Espressing Collaborative and Competitive Coordination among Abductive Agent: ALIAS framework and LAILA language

Marco Battarra

Sistemi Multi Agente

2008/2009

Introduction

- Paradigm shift: away from building stand-alone applications from scratch, to the development of distributed systems built largely from pre-existing components
- “Agentification”
  - The inner world: agent architecture , modeling of agent knowledge and reasoning
  - The social life: communication , coordination
- Our approach
  - The interaction patterns: collaboration and competition
  - The reasoning: abduction
  - The framework: ALIAS
  - The language: LAILA

Interaction patterns

- Collaboration and competition have been recognized as possible interaction patterns in diagnostic processes carried out by multiple agents
- Let us consider, as an example for the collaborative case, a group of medical doctors, each one expert in a particular area, who must collaborate to formulate a diagnosis for a given set of symptoms. Each expert can be modeled by an agent whose task is to find a disease as an explanation for a sub-part of the symptoms area. In some cases, the hypotheses raised by one agent may be inconsistent with those raised by a collaborating agent
- As an example of the competitive case, different, alternative diagnoses can thus be proposed by the different agents, and the best one can be chosen according to some policy; for instance, the one with a greater rate of incidence, or the most plausible one with respect to the patient's clinical history

Abduction for agent reasoning

- The scenery: knowledge-intensive distributed application
- Agents need a guess about distributed goal which cannot be solved locally since the local knowledge is incomplete
- The CWA usually adopted in logic programming can be no longer assumed, and some form of open reasoning has to be considered
- Abduction has been widely recognized as a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge
- Incomplete knowledge is handled by labeling some pieces of information as abducibles, i.e., possible hypotheses which can be assumed to enlarge the agent's knowledge, provided that they are consistent with the current knowledge base.
- In a multi-agent environment, abductive reasoning requires a form of coordination among agents in order to guarantee that abduced hypotheses are assumed consistently.

Abductive logic programs

- A triple <P, A, IC>
  - P is a logic program, possibly with abducible atoms in clause bodies
  - A is a set of abducible predicates
  - IC is a set of integrity constraints (denials containing at least one abducible)
- Declarative semantics

$$\begin{cases} P \cup \delta \models G \\ P \cup \delta \text{ “}satisfies\text{”} \ IC \\ \delta \subseteq \mathcal{A} \end{cases}$$

Given an abductive program and a formula G, the goal of abduction is to find a (possibly minimal) set of atoms δ that belongs to A which together with P *entails* G. It is also required that the program P ∪ D *satisfies* IC.

Abduction proof procedure

- We adopt *Kakas-Mancarella* proof-procedure for abductive derivation
- Intuitively, an abductive derivation is the usual logic programming derivation suitably extended in order to consider abducibles
- When an abducible atom *h* is encountered during this derivation, it is assumed, provided this is consistent. The consistency check of a hypothesis, then, starts the second kind of derivation.
- The *consistency derivation* verifies that, when the hypothesis *h* is assumed and added to the current set of hypotheses, any integrity constraint containing h fails. During this latter procedure, when an abducible L is encountered, in order to prove its failure, an *abductive derivation* for its complement is attempted.

Example

Let us consider the following program:

grass is wet :- rained last night
grass is wet :- sprinkler was on
shoes are wet :- grass is wet

with integrity constraint:

:- rained last night, sprinkler was on.

Let predicates *rained last night* and *sprinkler was on* be abducible. The observation *shoes are wet* is explained by two (minimal) sets of sentences, respectively:

Abd1 = *{rained last night, not sprinkler was on} and*
Abd2 = *{sprinkler was on, not rained last night}*

Abduction in multi agent environment: toward ALIAS

- Agents can dynamically join into groups (bunch), with the purpose, for instance, of finding the solution of a goal in a collaborative way
- Although the set of program clauses and integrity constraints might differ from agent to agent, we assume that the set of abducible predicates is the same for all the agents in a bunch.
- When proving a given goal, if an agent A assumes a new hypothesis h, all the agents belonging to the same bunch must check the consistency of h with their own integrity constraints. These checks could possibly raise new hypotheses, whose consistency within the bunch has to be recursively checked. Therefore the abductive explanation of a goal within a bunch of agents is a set of abduced hypotheses, agreed by all agents in the bunch.
- While abductive derivation is limited to the local knowledge base, consistency derivations have to be coordinated within the bunch

Example /1

Let us consider again the situation of the previous example, with one additional agent; in particular, let us consider two agents: A0 and A1. A0 enclose the following abductive logic program:

grass is wet :- sprinkler was on.

shoes are wet :- grass is wet.

with no integrity constraint. Agent A1 (arguing with A0) knows that the sprinkler takes the water from a tank that is emptied after the sprinkler has been on; A1's knowledge base is

tank is full :- not sprinkler was on.

with integrity constraint

:- sprinkler was on; tank is full.

Again, let predicates *rained last night* and *sprinkler was on* be abducible, and $\Delta = \emptyset$ the current set of hypotheses.

Example /2

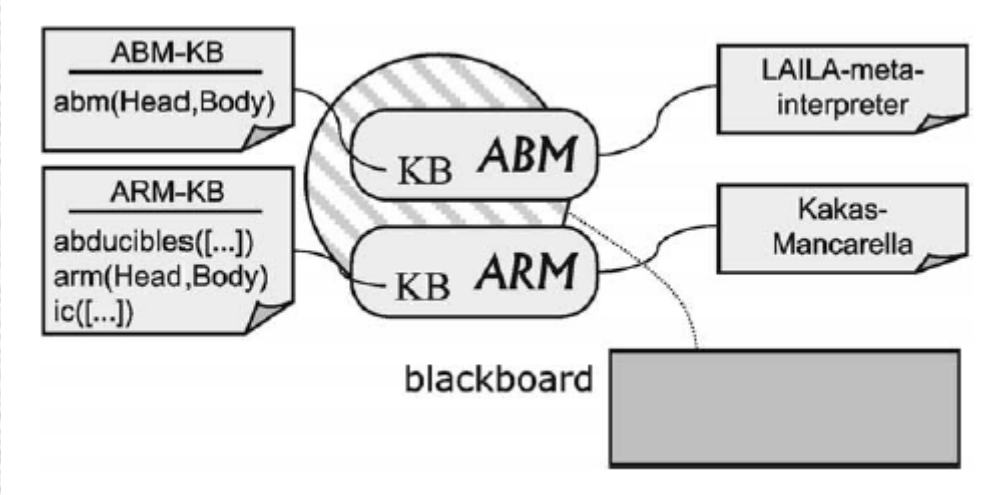
In order to prove the goal (observation) *:-shoes are wet*, agent A0 applies its clause reducing the goal to *:-grass is wet* and therefore to *:-sprinkler was on*. Then, since *sprinkler was on* is abducible, the consistency for sprinkler was on has to be checked within the bunch {A0; A1}. In particular, agent A1 tries to prove the failure of the integrity constraint *:-sprinkler was on, tank is full*. The failure of *tank is full* is proved in one-step derivation by reducing the goal to *:-not sprinkler was on*. Since *not sprinkler was on* is abducible but its complement *sprinkler was on* has already been assumed, this derivation fails, and the consistency check succeeds. Then, A0 agrees with A1 in assuming *sprinkler was on*, that is definitively added to the current set of hypotheses *Δ= {sprinkler was on, not tank is full}.*

The architecture of ALIAS

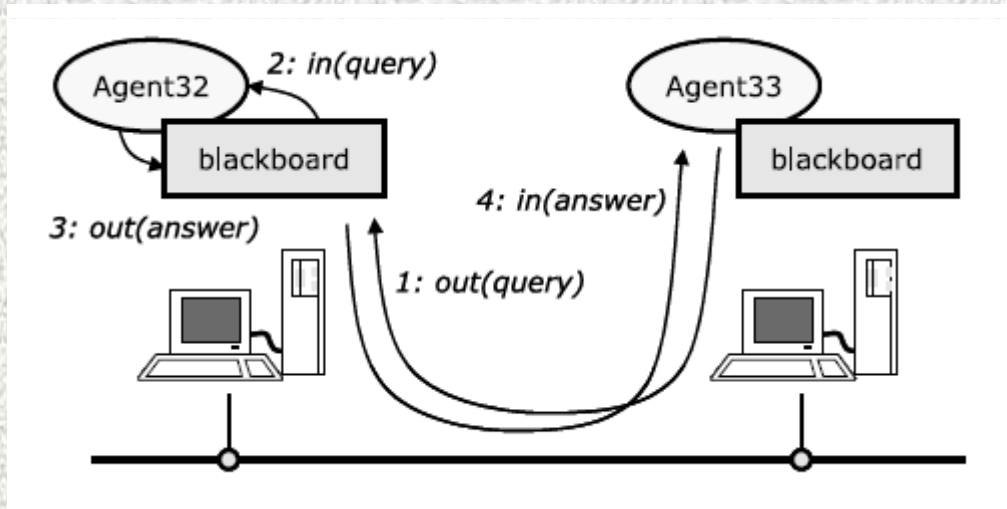
- *Abductive LogIc Agent System* is an agent architecture where several intelligent agents can either autonomously reason on their own local knowledge bases by means of abductive reasoning or exhibit a social behaviour, interacting with other agents following different coordination patterns
- It is a framework that provides mechanism for agent bunches, local abduction and global consistency check
- Inner structure of each agent
  - *Abductive reasoning module* (ARM), enclosing the agent abductive knowledgebase as an abductive logic program used to perform abductive reasoning
  - *Agent behaviour module* (ABM), enclosing the agent behaviour knowledge base, as a set of LAILA clauses that describe action and interaction of the agent with the environment
- Each multi-agent application is represented by a set of abductive agents, each modeled by its knowledge bases

The implementation of ALIAS

- The implementation relies on Jinni
  - High level and declarative features
  - Rapid prototyping
  - Qualitative leap in terms of code readability, homogeneity, maintenance, expressiveness
  - Platform indipendent
  - Less efficiency
- Agents communicate through Linda-like primitives, by posting and consuming tuples in a shared blackboard space: each agent provides a blackboard as a communication space, as a means to update the status of a goal demonstration



Prolog implementation of ALIAS agent's modules



Networked ALIAS agent

The LAILA language

- Language for Abductive Logic Agents
- LAILA models action and interaction among intelligent agent whose main form of reasoning is represented by abduction
- LAILA focuses on agent social behaviour, and especially on how each agent can request the demostration of goals to other agents in the system
- Agent behavior expressed in a logic programming style

The syntax of LAILA

- Let V be the vocabulary of the language:

  V= {←, , , & , ; , > , ↓ , A_0,…..,A_n; a_0,….., a_k , not , true, .}
- A LAILA program is a set of L-clauses, possibly distributed among agents.
- An L-clause is defined as follows:

L – clause	::= Atom ← Body.
Body	::= Formula; Body | Formula
Formula	::= SingleFormula & Formula | SingleFormula
SingleFormula	::= ↓ (Goal)|Agent>(Goal) | Goal
Goal	::= Literal | Literal, Goal | true
Literal	::= Atom | not Atom
Atom	::= a_0 | a_1 | : : : | a_k
Agent	::= A_0 |A_1 | : : : |A_n

Query::= ? Body

Example

- Let us consider, for instance, the following LAILA competitive query, formulated by agent A0:

  ? ↓ *(G1); A1>(G2)*

  It means that A0 must either perform a local abductive derivation for the goal (G1), or ask agent A1 to demonstrate goal G2. Should several answers be available, the system will select only one of them.
- Let us consider, now, the following collaborative query, made by agent A0:

  ? A1>(G3) & A2>(G4)

  It means that agent A0 asks agent A1 to prove G3 and A2 to prove goal G4; the result is obtained by merging the answers produced by A1 and A2 in a unique consistent set of abducibles.

The semantics of LAILA

Successful top–down derivation: Let P be a program, δ a set of abduced hypotheses and F a formula. A successful top–down derivation for F in P can be traced in terms of a (possibly infinite) tree such that:

- The root node is labelled by $A \vdash_{\delta} F$, where A is an agent;
- The internal nodes are derived by using the inference rules defined in the following;
- All the leaves are labelled by the empty formula (true).

Down reflection formula:

$$\frac{A \vdash^{abd}{}_{\delta} G}{A \vdash_{\delta} \downarrow(G)}$$

Communication formula:

$$\frac{A_1 \vdash_{\delta_1} G \wedge \{A_0, A_1\} \vdash^{cons}_{\delta} \delta_1}{A_0 \vdash_{\delta} \quad A_1 > (G)}$$

It requires a consistency check among the two agents, the issuer (A_0) and the solver (A_1)

Collaborative formula:

$$\frac{A_0 \vdash_{\delta'} f \wedge A_0 \vdash_{\delta''} F \wedge A_0 \cup b(f \& F) \vdash^{cons}_{\delta} \delta' \cup \delta''}{A_0 \vdash_{\delta} f \& F}$$

Thus, the collaborative formula results in a branching of the tree from this node to three sub-trees, one for proving *f,* another one for *F* and a last one for the consistency check. If either f proof or F proof fails, the concerning query fails.

Competitive formula:

$$\frac{A_0 \vdash_\delta f}{A_0 \vdash_\delta f;F}$$

$$\frac{A_0 \vdash_\delta F}{A_0 \vdash_\delta f;F}$$

The competitive formula results in a non deterministic choice of one inference rule of the two listed above. If both fail, the concerning competitive query fails.

Consistency check:

$$\frac{\forall A_i \in B \quad A_i \vdash^{abd}_{\delta_i} \delta \wedge B \overset{cons}{\vdash}_{\delta_1} \bigcup_{A_i \in B} \delta_i \wedge \delta \subset \bigcup_{A_j \in B} \delta_j}{B \overset{cons}{\vdash}_{\delta_1} \delta}$$

$$\frac{\forall A_i \in B \quad A_i \vdash^{abd}_{\not\delta} \delta}{B \overset{cons}{\vdash}_{\delta} \delta}$$

Goal formula:

$$\frac{A \vdash_{\delta_1} h \wedge A \vdash_{\delta} \delta_1 \cup G}{A \vdash_{\delta} h, G}$$

$$\frac{\exists h \leftarrow G \in A \wedge A \vdash_{\delta} G}{A \vdash_{\delta} h}$$

$$\frac{}{A \vdash_{\emptyset} \texttt{true}}$$

An example: medical diagnosis

- In medicine, we cannot rely on the assumption about completeness of knowledge
- Let us consider, for instance, the case of a patient exhibiting a lower limbs suffering. In this case, the general medicine doctor could refer to four doctor, each specialized in a particular area: an osteologist, a neurologist, a cardiologist and an angiologist
- We represent each doctor as an ALIAS agent with the aim of showing that coordination connectives such as collaboration and competition could be used to suitably merge incomplete knowledge by joining two or more agents for demonstrating a LAILA query.

An example: medical diagnosis /2

For the sake of synthesis, in agents KBs we denoted symptoms and diseases with the following acronyms:

LLSS = Lower Limbs Suffering under Stress
LLSR = Lower Limbs Suffering in Repose
LLSW = Lower limbs Swelling
SS = Spine Suffering
SD = Skeletal Deformity
HP = Heart Problems
EM = Embolus
THR = Thrombosis
SPO = Spondylitis
MC = Calcium Deficiency
SCIA = Sciatica
DIA = Diabetes

An example: medical diagnosis /3

Osteologist

ABM:

SD(Gender)← ↓(Gender)

ARM :

SD(Gender) :- MC(yes), SPO(yes, Gender)

SD(Gender) :- MC(yes), SPO(no, Gender)

SD(Gender) :- MC(no) , SPO(yes, Gender)

:- MC(yes), MC(no).

:- SPO(yes,M), SPO(no,M).

:- SPO(yes,F), SPO(no,F).

An example: medical diagnosis /4

Neurologist

ABM:

SS(Gender)← ↓(Gender)

ARM :

LLSR :- SCIA(yes)

SS :- SCIA(yes), SPO(yes, Gender)

SS :- SCIA(yes), SPO(no, Gender)

SS :- SCIA(no) , SPO(yes, Gender)

:- SCIA(yes), SCIA(no).

:- SPO(yes,M), SPO(no,M).

:- SPO(yes,F), SPO(no,F).

An example: medical diagnosis /5

Angiologist

ABM:

LLSS ← ↓ *LLLS*

ARM :

LLSS :- EM(yes), THR(yes), DIA(yes)

LLSS :- EM(yes), THR(yes), DIA(no)

LLSS :- EM(yes), THR(no) , DIA(yes)

LLSS :- EM(yes), THR(no) , DIA(no)

LLSS :- EM(no) , THR(yes), DIA(yes)

LLSS :- EM(no) , THR(yes), DIA(no)

LLSS :- EM(no) , THR(no) , DIA(yes)

LLSR :- DIA(yes)

:- EM(yes), EM(no).

:- THR(yes), THR(no).

:- DIA(yes), DIA(no).

An example: medical diagnosis /6

Cardiologist

ABM:

LLSS ← ↓ LLSS

ARM :

LLSS :- HP(yes), EM(yes), THR(yes)

LLSS :- HP(yes), EM(yes), THR(no)

LLSS :- HP(yes), EM(no) , THR(yes)

LLSS :- HP(yes), EM(no) , THR(no)

LLSS :- HP(no) , EM(yes), THR(yes)

LLSS :- HP(no) , EM(yes), THR(no)

LLSS :- HP(no) , EM(no) , THR(yes)

LLSW :- HP(yes)

:- HP(yes), HP(no)

:- EM(yes), EM(no).

:- THR(yes), THR(no)

An example: medical diagnosis /7

Collaborative diagnosis: Osteologist > SD(m) & Neurologist > SS(m).

The Osteologist agent returns the following diagnoses:

δ_{O1} = [MC(yes), SPO(yes,m)]

δ_{O2} = [MC(yes), SPO(no,m)]

δ_{O3} = [MC(no) , SPO(yes,m)]

Similarly, the Neurologist agent returns the following diagnoses:

δ_{N1} = [SCIA(yes), SPO(yes,m)]

δ_{N1} = [SCIA(yes), SPO(no,m)]

δ_{N1} = [SCIA(no), SPO(yes,m)]

The resulting set of consistent diagnoses:

δ_{q1} = [MC(yes), SPO(yes,m), SCIA(yes)]

δ_{q1} = [MC(no) , SPO(yes,m), SCIA(yes)]

δ_{q1} = [MC(yes), SPO(no,m) , SCIA(yes)]

δ_{q1} = [MC(yes), SPO(yes,m), SCIA(no)]

δ_{q1} = [MC(no) , SPO(yes,m), SCIA(no)]

An example: medical diagnosis /8

Competitive diagnosis: Angiologist > LLSS(m) ; Cardiologist > LLSS(m).

The Angiologist agent returns the following diagnoses:

δ_{A1} = [EM(yes), THR(yes), DIA(yes)]

δ_{A2} = [EM(yes), THR(yes), DIA(no)]

δ_{A3} = [EM(yes), THR(no) , DIA(yes)]

δ_{A4} = [EM(yes), THR(no) , DIA(no)]

δ_{A5} = [EM(no) , THR(yes), DIA(yes)]

δ_{A6} = [EM(no) , THR(yes), DIA(no)]

δ_{A7} = [EM(no) , THR(no) , DIA(yes)]

The Cardiologist agent returns the following diagnoses:

δ_{C1} = [HP(yes), EM(yes), THR(yes)]

δ_{C2} = [HP(yes), EM(yes), THR(no)]

δ_{C3} = [HP(yes), EM(no) , THR(yes)]

δ_{C4} = [HP(yes), EM(no) , THR(no)]

δ_{C5} = [HP(no) , EM(yes), THR(yes)]

δ_{C6} = [HP(no) , EM(yes), THR(no)]

δ_{C7} = [HP(no) , EM(no) , THR(yes)]

References

- Ciampolini, Lamma, Mello, Torroni. LAILA: a language for coordinating abductive reasoning among logic agent
- Ciampolini, Torroni. Using Abductive Logic Agents for Legal Justification
- Ciampolini, Lamma, Mello, Toni, Torroni. Cooperation and competition in ALIAS: a logic framework for agent that negotiate
- Expressing Collaborative and Competitive Coordination among Abductive Logic Agents
- Ciampolini, Mello, Storari. Distributed Medical Diagnosis with Abductive Logic Agents