Distributed Abductive Reasoning with Constraints (Extended Abstract)

Jiefei Ma Alessandra Russo Krysia Broda Emil Lupu
Department of Computing, Imperial College London, United Kingdom, SW7 2AZ
{jm103,ar3,kb,ecl1}@doc.ic.ac.uk

ABSTRACT

Abductive inference has many known applications in multi-agent systems. However, most abductive frameworks rely on a centrally executed proof procedure whereas many of the application problems are distributed by nature. Confidentiality and communication overhead concerns often preclude transmitting all the knowledge required for centralised reasoning. We present in this paper a novel multi-agent abductive reasoning framework underpinned by a flexible and extensible distributed proof procedure that permits collaborative abductive reasoning with constraints between agents over decentralised knowledge.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms

Keywords

Multi-agent Reasoning, Abductive Logic Programming

1. INTRODUCTION

Abductive reasoning is a powerful inference mechanism that can generate conditional proofs. The combination of Abduction and Logic Programming (ALP) [5] has many known applications, such as planning, scheduling, cognitive robotics, medical diagnosis and policy analysis [3]. However, most abductive frameworks [6, 4, 7] rely on a centrally executed proof procedure whereas many of the application problems are distributed by nature. Confidentiality and communication overhead concerns often preclude transmitting all the knowledge required for centralised reasoning. Recently, ALIAS [1] and DARE [8] have shown how to distribute abductive computation in a collaborative system. However, their distributed proof procedures, which are based on the well-known Kakas-Mancarella procedure [6], do not support *constructive negation* [10] and cannot compute non-ground conditional proofs. Hence they cannot be used for applications such as scheduling and planning involving time and cost, which require constraint processing [11].

In order to overcome this limitation, we propose DAREC, a distributed abductive logic programming framework underpinned by a general and customisable proof procedure that permits collaborative abductive reasoning with constraint processing between decentralised (computational logic) agents. The collaborative reasoning can be seen as a state rewriting process, where the *reasoning state*, initially containing just the query, is exchanged between the agents. Agents rewrite the state during their local inference by adding relevant (non-ground) assumptions and (dynamically generated) constraints that can then be checked for global consistency. The answer can then be extracted from the final state when the overall reasoning succeeds. DAREC can be applied to both *closed* and *open* multi-agent systems, i.e. whether the set of agents is fixed or changes dynamically.

Cite as: Distributed Abductive Reasoning with Constraints (Extended Abstract), Jiefei Ma, Alessandra Russo, Krysia Broda and Emil Lupu, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 1381-1382

2. FRAMEWORK

Let $\mathcal{L}$ be a predicate logic language consisting of three disjoint sets of predicates: *abducible*, *non-abducible* and *constraint*. Each agent in DAREC has a unique identifier *id* and can be modeled with an *abductive framework* $\mathcal{F}_{id} = \langle \Pi, \mathcal{AB}, \mathcal{I} \rangle$, where Π is a constraint logic program called the *background knowledge*, $\mathcal{AB}$ is a set of abducible predicates, and $\mathcal{I}$ is a set of *integrity constraints* of the form $\leftarrow B$ where B is a conjunction of literals over $\mathcal{L}$. Given a set Σ of such agents, the DAREC framework $\mathcal{F}^{dis}$ is $\langle \Sigma, \Pi^{dis}, \mathcal{AB}, \mathcal{I}^{dis} \rangle$, where (a) $\Pi^{dis} = \bigcup_{i \in \Sigma} \Pi_i$, (b) $\mathcal{I}^{dis} = \bigcup_{i \in \Sigma}$, and (c) $\mathcal{AB} = \mathcal{AB}_i = AB_j$ for any $i, j \in \Sigma$, which implies that all the agents agree on the same set of abducibles. Although semantically Π^{dis} and $\mathcal{I}^{dis}$ are the unions of all the agent background knowledge and integrity constraints respectively, they are not physically centralised.

Given a query $\mathcal{Q}$ (a conjunction of literals over $\mathcal{L}$ called the *goals*), a DAREC answer w.r.t. $\mathcal{F}^{dis}$ is a pair $\langle \Delta, \theta \rangle$, such that: (i) Δ is a set of abducible atoms, θ is a set of variable substitutions; (ii) $\Pi^{dis} \cup \Delta \models \mathcal{Q}\theta$; and (iii) $\Pi^{dis} \cup \Delta \models \mathcal{I}^{dis}$. Thus, condition (iii) defines the requirement of *global consistency* – the answer is consistent with the overall agent integrity constraints. This is different from ALIAS where only local consistency is guaranteed, i.e. the answer is consistent with each agent's integrity constraints locally.

3. PROOF PROCEDURE

The DAREC proof procedure assumes that agents (1) execute the proof procedure docilely when requested, and (2) can find out what non-abducibles are known by (i.e. defined in) other agents. The former ensures that the agents *are*

willing to cooperate and will not sabotage the collaboration. The latter enables the agents to *know how to cooperate* (i.e. who to ask for help). To achieve this, we may simply assume the availability of a "yellow page" directory, which is accessible by all agents and records what non-abducible predicates are defined by who. Such assumption can be removed when protocols such as *auction* or *contract net* are adopted.

When one agent receives an initial query, an initial *global state* is created. The state can then be passed around the set of agents like a "token". There are two main components of the state: the set of *remaining goals* (i.e. initially containing only the query) and an (initially empty) *store* for accumulating the intermediate computational results, such as the abducibles and constraints collected along the collaboration, as well as the dynamic integrity constraints (called *denials*) generated for the global *negation as failure* (NAF) [2] process. In addition, each agent may *tag* a non-abducible goal, an abducible or a denial in the state with its identifier. This meta-data is also passed along with the state and its usage will be described shortly. The state can be possessed by one agent at any time, and can be modified by the agent through abductive inference with ten *local inference rules*, which are extended from those of ASystem [7]. In addition, there are three *transitional inference rules*, namely *TR* (*Receive State*), *TH* (*Request for Help*) and *TC* (*Request for Check*), used by an agent to process a received state or a state to be sent out. For instance, when an agent is stuck at reducing a non-abducible goal locally, i.e. due to its lack of knowledge, it may pass the state to another agent (e.g. discovered from the directory) for help, by the TH rule. Furthermore, such state passing may be postponed by the agent: if there are outstanding goals, the non-abducible goal can be temporarily delayed and the agent can continue with the local inference. Whenever a new abducible or denial is collected, it must be checked eventually by all agents in order to ensure global consistency. This is enforced by the TC and TR rules: the former urges an agent to pass the state to another who has not checked the new abducibles or denials, and the latter presses the recipient to resolve the abducibles and denials with its local knowledge and integrity constraints, which may subsequently generate new goals. Thus, the tagging meta-data in the state can be used to prevent the same goal being delayed and avoid the same abducible/denial being checked twice by the same agent. The collaboration terminates and succeeds when a *solved state*, in which no goal remains and all abducible/denials have been checked by all agents, is obtained. A DAREC answer can then be extracted from the solved state.

The DAREC proof procedure is sound and complete w.r.t. the three-valued semantics for abductive logic programs [9]. A working prototype has been implemented in Prolog [1].

4. EXTENSIONS

Depending on the application needs, we consider two extensions for the DAREC framework described: *separation of global and local non-abducibles*, and *support for open MAS*. With the former, agents can model *private* knowledge using the local non-abducibles, whose global NAF processes are not needed. With the latter, agents are allowed to join or leave the collaboration at will. To achieve this, the global state also records the set of agents that have ever possessed it. Whenever an agent from the set leaves, the collaboration must "backtrack" from the point when the agent was first sent the state.

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

5. CONCLUSION

Token passing is a simple but effective way of coordinating agent collaboration. Unlike DARE and ALIAS, the DAREC proof procedure does not interleave between the *abductive* and *consistency* derivations. It is more flexible as it allows the agents to decide when to request for help and consistency check, which can reduce communications and result in performance gain. The information passed between agents is minimal, as only the exchangeable tasks (i.e. goals), partial answer (i.e. the abducibles) and the consistency constraints (i.e. denials) are kept in the global state token. The tagging meta-data is merely recorded and used for efficiency.

As future work we will extend DAREC to allow also local abducibles, which agents can collect to form the global answer but do not pass to others. Since we aim to use DAREC for real distributed applications deployed on resource constrained devices such as PDAs and smart-phones, we also aim to benchmark different rule and goal selection strategies to evaluate their impact on scale-down to small devices and scale-up to many agents and rules.

6. ACKNOWLEDGMENTS

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.

7. REFERENCES

[1] 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 Math. and AI*, 37(1–2):65–91, 2003.

[2] K. Clark. Negation as failure. *Logic and Databases*, pages 293–322, 1978.

[3] R. Craven, J. Lobo, J. Ma, A. Russo, E. Lupu, A. Bandara, S. Calo, and M. Sloman. Expressive policy analysis with enhanced system dynamicity. In *ASIACCS 09*, 2009.

[4] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. In *In Proc. JELIA04*, pages 31–43. Springer, 2004.

[5] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[6] A. C. Kakas and P. Mancarella. Abductive logic programming. In *LPNMR*, pages 49–61, 1990.

[7] A. C. Kakas, B. V. Nuffelen, and M. Denecker. A-system: Problem solving through abduction. In *IJCAI*, pages 591–596, 2001.

[8] J. Ma, A. Russo, K. Broda, and K. Clark. DARE: a system for distributed abductive reasoning. *Journal of AAMAS*, 16(3):271–297, 2008.

[9] F. Teusink. Three-valued completion for abductive logic programs. In *ALP: Proc. of the 4th Int. Conf. on Algebraic and logic programming*, pages 171–200, 1996.

[10] M. Wallace. Negation by constraints: A sound and efficient implementation of negation in deductive databases. In *SLP*, pages 253–263, 1987.

[11] M. Wallace. Constraint logic programming. In *Computational Logic: Logic Prog. and Beyond*, pages 512–532. Springer-Verlag, 2002.