(X),T) \leftarrow X \in \{alex, bob\}.\\
&holds(admin(X),T) \leftarrow X \in \{admin1, admin2\}.\\
&holds(canInitiateOrder(X),T)@team1 \leftarrow holds(hasRole(X,initiator),T).\\
&holds(canVerifyOrder(X),T)@team1 \leftarrow holds(hasRole(X,verifier),T).
\end{aligned}
$$

One of the administrators, $admin1$, can assign the `verifier` role to users. It has a *blanket* policy rule *"a user can be assigned a role if it is permitted by at least one local policy"*. Blanket policy rules that can be used for modelling conflict resolutions are also private to the agent, e.g., in $admin1$, positive authorisation policies (rules with *permitted* as head) has higher priority than negative authorisation policies (rules with *denied* as head). In addition, $admin1$ maintains a local *trust level* database of the users and has a local positive authorisation policy *"the* `verifier` *can be assigned to a user if the user's trust level is greater than 4"*. Thus, Π_{admin1} contains at least the following rules:

$$permitted(admin1, Ta, assign(Role), T)@admin1 \leftarrow$$
$$\quad holds(managed\_role(Role), T), permitted(admin1, Ta, assign(Role), T).$$
$$permitted(admin1, Ta, assign(verifier), T) \leftarrow holds(trust\_level(Ta, L), T), L > 4.$$
$$holds(trust\_level(alex, 3), T).$$
$$holds(trust\_level(bob, 5), T).$$
$$holds(managed\_role(verifier), T).$$

The other administrator, $admin2$, can assign the `initiator` role to users. It has a different blanket policy rule *"a user can be assigned a role if none of the local policies denies it"* (i.e., negative authorisation has higher priority). It also tries to implement SoD by having a local negative authorisation policy *"a user cannot be assigned to the* `initiator` *role if it has been assigned the conflicting* `verifier` *role"*. Thus, Π_{admin2} contains at least of the following:

$$permitted(admin2, Ta, assign(Role), T)@admin2 \leftarrow$$
$$\quad holds(managed\_role(Role), T), \neg denied(admin2, Ta, assign(Role), T).$$
$$denied(admin2, Ta, assign(initiator), T) \leftarrow$$
$$\quad holds(conflicting(initiator, Role), T), request(Su, Ta, assign(Role), T1), T1 < T.$$
$$holds(managed\_role(initiator), T).$$
$$holds(conflicting(initiator, verifier), T).$$

To check if team1 can fulfil a task with the administrators under the SoD constraint, we can use the query $\exists X, Y, T, Z.[holds(canInitiateOrder(X), T)@Z, holds(canVerifyOrder(Y), T)@Z \wedge X \neq Y]$. Our system can succeed the query and find one answer: $Ans_1 = request(admin1, bob, assign(verifier), T1) \wedge request(admin2, alex, assign(initiator), T2) \wedge T1 < T \wedge T2 < T \wedge Z = team1$. To check if the existing administrators' policies can guarantee the SoD security property of the overall system, i.e., *"it is not possible that someone can complete an order alone"*, we can use the (negated) query $\exists X, T, Z.[holds(canInitiateOrder(X), T)@Z, holds(canVerify\text{-}Order(X), T)@Z]$. Our system can also succeed this query and find one answer: $Ans_2 = req(admin1, bob, assign(verifier), T1) \wedge req(admin2, bob, assign(initiator), T2) \wedge T1 \leq T2 \wedge T2 < T \wedge Z = team1$. Therefore, the existing policies are not sufficient to guarantee the SoD property, and the administrators have to revise their policies.

5.1 Benchmarking

To study the scalability of DAREC[2] in distributed policy analysis, an auto-testing environment has been developed. Given a set of tunable parameters, the environment is able to generate policy rules and action effect rules with randomised conditions conforming to the language in [3]. These rules are then distributed among a specified number of agents. Note that the language in [3] guarantees the overall system model (as a logic program) is abductive acyclic and hence the executions of policy analysis queries always terminate. For each distributed computation, the total number of messages exchanged between agents, the average ping time and the total time for computing all solutions for a given query are recorded. For example, empirical results (e.g., 5 agents, each having about 250 rules; each rule body has on

average 10 conditions, 3 of which are askables) showed that for about two thirds of the tested queries, the DAREC[2] computation is about 1.26∼6.17 times faster than a computation using abduction over the centralised rules with an average ping time between agents being 0.015ms. This was expected as concurrent computation was performed during collaborative reasoning.

6 Conclusion and Future Work

Confidentiality in knowledge is one important constraint that makes a multi-agent reasoning problem challenging, and it is also a very common assumption in MAS's. The main contributions of this paper include (1) a logical framework for modelling the distributed knowledge of a multi-agent system where the agent knowledge bases are correlated and have private information, and (2) a top-down distributed abductive algorithm which allows agents to perform collaborative hypothetical reasoning without disclosing private information. By limiting the set of abducible predicates to be empty, the system becomes a general purpose distributed deductive theorem prover that performs constructive negation. This feature is very useful when dealing with logic programs with unbounded domains (e.g., Π_{admin2} in Example 7) that cannot be implemented using bottom-up algorithms like answer set programming. The system has many potential applications including distributed security policy analysis. As a future work, we aim to perform more benchmarking to investigate the performance under different safe goal selection strategies, agent selection strategies and agent interaction strategies, and extend our system to handle private abducible predicates.

References

1 G. Bourgne, K. Inoue, and N. Maudet. Abduction of distributed theories through local interactions. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 901–906, 2010.

2 A. Ciampolini, E. Lamma, P. Mello, F. Toni, and P. Torroni. Cooperation and competition in ALIAS: a logic framework for agents that negotiate. *Annals of Mathematics and Artificial Intelligence*, 37(1–2):65–91, 2003.

3 R. Craven, R. Lobo, J. Ma, A. Russo, E.C. Lupu, and A. Bandara. Expressive policy analysis with enhanced system dynamicity. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 239–250, 2009.

4 M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference of Logic Programming*, 1988.

5 A.C. Kakas and P. Mancarella. Abductive logic programming. In *Proceedings of the Workshop Logic Programming and Non-Monotonic Logic*, pages 49–61, 1990.

6 A.C. Kakas, B. Van Nuffelen, and M. Denecker. A-system: Problem solving through abduction. In *Proceedings of 17th International Joint Conference on Artificial Intelligence*, pages 591–596, 2001.

7 J. Ma, A. Russo, K. Broda, and K. Clark. DARE: a system for distributed abductive reasoning. *Journal of Autonomous Agents and Multi-Agent Systems*, 16(3):271–297, 2008.

8 J. Ma, A. Russo, K. Broda, and E. Lupu. Distributed abductive reasoning with constraints. In *Post-proceedings of the 8th International Workshop on Declarative Agent Languages and Technologies*, 2010.

9 H. Nabeshima, K. Iwanuma, K. Inoue, and O. Ray. Solar: An automated deduction system for consequence finding. *AI Communications*, 23(2-3):183–203, March 2010.

10 Bert Van Nuffelen. *Abductive Constraint Logic Programming: Implementation and Applications*. PhD thesis, Department of Computer Science, K.U.Leuven, 2004.

11 K. Satoh, K. Inoue, K. Iwanuma, and C. Sakama. Speculative computation by abduction under incomplete communication environments. In *Proceedings of the 4th International Conference on Multi-Agent Systems*, pages 263–270, 2000.

12 M.P. Sindlar, M.M. Dastani, F. Dignum, and J.C. Meyer. Mental state abduction of bdi-based agents. pages 161–178, 2009.

13 Frank Teusink. Three-valued completion for abductive logic programs. *Theoretical Computer Science*, 165(1):171–200, 1996.

14 Sofie Verbaeten. Termination analysis for abductive general logic programs. In *International Conference on Logic Programming*, pages 365–379, 1999.