Università degli Studi di Bologna
DEIS

Using Abductive Logic Agents for Legal Justification

A. Ciampolini P. Torroni

August 7, 2002

DEIS Technical Report no. DEIS-LIA-02-002 *LIA Series no. 55*

Using Abductive Logic Agents for Legal Justification

A. Ciampolini P. Torroni

DEIS, Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{aciampolini, ptorroni}@deis.unibo.it

August 7, 2002

Abstract. In this paper we present a novel approach for legal justification based on an abductive multi-agent system. Legal justification has the main objective of finding the plausible explanations for some given pieces of evidence, for instance in a crime trial. In our approach, the process of justification is done through the collaborative abductive reasoning of agents, which operate within a logic-based architecture called *ALIAS*. We apply the proposed approach on an example borrowed from the literature. We show that using ALIAS agents in legal justification allows not only to produce plausible explanations for observed evidences, but also to detect either collusion or inconsistencies among trial characters, and to take into consideration the credibility of the persons (e.g., witnesses) involved in the trial.

Keywords: *Multi-agent systems, logic based reasoning, abduction, legal justification*

1 Introduction

In recent years, the interest for intelligent agents has considerably grown from both theoretical and practical point of view [19]. The agent paradigm, in fact, is well suited to represent applications that merge features inherited both from the distributed systems area (such as locality, distribution, interaction, mobility etc.) and from artificial intelligence (such as reasoning, adaptation, etc.). In particular, intelligent agents require both reasoning capabilities and social abilities, which make interaction and coordination among agents possible.

Abduction has been widely recognized as a powerful mechanism for hypothetical reasoning in presence of incomplete knowledge [6, 9, 15]: it is generally understood as reasoning from effects to causes, and also captures other important issues such as reasoning with defaults and beliefs (see for instance [17, 20]). For these reasons, abduction has been already extensively applied to legal reasoning, especially as an explanatory inference technique in legal trials [27, 10].

In this paper we present a novel approach for legal justification based on an abductive multi-agent system. Legal justification has the main objective of finding the plausible explanations for some given pieces of evidence or observations, for instance in a crime trial. The process of legal justification is supported by abductive reasoning. In fact, abduction could be particularly useful when the knowledge base representing the problem domain is either incomplete or multiple. This is the case, for instance, of legal trials, where some pieces of knowledge about the crime are often missing; moreover, multiple knowledge is available in the case of several witnesses, when it could happen that some of them provide a different version about the crime, and some inconsistencies could arise.

We deal with legal justification by means of ALIAS, an agent architecture where agents are logic-based and capable to reason with abduction. ALIAS agents can be dynamically grouped and their reasoning can be coordinated by means of proper operators (namely, *collaboration* and *competition*).

The multi-agent approach we are proposing relies on several agents each representing a distinct actor in the trial (e.g., the detective, suspects, witnesses, etc.). Once we represent each trial actor by a single abductive agent, we are able to dynamically group (some of) the agents and coordinate them for the explanation of goals. As for traditional abductive reasoning, the system starts from a set S of observed pieces of evidence, to provide one or more sets of hypotheses as possible explanations for S, using the knowledge of a group of agents. In addition, the dynamic composition of agents knowledge bases allows to perform some additional tasks that could help both the detective and judges in solving the case. For instance, it is possible to check the coherence of distinct witnesses, thus possibly to detect either collusion or false testimonies.

It is worthwhile noticing that, in realistic cases, finding explanations for a given evidence could be unsuccessful due to various reasons: for instance, not all the available pieces of evidence could be reliable, depending, for instance, on the source. We faced this problem by representing the reliability either of each *dramatis persona* or even of single pieces of knowledge (e.g. a passage in a witness interrogation) by means of proper hypotheses. In this way we can obtain justifications labelled with reliability hypotheses, that could further help detective and juries in understanding behaviours and roles of actors involved in the trial. We tested the system on the Peyer case, a well known trial that has been adopted as a test bed by previous abduction-based approaches [27, 10, 22].

The paper is organized ad follows. Section 2 gives some preliminary notions on abduction and presents *ALIAS*. In section 3 we show how to apply abduction and agents to the Peyer case, and give some general guidelines to be followed when using abductive agents in legal trials. A discussion on related literature is presented in section 4. Conclusions follow.

2 Logic-based abductive reasoning and Abductive Logic Agents

In this section we give some preliminary notions on abductive logic programming and multi-agent systems. In particular, we firstly introduce logic-based abduction, as a tool for hypothetical reasoning. We then present *ALIAS*, the multi-agent architecture based on abductive logic agents that we use in the following to model the process of legal justification.

2.1 Abductive logic programming

An assumption that is often made in multi-agent systems is that individuals only have a partial knowledge of the environment they are situated in. This is particularly true in the legal application domains, if we want to use agents to model physical persons or different jurisdictional institutions, as we will see in the sequel. Therefore, in most knowledge-intensive applications of agents, it is often the case that an intelligent agent requires some sort of *guess* about a computation which cannot be performed locally, since the local knowledge is incomplete. In the legal setting, the incomplete knowledge could be, for instance, the information that each person involved in a legal trial (either the detective, a witness or a judge) has about the crime. In a logic programming perspective, we could rephrase it by saying that a guess is needed because the agent has a *goal* which cannot be solved locally. In this situation, the Closed World Assumption [4] usually adopted in logic programming can be no longer assumed, and some form of *open* or *abductive* reasoning has to be considered. Different forms of abduction, but not in logic programming, have already been used for legal justification by different authors [27, 10].

Abduction is generally understood as reasoning from effects to causes, and captures other important issues, such as reasoning with defaults and beliefs (see for instance [17, 20]). Incomplete knowledge is handled by labeling some pieces of information as abducibles, i.e., possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base. In the context of intelligent agents, abduction can be regarded as a way of *dynamically* enlarging the agent's knowledge with abducible atoms. In the case of a single agent that uses a proof to reason abductively, such proof will guarantee that the abduced hypotheses that enlarge the agent's knowledge base are consistent with its knowledge. To this purpose, an agent's knowledge base is equipped with some integrity constraints, that must never be violated.

Before giving a formal definition of abductive logic programs, we will briefly recall some basic concepts about logic programs.

A logic program is a set of clauses [1]

$$A \leftarrow L_1, \ldots, L_n$$

where A is an atom, and each L_i is a literal, i.e., an atom B or its negation *not* B. An *atom* is an n-ary predicate symbol followed by a list of n terms $p(t_1, \ldots, t_n)$, or `true`, or `false`. A *literal* is an atom or the negation of an atom. A *term* is a constant, a variable or a compound term.

Note that `not` represents default negation or negation as failure. All variables in $A, L_1, \ldots, L_n$ are assumed to be universally quantified from the outside. A is referred to as the *head* of the clause and $L_1, \ldots, L_n$ as its *body*.

Definition 1 (*abductive logic program*) An *abductive logic program* (ALP) is a triple $\langle P, \mathcal{A}, IC \rangle$ where P is a logic program, possibly with abducible atoms in clause bodies; $\mathcal{A}$ is a set of *abducible predicates*, i.e. *open* predicates which can be used to form explaining sentences; IC is a set of integrity constraints.

Given an abductive program $\langle P, \mathcal{A}, IC \rangle$ and a sentence G, the purpose of abduction is to find a (possibly minimal) set of abducible atoms δ which together with P "entails" G, and such that $P \cup \delta$ "satisfies" IC. Depending on the chosen semantics for the given logic program [14], various notions of "entailment" are possible, such that, for every ground literal in the language, either the literal or its complement is entailed, and such that no literal and its complement are entailed at the same time. In the following, we will use the following notation for the concept of "abductive entailment":

$$\langle P, \mathcal{A}, IC \rangle \overset{abd}{\vdash}_{\delta} \quad G$$

and it is possible to formalize its declarative semantics in the following way:

Definition 2 (*declarative semantics of an ALP*) Let $\langle P, \mathcal{A}, IC \rangle$ be an abductive logic program. Then, given a goal G, the *declarative semantics* of $\langle P, \mathcal{A}, IC \rangle \overset{abd}{\vdash}_{\delta} \quad G$ is defined as follows:

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

where $\models$ is the entailment symbol.

[1] From now on, we will use logic programming notation as it is defined in [18]. Moreover, in this work we will adopt the Edinburgh notation for standard Prolog, where variable names start with upper case or '_', and constants start with lower case.

There are also different notions of "integrity constraint satisfaction" in literature. Different procedures / semantics will adopt different such notions. Moreover, there is not a unique *syntax* to write integrity constraints: depending on the adopted proof-procedure, they could be written for example as denials, as in the case of the Kakas-Mancarella proof-procedure, or as forward rules (implications), as in the IFF proof-procedure. All proofs require that each integrity constraint contains at least one abducible, and that abducible predicates are not defined, i.e., there is no clause in an ALP that has an abducible in the head. In fact, this would mean that the abducible could be derived by deduction, while we consider abducibles as open predicates that can be assumed true or false, depending on the goal to solve. It is worthwhile noticing that, given a program P, possibly containing definitions of abducible predicates, it is possible to syntactically transform P into another semantically equivalent program, by means of auxiliary predicates, where no abducible is defined. Such transformation is shown in [14]. The following example shows a simple abductive logic program, written according to the syntax of integrity constraints required by the Kakas-Mancarella proof-procedure.

Example 1 Let us consider the following program (inspired by the Peyer case, described in section 3), expressing two possible explanations for the retrieval of the body of Cara Knott, in the Peyer trial. [2] In this example the atoms *peyer_killed_knott* and *someone_else_killed_knott* are abducible predicates:

$found\_knott's\_body \leftarrow peyer\_killed\_knott.$
$found\_knott's\_body \leftarrow someone\_else\_killed\_knott.$

with integrity constraint:

$\leftarrow someone\_else\_killed\_knott, peyer\_killed\_knott.$

In order to satisfy such constraint, the predicates *someone_else_killed_knott* and *peyer_killed_knott* cannot be assumed both true at the same time. The observation $found\_knott's\_body$ is therefore explained by two (minimal) sets of sentences, δ_1 and δ_2:

$\delta_1 = \{someone\_else\_killed\_knott, not\ peyer\_killed\_knott\}$ and
$\delta_2 = \{peyer\_killed\_knott, not\ someone\_else\_killed\_knott\}$

According to [9], negation as default, possibly occurring in clause bodies, can be recovered into abduction by replacing negated literals of the form *not a* with a new

[2] "Peyer" is the name of the main defendant in the trial.

positive, abducible atom not_a and by adding the integrity constraint $\leftarrow a, not\_a$ to IC. The natural syntactic correspondence between a standard atom and its negation by default is given by the following notion of complement:

$$\bar{l} = \begin{cases} \alpha & if \quad l = not\_\alpha \\ not\_\alpha & otherwise \end{cases}$$

where α is an atom. All negated literals in an ALP are considered abducibles.

The integrity constraints used for handling negation as default, like the constraint $\leftarrow p, not\ p$, are considered trivial, and we will leave them implicit and suppose that they are replicated in each agent's knowledge base.

We will briefly introduce now the Kakas-Mancarella proof-procedure for abductive derivation, that implements the declarative semantics defined above. This is not the only abductive proof-procedures defined in the logic programming literature [11, 7, 5], but it is the one adopted by ALIAS agents.

2.2 The Kakas-Mancarella proof-procedure

The Kakas-Mancarella (KM, for short) proof-procedure [16] is an extension of the proof-procedure for ordinary logic programming proposed by [9]. Both procedures extend the basic resolution mechanism adopted in SLD and SLDNF in ordinary logic programming by introducing the notion of *abductive* and *consistency* derivations.

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 (i.e., the bodies of all the integrity constraints containing h are false). During this latter procedure, when an abducible L is encountered, in order to prove its failure, an abductive derivation for its complement, $\overline{L}$, is attempted. In other words, as we said before, an abductive derivation computes a set of hypotheses "entailing" (with respect to the given logic program) a given goal/query/observation, starting from a given set of initial hypotheses. In order to compute such set and guarantee that the integrity constraints are "satisfied", an abductive derivation might require subsidiary consistency derivations, which in turn might require additional abductive derivations to ensure integrity constraint "satisfaction".

The only detail that we would like to give about the KM is about the syntax of abductive logic programs. In its original formulation, the KM requires that P is a propositional logic program, expressed in form of ground clauses. The integrity constraints are expressed in the form of denials of atoms, as in Example 1; at least one of such atoms must be abducible. Integrity constraint satisfaction in KM requires that at least one atom in the body of an integrity constraint be false. For instance, given the constraint

$\leftarrow a, b, c.$

and the set $\{a, b\}$ of abducibles, if we want to assume a to be true, we must either assume b false (by abducing *not* b) or prove that c is false. Both ways may require a deductive derivation in P, and possibly some more consistency steps. Such process is iterated until we find a consistent and "stable" set of abducibles δ, in the sense that further consistency steps do not result in an enlargement of such δ. The KM proof-procedure can be adopted also with first order logic programs (not only propositional): in that case, all variables in the body of an integrity constraint are assumed to be universally quantified from the outside. The KM proof-procedure has been proven sound [1], for some specific notions of "entailment" and "satisfaction", with respect to the semantics defined in [8] and weakly complete for the 3-valued stable model semantics of [1], as well as the argumentation-semantics defined in [28].

2.3 Abductive Logic-based multi-agent system: the ALIAS architecture

The agent metaphor is widely used in distributed artificial intelligence, although no real agreement has ever been reached on the definition of agent. In fact, the term 'agent' has been used to describe a broad range of computational entities, ranging from simple systems to very large ones [19]. In our work, we use the agent metaphor as a model of a real world legal case, and we specify the agent knowledge by using the logic programming paradigm [29], and particularly, a logic language. We will define here the concept of agent system, from a logic-programming perspective.

Definition 3 (*agent system*) An *agent system* is a finite set A, where each $x \in A$ is a ground term, representing the name of an agent, equipped with a *knowledge base* $\mathcal{K}(x)$. Each $\mathcal{K}(x)$ includes an abductive logic program.

We assume that, in an agent system, the agents share a common language. In particular, in *ALIAS*, the individuals will have separate knowledge bases but will refer to a common set of abducibles $\mathcal{A}$.

In ALIAS, each agent may thus autonomously reason on its own knowledge base and find abductive explanations to submitted goals.

Furthermore, as far as conflicting agents beliefs and goals are concerned, we propose a technique aimed to maintaining consistency in an environment made of multiple individuals, and to aiding the agent reasoning through possible conflicts of beliefs. To this purpose, Abductive Logic Agents can dynamically join into groups, or *bunches*, with the purpose, for instance, of finding the solution of a given goal in a *collaborative* way. In this perspective, although the set of program clauses and integrity constraints might differ from agent to agent, the set of abducible predicates (default predicates included) is the same for all the agents in a bunch. This implies that when proving a given goal, if an agent A assumes a new hypothesis h, all the arguing agents (i.e., the agents belonging to the same bunch) must check the consistency of h with their own integrity constraints. These checks could raise new hypotheses, whose consistency within the bunch has to be recursively checked. Therefore, in ALIAS, the abductive explanation of a goal within a bunch of agents is a set of abduced hypotheses, agreed by all agents in the bunch.

To this purpose, the current *ALIAS* implementation provides mechanisms to support agent bunches, local abduction and global consistency checks. As far as local abduction is concerned, the current implementation adopts the KM proof-procedure.[3]

In order to guarantee consistency of abduced hypotheses within a bunch, we adopt a two-phase algorithm, which first collects the results of local[4] abduction processes by running the KM proof-procedure, and then, depending on the chosen coordination policy, checks their consistency, and, in case of success obtains a consistent set of hypotheses Δ. The following example will show how abduction is performed within a bunch of two agents.

Example 2 Let us consider again Example 1, where all the knowledge needed to find an explanation to the observation *found_knott's_body* is enclosed in a single agent, that we will call A_0. Now, let us suppose that another agent, say A_1, has a additional knowledge about the same problem. In particular, let A_1's knowledge base be:

peyer_has_alibi.

and the following integrity constraint:

$\leftarrow$ *peyer_killed_knott, peyer_has_alibi.*

In order for them to communicate the result of their reasoning, expressed in form of conjuncts of abducibles, we assume that they share the same set of abducibles. Let such set be

$$\mathcal{A} = \{someone\_else\_killed\_knott, peyer\_killed\_knott\}$$

[3]It is worth noticing, however, that despite this implementation choice, the ALIAS architecture is not strictly bound to the KM proof-procedure: the same high-level features of the system could exploit different abduction algorithms ([7, 13].

[4]In this context, *local* means *performed within the agent that demonstrates the query.*

plus all negated literals. The two agents therefore form a bunch together, $B = \{A_0, A_1\}$.

In order to demonstrate the goal (observation) *:-found_knott's_body*, agent A_0 applies its first clause reducing the goal to *:- peyer_killed_knott.* Then, since *peyer_killed_knott* is abducible the consistency for *peyer_killed_knott* has to be checked within the bunch $\{A_0, A_1\}$. In particular, agent A_1 tries to prove the failure of the integrity constraint $\leftarrow$ *peyer_killed_knott, peyer_has_alibi.* To do this, the failure of *peyer_has_alibi* must be proved: this derivation fails, since A_1's ALP contains the fact *peyer_has_alibi.*

Then, A_0 applies its second clause reducing the goal to *:- someone_else_killed_knott*; again, the consistency check would require to prove the failure of integrity constraints: in this case, only the constraint in A_0 contains the predicate *someone_else_killed_knott* representing an hypothesis against Peyer's guilt. In order to prove its failure, we abduce also the hypothesis *not peyer_killed_knott* so the derivation terminates with success, producing the set of hypothesis

$$\delta = \{someone\_else\_killed\_knott,\ not\ peyer\_killed\_knott\}$$

In the previous example we have considered a pre-defined bunch of two agents for explaining goals with a set of abducible hypotheses which is globally consistent with the knowledge bases of all agents in the bunch. However, in ALIAS more flexible forms for agents composition and coordination are possible, thanks to a *Language for AbductIve Logic Agents* (*LAILA*). In the following we describe it briefly, especially focusing on the LAILA operators used for agent grouping, interaction and coordination.

2.4 The LAILA Language

The *ALIAS* agent behaviour can be expressed by means of the *Language for AbductIve Logic Agents*, LAILA. This language allows to model agent actions and interactions in a logic programming style.

In particular, using the LAILA language, the programmer can express the features of an abductive multi-agent application in a declarative way. In this paper we will focus on agent social behaviour, and on how each agent can request demonstration of goals to other agents in the system. A detailed description of the language can be found in [3].

With regard to agent coordination, we introduce two composition operators: the collaborative AND operator (`&`) and the competitive operator (`;`) which can be used by each agent to express and coordinate abductive queries to other (set of) agents. The language provides also a communication operator (`>`) which is used to submit queries to other agents.

Competitive queries. The competitive operator (;) could be exploited to formulate a *disjunction* of alternative queries involving two (or more) abductive agents.

Let us consider the following LAILA competitive query q, formulated by agent A_0:

$$? \texttt{A1} > \texttt{G}_1 \; ; \; \texttt{A2} > \texttt{G}_2$$

It means that A_0 must either perform a local abductive derivation for the goal (G_1), *or* ask agent A_1 to demonstrate goal G_2. If several answers are available, the system will select only one of them.

In more detail, the competitive query q causes the following effects:

- A_0 asks A_1 to solve G_1; if G_1 succeeds in A_1, N ($N > 0$) abductive explanations δ_{1i} ($i \in [1, \ldots N]$), consistent in the bunch $\{A_0, A_1\}$, could be obtained for G_1.

- A_0 asks A_2 to solve G_2; if G_2 succeeds in A_2, M ($M > 0$) abductive explanations δ_{2j} ($j \in [1, \ldots M]$) consistent in the bunch $\{A_0, A_2\}$, could be obtained for G_2.

- The resulting abductive explanation is either δ_{1i} (for some $i \in [1, \ldots, N]$) or δ_{2j} (for some $j \in [1, \ldots, M]$).

If both A_1 and A_2 fail, the competitive query fails.

Collaborative queries. The collaborative operator (&) is used to ask queries to different abductive agents with the aim of merging their answers into a unique consistent set of hypotheses. For instance, each query could represent a sub-problem of the problem to be solved.

Let us give an example of collaborative query, issued by agent A_0:

$$? \texttt{A1} > \texttt{G1} \; \& \; \texttt{A2} > \texttt{G2}$$

It expresses that agent A_0 asks agent A_1 to prove G_1 *and* at the same time A_2 to prove goal G_2; the query will succeed *if and only if* both agents A_1 and A_2 succeed; in particular, it will cause the following effects:

- A_0 asks A_1 to solve $G1$; if $G1$ succeeds in A_1, N ($N > 0$) abductive explanations δ_{1i} ($i \in [1, \ldots N]$), consistent in the bunch $\{A_0, A_1\}$, could be obtained for $G1$.

- A_0 asks A_2 to solve $G2$; if $G2$ succeeds in A_2, M ($M > 0$) abductive explanations δ_{2j} ($j \in [1, \ldots M]$) consistent in the bunch $\{A_0, A_2\}$, could be obtained for $G2$.

- The abductive explanation for the collaborative query q is therefore a set of hypotheses:

$$\Delta_q = \{\delta_k \mid \delta_k \ = \ \delta_{1i} \cup \delta_{2j} \ \wedge \ \delta_k \text{ is consistent in } \{A_0, A_1, A_2\}\}$$

If either $A1 > G1$ or $A2 > G2$ fails, the query q fails.

In [3] the interested reader can find in detail the operational semantics of the operators cited above, along with the definition of our notion of *consistency*. The declarative semantics of the notion of consistency of a set of hypotheses δ in a bunch $B = \{A_i\}$ of agents is defined as follows:

$$\forall A_i \in B \ \ P_i \cup \delta \not\models \bigcup_{j \in S} IC_j$$

In this notation, S is the set of indexes of agents A_i in B, and each agent A_i contains a logic program P_i and a set of integrity constraints IC_i. This definition ensures the absence of conflicts between δ and any integrity constraints in any of B's agents. LAILA allows to implement such semantics with no need to put the agents' knowledge bases together (all the computation is done locally).

2.5 ALIAS implementation

The current implementation of *ALIAS* relies on Jinni [26], a lightweight, multi-threaded, logic programming language extended with Linda-like primitives, used as a development platform for building multi-agent applications based on logic agents. Although we are aware that a more direct and customized implementation should be more efficient than the current one, we choose Jinni mainly for its high-level and declarative features that allow us to achieve a rapid prototyping of the *ALIAS* system. The adoption of Jinni allows an effective qualitative leap in terms of code readability, homogeneity, easy maintenance, and expressiveness. In fact, both the reasoning and the coordination issues introduced by *ALIAS* are solved by means of logic clauses, leading to the development of a fully declarative design. Moreover, *ALIAS* is platform-independent, since Jinni is written in Java.

3 Multi-agents and logic-based abduction for legal justification: a case study

In this section, we show a possible application of ALIAS in the legal domain. To this purpose, we borrow an example from the literature: the Peyer case. This case has been first cited in [27] by Thagard, as a case study for to show hypotheses enforcement via his connectionist program, ECHO, and further reconsidered by other authors doing related research in abduction and its application. In Section 4 we will give a brief review of some related work. Now we would like to introduce the Peyer case and show how it can be modelled in ALP.

3.1 The Peyer Case

The Peyer case is about a twenty-two-year-old San Diego State University honor student, Cara Knott, found strangled and thrown from a 75-foot bridge beside Interstate 15, near San Diego, California, on December 27, 1986. The trial ended in June 1988 in San Diego Superior Court, with Craig Peyer of Poway found guilty of first-degree murder. Thagard's formalization (appeared in 1989) of the problem, further used by Fox and Josephson, is based on evidence and hypotheses as appeared, the day following the murder, in the *San Diego Union* and the *San Diego Tribune*. Based on Thagard's report, we will show here how the case can be put in the form of abductive logic programs, representing the knowledge of a multitude of actors (the agents) that actually plaid a role in the Peyer trail, as witnesses, suspect (Peyer himself) that really existed and of an imaginary detective.

Following on Thagard's formalization, we introduce a number of predicates divided into *evidence predicates*, including testimonies, evidence, findings, matters of fact, etc., *hypotheses*, both for Peyer's guilt and innocence, and *rules* that explain evidence predicates in terms of hypotheses. Being a case study, the formalization of the problem is indeed simplified. We could easily imagine real application scenarios, where agents representing juries could for instance enclose the whole corpus of laws of a country, and abductively reason about the applicability of certain laws in certain cases, and so on.

Let us start by giving the formalization of the Peyer's case as in [27]. In Figure 1 we list a collection of evidence, in Figure 2 the hypotheses and in Figure 3 the case for Peyer's innocence and guilt, expressed in terms of rules that bind evidence with possible explanations. In these figures we adopt the same notation as Thagard [27] for evidence and hypotheses. In Figure 4 we show the "contradictions", that we express in our framework in terms of integrity constraints.

Evidence

[**E1**]**:** *Knott's body and car were found on a frontage road near I-15.*
[**E2**]**:** *22 young women reported being talked to at length by Peyer after being stopped near where Knott's body was found.*
[**E3**]**:** *Calderwood said that he saw a patrol car pull over a Volkswagon like Knott's near I-15.*
[**E7**]**:** *Ogilvie said Peyer quizzed her about the case and acted strangely.*
[**E8**]**:** *Dotson said Olgivie is a liar.*
[**E12**]**:** *Anderson says she saw Peyer wipe off his nightstick in his trunk.*
[**E14**]**:** *Bloodstains found on Knott's clothes matched Peyer's blood.*

Figure 1. *Excerpt A of Thagard's modeling of the Peyer's case: evidence of Peyer's case.*

Hypotheses that Peyer is guilty

[**G1**]**:** *Peyers killed Knott.*
[**G4**]**:** *Peyer pulled Knott over.*
[**G6**]**:** *Peyer like to pull over young women.*
[**G7**]**:** *Peyer had a bloody nightstick.*

Hypotheses that Peyer is innocent

[**I1**]**:** *Someone other than Peyer killed Knott.*
[**I4**]**:** *Ogilvie lied.*
[**I4A**]**:** *Ogilvie is a liar.*
[**I7**]**:** *Anderson was mistaken about the nightstick.*

Figure 2. *Excerpt B of Thagard's modeling of the Peyer's case: list of possible explanations of evidence.*

The case for Peyer's guilt

G7 ← **G1**
E1 ← **G1**
E2 ← **G6**
E3 ← **G4**
E7 ← **G1**
E12 ← **G7**
E14 ← **G1**

The case for Peyer's innocence

E1 ← **I1**
E7 ← **I4**
E8 ← **I4A**
I4 ← **I4A**
E12 ← **I7**

Figure 3. *Excerpt C of Thagard's modeling of the Peyer's case: relationships between evidence and possible explanations.*

Contradictions

← **G1, I1**
← **G7, I7**

Figure 4. *Excerpt D of Thagard's modeling of the Peyer's case: contradictions between hypotheses.*

3.2 Modeling the Peyer case in ALIAS

In this section we model the Peyer case using abductive logic-based agents and the ALIAS multi-agent architecture.

As already anticipated, the *dramatis personæ*, i.e., the characters that are relevant to solve the case, are modelled by means of agents. We refer to some standard roles that an agent can play, for instance, a detective, a witness, a suspect, a member of a jury, etc. In the examples that we show in the sequel, we will call d the detective, w_X the witnesses, and s_X a suspect, where X is the name of the particular witness or suspect.

In our framework, we also need to model the *data* that are relevant to make any kind of inference. We distinguish between a *piece of evidence* and a *testimony*. A piece of evidence is, for instance, the one identified with **E14** in [27] (v. Figure 1): "*Bloodstains found on Knott's clothes matched Peyer's blood*". Pieces of evidence are mapped as non-abducible predicates, being incontrovertible facts. They can be modeled either as facts, that are assumed to be true, or as observations, that must be explained y means of suitable hypotheses.

For instance, "*22 young women reported being talked to at length by Peyer*" (**E2**) is a fact, that can be used to explain more facts. "*Knott's body and car were found on a frontage road near I-15*" (**E1**), instead, is an observation that can be explained by means of hypotheses, like: "*Peyers killed Knott*" (**G1**), or "Someone else killed Knott" (**I1**).

Testimonies do not represent facts. Witnesses can be more or less reliable. What is true in a testimony is indeed the act of uttering it, but not necessarily the content of the testimony. If Calderwood declares "I saw a patrol car pull over a Volkswagen like Knott's near I-15", this does not necessarily imply that a patrol car pulled over any Volkswagen ever, while it does indeed imply that Calderwood ($\mathtt{w_{calderwood}}$) released such a testimony. In our representation of a case, we will therefore distinguish between a testimony and content of a testimony, and consider the content of a testimony as a defeasible hypothesis that can be assumed true, or false. We will map testimonies into atoms starting by "`t`", and hypotheses as atoms starting by "`h`": for instance, "Calderwood said that a patrol car pulled over a Volkswagen like

Knott's near I-15" will be mapped into the atom $\mathtt{t}_{\mathtt{patrol\_car\_pulled\_over\_vw}}$, while the hypothesis "a patrol car pulled over a Volkswagen like Knott's near I-15" will be mapped into $\mathtt{h}_{\mathtt{patrol\_car\_pulled\_over\_vw}}$.

Operationally, as we will see in the sequel, this kind of hypotheses which can emerge from direct testimonies will be generated during the multi-agent abductive inference process by the witness agents themselves. For instance, a witness will reply to a question about some aspects of the case, by means of rules of the kind:

$$\mathtt{ask(Aspect)} \leftarrow \mathtt{h\_X}$$

For the example above, the agent acting on the behalf of Calderwood will have the following rule, which expresses its ability to answer to a question about the car:

$$\mathtt{ask(car)} \leftarrow \mathtt{h}_{\mathtt{patrol\_car\_pulled\_over\_vw}}$$

In [27], a rule like this is represented by means of a proposition (see Figure 2), and there is no distinction between the content of a proposition and the act of uttering the proposition itself.

The objective of a detective d in this setting is that of providing objective explanations of observations, thus finding for instance the responsible persons of a crime. This is done in our multi-agent framework in the following way: The detective triggers the explanation of testimonies by means of constraints of the kind:

$$\leftarrow \mathtt{h}_{\mathtt{t\_X}}, \mathtt{not\ expl}(\mathtt{h}_{\mathtt{t\_X}})$$

where `expl` is a meta-predicate on abducibles, defined by rules that represent the detective's reasoning activity.

For instance:

$$\leftarrow \mathtt{h}_{\mathtt{patrol\_car\_pulled\_over\_vw}}, \mathtt{not\ expl}(\mathtt{h}_{\mathtt{patrol\_car\_pulled\_over\_vw}}).$$
$$\mathtt{expl}(\mathtt{h}_{\mathtt{patrol\_car\_pulled\_over\_vw}}) \leftarrow \mathtt{h}_{\mathtt{peyer\_pulled\_over\_knott}}.$$
$$\mathtt{expl}(\mathtt{h}_{\mathtt{patrol\_car\_pulled\_over\_vw}}) \leftarrow \mathtt{h}_{\mathtt{peyer\_pulled\_over\_another\_vw}}.$$
$$\mathtt{expl}(\mathtt{h}_{\mathtt{patrol\_car\_pulled\_over\_vw}}) \leftarrow \mathtt{h}_{\mathtt{another\_patrol\_car\_pulled\_over\_knott}}.$$

et cetera.

Modelling reliability

This schema can be also adopted in the case of witnesses that predicate about other witnesses' reliability. The Peyer case provides us once more with an example of this: a witness, Dotson, declares that another witness, Ogilvie, is a liar. Inside Dotson's knowledge base we find the rule:

$\mathtt{ask(ogilvie)} \leftarrow \mathtt{h}_{\mathtt{ogilvie\_liar}}$.

and inside the detective, the constraint that puts such hypothesis into relationship with its explanation.

$\leftarrow \mathtt{h}_{\mathtt{ogilvie\_liar}}$, $\mathtt{not\ expl}(\mathtt{h}_{\mathtt{ogilvie\_liar}})$.
$\mathtt{expl}(\mathtt{h}_{\mathtt{ogilvie\_liar}}) \leftarrow \mathtt{not\ h}_{\mathtt{rel\_ogilvie}}$.

We imagine the solution of a case such as Peyer's as a dynamic process, in which we can, for instance, add more questions to witnesses, relevant facts, or evidence depending on the current state of the trial and on our knowledge.

Once we obtain a certain conjunct of hypotheses Δ, with a particular group of agents, we can check it against an enlarged group of agents including more witnesses, for example in order to verify their reliability. Here comes in hand the disjunctive LAILA operator (competitive operator):

$\Delta\ \ \&\ \ (\mathtt{w\_X} > \mathtt{ask(Y)}\ ;\ \mathtt{not\ h}_{\mathtt{rel\_Y}})$.

In the case of Ogilvie/Dotson, if I am "certain" that Ogilvie is a reliable witness, I can query Dotson about Ogilvie herself in the following way:

$\{\mathtt{h}_{\mathtt{rel\_ogilvie}}\}\ \ \&\ \ (\mathtt{w\_dotson} > \mathtt{ask(ogilvie)}\ ;\ \mathtt{not\ h}_{\mathtt{rel\_dotson}})$.

This LAILA query would produce the following effects: if we follow the first branch of the competitive query we would fail, since from $\mathtt{w\_dotson} > \mathtt{ask(ogilvie)}$ we derive the hypothesis $\mathtt{h}_{\mathtt{ogilvie\_liar}}$ from Dotson's query, that would activate the detective's integrity constraint which forces him to explain $\mathtt{h}_{\mathtt{ogilvie\_liar}}$: $\mathtt{expl}(\mathtt{h}_{\mathtt{ogilvie\_liar}})$. This together with the other integrity constraint, would generate the hypothesis $\mathtt{h}_{\mathtt{rel\_ogilvie}}$ which clashes with Δ.

The second branch, instead, succeeds, leading to a specialization of the initial hypothesis: $\Delta' = \{\mathtt{h}_{\mathtt{rel\_ogilvie}}, \mathtt{not\ h}_{\mathtt{rel\_dotson}}\}$. I can therefore conclude that Dotson is not a reliable witness.

In this way, the detective is able to collect all the hypotheses that lead to a possible reconstruction of the events, and he will try to prove it in collaboration with the group of witnesses that proved so far reliable.

For instance, a possible logical justification of the case for Peyer's guilt is:

$\mathtt{expl}(\mathtt{h}_{\mathtt{peyer\_killed\_knott}}) \leftarrow$
$\quad \mathtt{h}_{\mathtt{peyer\_pulled\_over\_knott}}$,

$\texttt{h}_{\texttt{peyer\_hit\_knott\_with\_nightstick}}$.

This justification is supported by specific testimonies (Calderwood's about the car, Anderson's about the nightstick, etc.). Therefore we can refine the detective's knowledge base by adding the following query:

$\texttt{d} > \texttt{expl}(\texttt{h}_{\texttt{peyer\_killed\_knott}})$ &
$\texttt{d} > \texttt{reliable\_witnesses}$ &
$\texttt{w}_{\texttt{calderwood}} > \texttt{ask(car)}$ &
$\texttt{w}_{\texttt{anderson}} > \texttt{ask(nightstick)}$.

where the definition of `reliable_witnesses` is updated depending on the current beliefs of the detective about the witness reliability. For instance:

$\texttt{reliable\_witnesses} \leftarrow$
  $\texttt{h}_{\texttt{rel\_calderwood}}$,
  $\texttt{h}_{\texttt{rel\_anderson}}$,
  $\texttt{not } \texttt{h}_{\texttt{rel\_dotson}}$.

Inconsistencies and collusion

It is possible in this way also to prove the inconsistencies of various testimonies: for instance, Peyer says that he did not have a bloody nightstick, while Anderson says that she saw Peyer wiping off the nightstick on his trunk:

% Peyer's knowledge base
$\texttt{ask(nightstick)} \leftarrow$
  $\texttt{h}_{\texttt{not\_bloody\_nightstick}}$.

% Anderson's knowledge base
$\texttt{ask(nightstick)} \leftarrow$
  $\texttt{h}_{\texttt{wipe\_nightstick}}$.

% detective's knowledge base
$\leftarrow \texttt{h}_{\texttt{wipe\_nightstick}}, \texttt{not expl}(\texttt{h}_{\texttt{wipe\_nightstick}})$
$\leftarrow \texttt{h}_{\texttt{not\_bloody\_nightstick}}, \texttt{h}_{\texttt{bloody\_nightstick}}$
$\texttt{expl}(\texttt{h}_{\texttt{wipe\_nightstick}}) \leftarrow \texttt{h}_{\texttt{bloody\_nightstick}}$

If the detective asks:

$\mathtt{s_{peyer}} > \mathtt{ask(nightstick)}$ & $\mathtt{w_{anderson}} > \mathtt{ask(nightstick)}$

it obtains the following: $\mathtt{s_{peyer}} > \mathtt{ask(nightstick)}$ produces $\mathtt{h_{not\_bloody\_nightstick}}$, which by the detective's integrity constraints generates not $\mathtt{h_{bloody\_nightstick}}$. On the right hand side, $\mathtt{s_{anderson}} > \mathtt{ask(nightstick)}$ produces $\mathtt{h_{wipe\_nightstick}}$, that by the detective's integrity constraints generates $\mathtt{expl(h_{wipe\_nightstick})}$ and therefore $\mathtt{h_{bloody\_nightstick}}$. This hypothesis is in contrast with the previous one obtained from Peyer. If both are the only open derivation branches in this query, it indeed fails, which can be interpreted as an inconsistency in the two persons' testimonies, provided that they are properly modelled in the system.

We see how the inquiry, as it progresses, can highlight inconsistencies among the testimonies. If $\mathtt{h_{rel\_anderson}}$ is in the set of witnesses that I currently consider reliable (defined by `reliable_witnesses`), I can obtain from a query such as that above specified a new Δ that gives reason to Anderson, and I can use the new hypothesis not $\mathtt{h_{rel\_peyer}}$ to update the definition of `reliable_witnesses`. Different definitions of `reliable_witnesses` can lead to different conclusions.

By proceeding in a similar way to that used to find out inconsistencies, we can use LAILA to determine groups of agents that agree between themselves, but that are not consistent with some evidence that the detective knows. This can be used, case by case, to raise some hypotheses that assume what the agreements between testimonies or suspects are.

3.3 Some guidelines about modeling case using ALIAS

To sum up, we give some guidelines about how to model a case in ALIAS.

- *Agents.* Used to model the *dramatis personæ* of a case. Each agent incorporates an abductive logic program for internal reasoning, plus a LAILA program for interactions and queries that involve more that one agent. An abductive logic program contains abducibles, definitions, and integrity constraints.

- *Abducibles.* Used to model the content of testimonies and hypotheses, the latter being either reliability hypotheses or other hypotheses adding during the solution of the case (e.g. hypotheses of agreement among other agents)

- *Definitions.* In the detective's program, used to explain hypotheses and to define a set of possible hypotheses that can be used as initial assumptions for a given query. For instance, an initial Δ, or a set of reliable agents. In a witness's program they define the `ask` predicate, used to answer to queries about specific aspects of the case. There could be indeed more definitions that represent the internal reasoning of individuals.

- *Integrity constraints.* In addition to implicit constraints used by the abductive proof (e.g., $\leftarrow \mathtt{h}_{\mathtt{bloody\_nightstick}}, \mathtt{not}\ \mathtt{h}_{\mathtt{bloody\_nightstick}}$), integrity constraints can be used to enforce the explanation of given hypotheses (e.g. $\leftarrow \mathtt{h}_{\mathtt{bloody\_nightstick}}, \mathtt{not}\ \mathtt{expl}(\mathtt{h}_{\mathtt{bloody\_nightstick}})$). As in Thagard [27], integrity constraints are used also to prevent from assuming contradictory hypotheses (e.g., $\leftarrow \mathtt{h}_{\mathtt{peyer\_killed\_knott}}, \mathtt{not}\ \mathtt{expl}(\mathtt{h}_{\mathtt{someone\_else\_killed\_knott}})$; v. Figure 4). There could be indeed more integrity constraints in the body of other agents, and not only in the detective's body. For instance, a witness who declares that two hypotheses cannot be assumed to be true at the same time (e.g., Peyer declares that he was not in the same place as Knott: $\leftarrow \mathtt{h}_{\mathtt{peyer\_on\_I-15}}, \mathtt{h}_{\mathtt{knott\_on\_I-15}}$).

4 Related Work

Our work relates to several research areas of both artificial intelligence and law. In this section, we mainly focus on the work done in artificial intelligence, with particular regard to abduction, logic programming and multi-agent systems.

Among the other approaches to modelling jurisdictional cases by means of abductive frameworks, Thagard in [27] describes a framework (ECHO) for explaining sets of propositions that he calls "evidence". Pieces of evidence are connected together by means of explanatory/contradictory relationships. We already discussed along the paper the main differences between Thagard's modeling of Peyer's case and ours. In particular, we saw how ALIAS/LAILA is based on an abductive logic programming-based framework where the knowledge is distributed, and allows for asking specific queries to the agents/reasoners, without having to consider all the facts in the model as a whole. This allows us to establish specific relations between pieces of evidence and testimonies, and groups of agents. ECHO, on the other hand, is a monolithic system, which does not have the concept of *dramatis personæ*, and is based on a connectionist model which adopts a continuous truth value. The evaluation of the model leads to a unique answer that shows which hypotheses are explanatory for the initial evidence, and which are not, based on a certain fixed threshold.

Josephson et al. also developed an abductive inference system called *Peirce* [10, 22], a domain independent abductive reasoner that also makes use of a connectionist model. Peirce generates "highly confidencial" hypotheses starting from an input constituted by a set of findings to be explained, together with some initial scores. As opposed to Thagard's ECHO, Peirce allows for assigning initial likelihood values to the hypotheses, and is based on the concept of "essentials", i.e., hypotheses that must be assumed true, since they are the only ones that explain some specific findings. Peirce's algorithm progressively enlarges the set of essentials and rules out hypotheses that are incompatible with them. The authors claim that this is faster than ECHO.

In our work, we do not make any statement regarding efficiency, but we focus on the expressive power of the agent metaphor. This allows to propose a model where different pieces of software have different roles, which reflect their role in the real world case. This has several advantages, not only in terms of model, but also in terms of flexibility and compositionality of the whole framework. We could imagine agents as components that can be re-used to deal with different cases: for instance, agents enclosing whole *corpora* of laws/regulations of some countries, or agents that represent organizations, or that reproduce the behaviour of other pieces of software. Although our implementation is only at a prototypical stage, we believe that the multi-agent approach based on abductive logic programming has the potential to become a flexible tool to model real world cases.

With regard to abductive logic programming, this approach has already been adopted to legal reasoning [24, 25], which we consider a further confirmation with respect to our choices made. In particular, we would like to cite the work by Satoh in [24] where the author proposes a method to translate case-based reasoning into abductive logic programming and shows how it is possible to model itentions of people involved in a trial to explain given observations. In [25], Satoh implements a dynamic notion of similarity of cases in legal reasoning. The system translates a legal case into an abductive logic program, possibly retrieves a similar case, and produces an explanation why the current case is similar to the retrieves one. This work could be used a support for making a library of cases, that could help in modelling legal cases and use them in ALIAS. Argumentation theory has also been estensively used in legal reasoning for instance to reconcile conflicting arguments in a similar way to what we do by means of logic programming [23]. We are not going to survey work on argumentation since it is out of the scope of this paper: however the interested reader in the relationship between abduction and argumentation can refer to [14]. For recent surveys on abduction in Logic Programming, see [12] and [14]. For a discussion on other work related to ALIAS/LAILA, see [2, 3].

5 Conclusion

In this paper we have presented a new multi-agent approach for legal justification. We have shown that the process of explaining evidences and other observation in a trial could profit from the use of logic-based abduction within the framework of a multi-agent architecture. In summary the main features of this approach are:

- the declarative representation of trial knowledge, by means of abductive logic programming, allows to express knowledge and reasoning criteria at a very high level.
- mapping distinct trial characters into distinct agents allows the modular and dynamic composition of distinct sources of knowledge; in this way it is possible

to perform collaborative/competitive reasoning on trial data, using the knowledge of two or more *dramatis personæ* to obtain a consistent set of hypotheses that explains the given data;

- agents collaborative explanation allows also the detection of incoherence and collusion among trial characters, and can produce reliability hypotheses about people implicated in the trial; these reliability hypotheses could drive detective's enquiry activity towards the final solution of the case.

As a matter of future work, we plan to extend the ALIAS architecture towards the introduction of probability values as scores associated to collaborative/competitive abductive explanations. In this respect, we have already studied the integration of the Probabilistic Horn Abduction theory [21] with abductive logic agents, in particular with the aim of performing distributed medical diagnosis. Therefore we will adapt that approach, if possible, to the case of legal justification and then we will possibly implement it by extending the ALIAS implementation.

Acknowledgements

This work was supported by the SOCS project, funded by the CEC, contract IST-2001-32530. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] A. Brogi, E. Lamma, P. Mancarella, and P. Mello. A unifying view for logic programming with non-monotonic reasoning. *Theoretical Computer Science*, 184:1–49, 1997.

[2] A. Ciampolini, E. Lamma, P. Mello, F. Toni, and P. Torroni. Co-operation and competition in *ALIAS*: a logic framework for agents that negotiate. *Annals of Mathematics and Artificial Intelligence*, to appear.

[3] A. Ciampolini, E. Lamma, P. Mello, and P. Torroni. LAILA: A language for coordinating abductive reasoning among logic agents. *Computer Languages*, 27(4):137–161, February 2002.

[4] K. L. Clark. *Negation as Failure*, chapter Logic and Databases. Plenum Press, New York, 1978.

[5] L. Console, D. Theseider Dupré, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[6] P. T. Cox and T. Pietrzykowski. Causes for events: Their computation and applications. In *Proceedings CADE-86*, page 608, 1986.

[7] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.

[8] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[9] K. Eshgi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proceedings 6th International Conference on Logic Programming*, page 234. MIT Press, 1989.

[10] R. Fox and J. Josephson. Peirce-IGTT: A domain-independent problem solver for abdcutive assembly. Technical report, The Laboratory for Artificial Intelligence Research, The Ohio State University, 1990.

[11] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 1997.

[12] A. Kakas and M. Denecker. Abduction in logic programming. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond. Part I*, number 2407 in LNAI, pages 402–436. Springer Verlag, 2002.

[13] A. C. Kakas. ACLP: integrating abduction and constraint solving. In *Proceedings NMR'00, Breckenridge, CO*, 2000.

[14] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. *Handbook of Logic in AI and Logic Programming*, 5:235–324, 1998.

[15] A. C. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proceedings 9th European Conference on Artificial Intelligence*. Pitman Pub., 1990.

[16] A. C. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of the first Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, pages 438–443, 1990.

[17] R. A. Kowalski. Problems and promises of computational logic. In *Proceedings Symposium on Computational Logic*, pages 1–36. Springer-Verlag, November 1990.

[18] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[19] M.Gini and T. Ishida (eds.). *Proceedings of the first Joint Conference on Autonomous Agents and Multiagent Systems*. ACM Press, 2002.

[20] D. L. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[21] D. L. Poole. Logic programming, abduction and probability:a top down anytyme algorithm for estimating prior and posterior probabilities. *Artificial Intelligence*, 64:81–129, 1993.

[22] W.F. Punch III, M.C. Tanner, J.R. Josephson, and J.W. Smith. Peirce: A tool for experimenting with abduction. *IEEE Expert*, 5(5):34–44, 1990.

[23] R.A.Kowalski and F. Toni. Argument and reconciliation. In *Proceedings of Workshop on Legal Reasoning International Symposium on FGCS Tokyo*, 1994.

[24] K. Satoh. Translating case-based reasoning into abductive logic programming. In *Proceedings European Conference on Artificial Intelligence (ECAI)*. John Wiley & Sons, Ltd., 1996.

[25] K. Satoh. Using two level abduction to decide similarity of cases. In *Proceedings 13th European Conference on Artificial Intelligence*, pages 398–402. John Wiley and Sons, Chichester, 1998.

[26] P. Tarau. Jinni: Intelligent mobile agent programming at the intersection of java and prolog. In *Proceedings PAAM'99, London, UK*, 1999.

[27] P. Thagard. Explanatory coherence. *Behavioral and Brain Sciences*, (3):435–502, 1989.

[28] F. Toni. A semantics for the Kakas-Mancarella procedure for abductive logic programming. In M. Alpuente and M. I. Sessa, editors, *Proc. GULP'95*, pages 231–242, 1995.

[29] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.