Abduction with Hypotheses Confirmation

Marco Alberti[1], Marco Gavanelli[1], Evelina Lamma[1],
Paola Mello[2], and Paolo Torroni[2]

[1] DI - University of Ferrara - Via Saragat, 1 - 44100 Ferrara, Italy.
{malberti|mgavanelli|elamma}@ing.unife.it
[2] DEIS - University of Bologna - Viale Risorgimento, 2 - 40136 Bologna, Italy.
{pmello|ptorroni}@deis.unibo.it

Abstract. Abduction can be seen as the formal inference corresponding to human hypothesis making. It typically has the purpose of explaining some given observation. In classical abduction, hypotheses could be made on events that may have occurred in the past. In general, abductive reasoning can be used to generate hypotheses about events possibly occurring in the future (forecasting), or may suggest further investigations that will confirm or disconfirm the hypotheses made in a previous step (as in scientific reasoning). We propose an operational framework based on Abductive Logic Programming, which extends existing frameworks in many respects, including accommodating dynamic observations and hypothesis confirmation.

1 Introduction

Often, reasoning paradigms in artificial intelligence mimic human reasoning, providing a formalization and a better understanding of the human basic inferences. Abductive reasoning can be seen as a formalization, in computational logics, of hypotheses making. In order to explain observations, we hypothesize that some (unknown) events have happened, or that some (not directly measurable) properties hold true. The hypothesized facts are then assumed as true, unless they are disconfirmed in the following.

Hypothesis making is particularly important in scientific reasoning: scientists will hypothesize properties about nature, which explain some observations; in subsequent work, they will try to prove (if possible), or at least to confirm the hypotheses. This process leads often to generating new alternative sets of hypotheses. Starting from hypotheses on the current situation, scientists try to foresee their possible consequences; this provides new hypotheses on the future behavior that will be confirmed or disconfirmed by the actual events.

A typical application of abductive reasoning is *diagnosis*. Starting from the observation of symptoms, physicians hypothesize in general possible alternative diseases that may have caused them. Following an iterative process, they will try to support their hypotheses, by prescribing further exams, of which they foresee the possible alternative results. They will then drop the hypotheses disconfirmed by such results, and take as most faithful those supported by them. New findings, such as results of exams or new symptoms, may help generating new hypotheses.

We can then describe this kind of hypothetical reasoning process as composed by three main elements: classically, *explaining observations*, by assuming possible

causes of the observed effects; but also, *adapting* such assumptions to upcoming events, such as new symptoms occurring, and *foreseeing* the occurrence of new events, which may or may not occur indeed.

In Abductive Logic Programming, many formalisms have been proposed [1–6], along with proof procedures able to provide, given a knowledge base and some observation, possible sets of hypotheses that explain the observation. Integrity Constraints are used to drive the process of hypothesis generation, to make such sets consistent, and possibly to suggest new hypotheses. Most frameworks focus on one aspect of abductive reasoning: assumption making, based on a static knowledge and on some observation.

In this work, we extend the concepts of abduction and abductive proof procedures in two main directions.

First, we cater for the dynamic acquisition of new facts (events), which possibly have an impact on the abductive reasoning process, and for confirmation (or disconfirmation) of hypotheses based on such events. We propose a language, able to state required properties of the events supporting the hypotheses: for instance, we could say that, given some combination of hypotheses and facts, we make the hypothesis that some new events will occur. We call this kind of hypothesis *expectation*. For this purpose, we express expectations as abducible literals. Expectations can be "positive (to be confirmed by certain events occurring), or "negative" (to be confirmed by certain events not occurring).

Second, in our framework, we need to be able to state that some event is expected to happen *within some time interval*: if the event does actually happen within it, the hypothesis is confirmed, it is disconfirmed otherwise. In doing so, we need to introduce *variables* (e.g. to model time), and to state *constraints* on variables occurring in abducible atoms. Moreover, possible expectation could be involving universal quantification: this typically happens with negative expectations ("The patient is expected *not* to show symptom Q *at all times*"). For this reason, we also need to cater for abducibles possibly containing universally quantified variables.

To summarize, the main new features of the present work with respect to classical ALP frameworks are:

- dynamic update of the knowledge base to cater for new events, whose occurrence interacts with the abductive reasoning process itself;
- confirmation and disconfirmation of hypotheses, by matching expectations and actual events;
- hypotheses with universally quantified variables;
- constraints à la Constraint Logic Programming [7].

We achieve these features by defining syntax, declarative and operational semantics of an abductive framework, based on an extension of the IFF proof procedure [4], called SCIFF[8].[3] Being the SCIFF an extension of an existing abductive

[3] Historically, the name SCIFF is due to the fact that this framework has been firstly applied to modelling protocol in agent *S*ocieties, and that is also deals with CLP *C*onstraints.

framework, it also caters for classic abductive logic programming (static knowledge, no notion of confirmation by events). However, due to space limitations, in this work we only focus on the original new parts.

The *S*CIFF has been implemented using *Constraint Handling Rules* [9] and integrated in a Java-based system for hypothetical reasoning [10].

In the following Sect. 2 we introduce our framework's knowledge representation. In Sect. 3 and 4 we provide declarative and operational semantics, and we show a soundness result. In Sect. 5 we show an example of the functioning of the *S*CIFF in a multi-agent setting. Before concluding, we discuss about related work in Sect. 6.

Additional details about the syntax of data structures used by the *S*CIFF and allowedness criteria used to prove soundness are given in [11].

2 Knowledge Representation

In this section we show the knowledge representation of the abstract abductive framework of the *S*CIFF.

As typical abductive frameworks [12], ours is representated by a 3-tuple $\langle DKB, \mathcal{IC}, A \rangle$ where:

- DKB is the (dynamic) knowledge base, union of KB (a logic program, possibly containing abducibles in the body of the clauses, which does not change during the computation) and **HAP** (the dynamic part, as explained below);
- $\mathcal{A}$ is the set of abducible predicates (in our case, **E**, **EN** and their explicit negations $\neg$**E** and $\neg$**EN**); and
- $\mathcal{IC}$ is the set of *Integrity Constraints*.

The set **HAP** is composed of ground facts (defining the **H** predicate), of the form

$$\mathbf{H}(Description[, Time])$$

where *Description* is a ground term representing the event that happened and *Time* is an integer number representing the time at which the event happened.

The history **HAP** dynamically grows during the computation, as new events happen; we do not model here the source of such events, but it can be imagined as a queue. We assume that events arrive in the same temporal order in which they happen.

The evolution of the history **HAP** reflects in the evolution of the abductive framework, of which **HAP** is part. We can formalize the evolution as a sequence *DALP* of frameworks, and denote with $DALP_{\mathbf{HAP}}$ the specific instance associated to a specific history **HAP**.

An instance $DALP_{\mathbf{HAP}}$ of an abductive framework is queried with a *goal* $\mathcal{G}$. The goal may contain both predicates defined in KB and abducibles.

The computation of an instance of an abductive framework will produce, by abduction, a set **EXP** of *expectations*, which represent events that are expected to (but might not) happen (*positive* expectations, of the form $\mathbf{E}(Description[, Time])$)

and events that are expected *not* to (but might) happen (*negative* expectations, of the form $\mathbf{EN}(Description[, Time])$). Typically, expectations will contain variables, over which CLP [7] constraints can be imposed.

The full syntax of our language is reported in [11].

We conclude this section with a simple example in the medical domain, where abduction is used to diagnose diseases starting from symptom observation. The aim of this example is to show the two main improvements of the *S*CIFF with respect to previous work: the dynamic acquisition of new facts, and the confirmation of hypotheses by events.

Example 1. Let us consider a symptom s, which can be explained by abducing one of three types of diseases, of which the first and the third are incompatible, and the second is accompanied by a condition (the patient's temperature is expected to increase):

$$symptom(s) :- \mathbf{E}(disease(d_1)), \mathbf{EN}(disease(d_3)).$$
$$symptom(s) :- \mathbf{E}(disease(d_2)), \mathbf{E}(temperature(high)).$$
$$symptom(s) :- \mathbf{E}(disease(d_3)), \mathbf{EN}(disease(d_1)).$$

The set $\mathcal{IC}$ of integrity constraints expresses what is expected or *should* happen or not, given some happened events and/or some abduced hypotheses. They are in the form of implications, and can involve both literals defined in the KB, and expectations and events in **EXP** and **HAP**. For example, an IC in $\mathcal{IC}$ could state that if the result of some exam r is positive, then we can hypothesize that the patient is not affected by disease d_1:

$$\mathbf{H}(result(r, positive)) \rightarrow \mathbf{EN}(disease(d_1))$$

If $\mathbf{H}(result(r, positive))$ actually happens, the integrity constraint would make us abduce $\mathbf{EN}(disease(d_1))$, which would rule out, in our framework, the possibility to abduce $\mathbf{E}(disease(d_1))$. We see how the dynamic occurrence of new events can drive the generation and selection of abductive explanations of goals. Let us now assume that the patient, at some point, shows the symptom $temperature(low)$. The following constraint can be used to express this fact to be inconsistent with an expectation about his temperature increasing:

$$\mathbf{E}(temperature(high)) \rightarrow \mathbf{EN}(temperature(low))$$

If the diagnosis $\mathbf{E}(disease(d_2)), \mathbf{E}(temperature(high))$ is chosen for s, this IC would have as a consequence the generation of the expectation $\mathbf{EN}(temperature(low))$, which would be frustrated by the fact $\mathbf{H}(temperature(low))$. The only possible explanation for s thus remains $\mathbf{E}(disease(d_3)), \mathbf{EN}(disease(d_1))$. We see by this example how the hypotheses can be disconfirmed by events.

The abductive system will usually have a goal, which typically is an observation for which we are searching for explanations; for example, a conjunction of *symptom* atoms.

3 Declarative semantics

In the previous section, we have defined an instance $DALP_{\mathbf{HAP}}$ of an abductive framework as a tuple $\langle DKB, \mathcal{A}, \mathcal{IC}, \rangle$, where $DKB = KB \cup \mathbf{HAP}$. In this section, we propose an abductive interpretation for $DALP_{\mathbf{HAP}}$, depending on the events in the history **HAP**. We adopt a three-valued logic, where literals of kind $\mathbf{H}()$ or $\neg\mathbf{H}()$ can be interpreted as true, false or unknown: when reasoning about future events, it is not possible to state their happening or non-happening.

Throughout this section, as usual when defining declarative semantics, we always consider the ground version of the knowledge base and integrity constraints, and consider CLP-like constraints as defined predicates.

Since an instance $DALP_{\mathbf{HAP}}$ is an abductive logic program, an abductive explanation should entail the goal and satisfy the integrity constraints:

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models G \tag{1}$$

$$KB \cup \mathbf{HAP} \cup \mathbf{EXP} \models IC \tag{2}$$

where, as in the IFF proof procedure [4], the symbol $\models$ stands for three valued completion semantics. Notice, however, that we do not complete the set **HAP**, as it would imply that events no events will happen in the future. In other words, the closed world assumption cannot hold for future events.

Among the sets of expectations computed as abductive explanations for an instance $DALP_{\mathbf{HAP}}$, we select the ones that are consistent with respect to the expected events (i.e., we do not want the same event to be both expected to happen and expected not to happen) and to negation:

Definition 1. *A set of expectations* **EXP** *is* E-consistent *if and only if for each (ground) term p:* $\{\mathbf{E}(p), \mathbf{EN}(p)\} \not\subseteq \mathbf{EXP}$.

A set of expectations **EXP** *is* ¬-consistent *if and only if for each (ground) term p:* $\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \mathbf{EXP}$ *and* $\{\mathbf{EN}(p), \neg\mathbf{EN}(p)\} \not\subseteq \mathbf{EXP}$.[4]

Finally, we require that the set of expectations computed as an abductive explanation for an instance $DALP_{\mathbf{HAP}}$ be *confirmed*:

Definition 2. Confirmation. *Given a history* **HAP**, *a set of expectations* **EXP** *is* confirmed *if and only if for each (ground) term p:*

$$\mathbf{HAP} \cup Comp(\mathbf{EXP}) \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \rightarrow \neg\mathbf{H}(p)\} \cup CET \not\models \mathit{false} \tag{3}$$

If Eq. 3 does not hold, the set of expectations is called disconfirmed.

Definition 2 requires that each negative expectation in **EXP** have no corresponding happened event; if there is one, **EXP** is disconfirmed. Disconfirmation of a positive expectation, instead, can only occur if some deadline on the expectation (expressed by CLP constraints on the time variable) is missed; otherwise, an event confirming the expectation may always occur in the future.

[4] For abducibles, we adopt the same viewpoint as in ACLP [5]: for each abducible predicate A, we have also the abducible predicate $\neg A$ for the negation of A together with the integrity constraint $(\forall X)\neg A(X), A(X) \rightarrow \bot$.

4 Operational Semantics

Our framework's IC are very much related to the integrity constraints of the IFF proof procedure [4]. This leads to the idea of using an extension of the IFF proof procedure for generating expectations, and check for their confirmation.

In particular, the additional features that we need are the following: (i) accept new events as they happen, (ii) produce a (disjunction of) set of expectations, (iii) detect confirmation of expectations, (iv) detect disconfirmation as soon as possible.

The proof procedure that we are about to present is called SCIFF. Following Fung and Kowalski's approach [4], we describe the SCIFF as a transition system. Due to space limitations, we will only focus here on the new transitions, while the reader can refer to [4] for the basic IFF transitions.

4.1 Data Structures

The SCIFF proof procedure is based on a transition system. Each state is defined by the tuple $T \equiv \langle R, CS, PSIC, \mathbf{EXP}, \mathbf{HAP}, \mathbf{CONF}, \mathbf{DISC} \rangle$, where R is the resolvent, CS is the constraint store, $PSIC$ is the set of partially solved integrity constraints, **EXP** is the set of (pending) expectations, **HAP** is the history of happened events, **CONF** is a set of confirmed hypotheses, **DISC** is a set of disconfirmed expectations.

Variable quantification In the IFF proof procedure, all the variables that occur in the resolvent or in abduced literals are existentially quantified, while the others (appearing only in implications) are universally quantified. Our proof procedure has to deal with universally quantified variables in the abducibles and in the resolvent. In the IFF proof procedure, variables in an implication are existentially quantified if they also appear in an abducible or in the resolvent. In our language, we can have existentially quantified variables in the integrity constraints even if they do not occur elsewhere (see [11]).

For all these reasons, in the operational semantic specification we leave the variable quantification explicit. Moreover, we need to distinguish between variables that appear in abduced literals (or in the resolvent) and variables occurring only in integrity constraints. The scope of the variables that occur only in an implication is the implication itself; the scope of the other variables is the whole tuple.

Initial Node and Success A derivation D is a sequence of nodes

$$T_0 \rightarrow T_1 \rightarrow \ldots \rightarrow T_{n-1} \rightarrow T_n.$$

Given a goal G and a set of integrity constraints $\mathcal{IC}$, the first node is:

$$T_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

i.e., the resolvent is initially the query ($R_0 = \{G\}$) and the partially solved integrity constraints $PSIC$ is the set of integrity constraints ($PSIC_0 = \mathcal{IC}$).

The other nodes $T_j, j > 0$, are obtained by applying the transitions defined in the next section, until no transition can be applied anymore (quiescence). Every arc in a derivation is labelled with the name of a transition.

Definition 3. *Starting with an instance* $DALP_{\mathbf{HAP}^i}$ *there exists a* successful derivation *for a goal* G *iff the proof tree with root node* $\langle \{G\}, \emptyset, \mathcal{IC}, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$ *has at least one leaf node* $\langle \emptyset, CS, PSIC, \mathbf{EXP}, \mathbf{HAP}^f, \mathbf{CONF}, \emptyset \rangle$ *where* $\mathbf{HAP}^f \supseteq \mathbf{HAP}^i$ *and* CS *is consistent (i.e., there exists a ground variable assignment such that all the constraints are satisfied). In that case, we write:*

$$DALP_{\mathbf{HAP}^i} \mid\!\sim^{\mathbf{HAP}^f}_{\mathbf{EXP} \cup \mathbf{CONF}} G$$

Notice that, coherently with the declarative semantics given earlier, a success node cannot contain disconfirmed hypotheses. However, in some applications, all the alternative sets may contain disconfirmed expectations and the goal could be finding a set of expectations with a minimal set of disconfirmed ones. For this reason, we preferred to map explicitly the set of disconfirmed expectations, instead of generating a simple *fail* node (paving the way for future extensions of the framework). Moreover, classical Logic Programming provides explanations (e.g., variable binding) about why a computation succeeds, but no explanation for failure nodes. In our framework, a failure can be explained by means of alternative sets of disconfirmed expectations.

From a non-failure leaf node N, answers can be extracted in a very similar way to the IFF proof procedure. First, a substitution σ' is computed such that σ' replaces all variables in N that are not universally quantified by ground terms, and σ' satisfies all the constraints in the store CS_N.

Definition 4. *Let* $\sigma = \sigma'|_{vars(G)}$ *be the restriction of* σ' *to the variables occurring in the initial goal* G*. Let* $\Delta = [\mathbf{CONF}_N \cup \mathbf{EXP}_N)]\sigma'$*. The pair* (Δ, σ) *is the* abductive answer *obtained from the node* N*.*

The SCIFF proof procedure performs some inferences based on the semantics of time, under the temporal order assumption (see Sect. 2).

Consistency In order to ensure **E**-consistency of the set of expectations, we require that the set $\mathcal{IC}$ of integrity constraints always contain the following:

$$\mathbf{E}(T) \wedge \mathbf{EN}(T) \rightarrow \bot$$

while for $\neg$-consistency, we impose the following integrity constraints:

$$\mathbf{E}(T) \wedge \neg\mathbf{E}(T) \rightarrow \bot \qquad \mathbf{EN}(T) \wedge \neg\mathbf{EN}(T) \rightarrow \bot$$

4.2 Transitions

The transitions are those of the IFF proof procedure, enlarged with those of CLP [7], and with specific transitions accommodating the concepts of confirmation of hypotheses, and dynamically growing history.

In this section, the letter k will indicate the level of a node; each transition will generate one or more nodes from level k to $k+1$. We will not explicitly report the new state for items that do not change; e.g., if a transition generates a new node from the node

$$T_k \equiv \langle R_k, CS_k, PSIC_k, \mathbf{EXP}_k, \mathbf{HAP}_k, \mathbf{CONF}_k, \mathbf{DISC}_k \rangle$$

and we do not explicitly state the value of R_{k+1}, it means that $R_{k+1} = R_k$.

IFF-like transitions The *S*CIFF proof procedure contains transitions borrowed from the IFF proof procedure, namely *Unfolding*, *Propagation*, *Splitting*, *Case Analysis*, *Factoring*, *Equivalence Rewriting* and *Logical Equivalence*. They have been extended to cope with abducibles containing universally quantified variables and with CLP constraints. We omit them here for lack of space; the basic transitions were proposed by Fung and Kowalski [4], while the extended ones can be found in a technical report [13].

Dynamically growing history The happening of events is dealt with by a transition *Happening*. This transition takes an event $\mathbf{H}(Event)$ from the external queue and puts it in the history $\mathbf{HAP}$; the transition *Happening* is applicable only if an *Event* such that $\mathbf{H}(Event) \notin \mathbf{HAP}$ is in the external queue.

Formally, from a node N_k transition *Happening* produces a single successor

$$\mathbf{HAP}_{k+1} = \mathbf{HAP}_k \cup \{\mathbf{H}(Event)\}.$$

Note that transition *Happening* should be applied to all the non-failure nodes (in the frontier).

Confirmation and Disconfirmation

Disconfirmation **EN** Given a node N with the following situation:

$$\mathbf{EXP}_k = \mathbf{EXP}' \cup \{\mathbf{EN}(E_1)\} \qquad \mathbf{HAP}_k = \mathbf{HAP}' \cup \{\mathbf{H}(E_2)\}$$

Disconfirmation **EN** produces two nodes N^1 and N^2, as follows:

N^1	N^2
$\mathbf{EXP}^1_{k+1} = \mathbf{EXP}'$	$\mathbf{EXP}^2_{k+1} = \mathbf{EXP}_k$
$\mathbf{DISC}^1_{k+1} = \mathbf{DISC}_k \cup \{\mathbf{EN}(E_1)\}$	$\mathbf{DISC}^2_{k+1} = \mathbf{DISC}_k$
$CS^1_{k+1} = CS_k \cup \{E_1 = E_2\}$	$CS^2_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

Remember that node N^1 is a failure node, as in success nodes $\mathbf{DISC} = \emptyset$ (see Sect. 3).

Example 2. Suppose that $\mathbf{HAP}_k = \{\mathbf{H}(p(1,2))\}$ and $\exists X \forall Y \mathbf{EXP}_k = \{\mathbf{EN}(p(X,Y))\}$. *Disconfirmation* **EN** will produce the two following nodes:

$\exists X \forall Y \mathbf{EXP}_k = \{\mathbf{EN}(p(X,Y))\}\ \mathbf{HAP}_k = \{\mathbf{H}(p(1,2))\}$

$CS^1_{k+1} = \{X = 1 \wedge Y = 2\}\ \ CS^2_{k+1} = \{X \neq 1 \vee Y \neq 2\}$
$\mathbf{DISC}^1_{k+1} = \{\mathbf{EN}(p(1,2))\}$
$CS_{k+2} = \{X \neq 1\}$

where the last simplification in the right branch is due to the rules of the constraint solver (see Section CLP).

Confirmation **E** Starting from a node N as follows:

$$\mathbf{EXP}_k = \mathbf{EXP}' \cup \{\mathbf{E}(E_1)\}, \qquad \mathbf{HAP}_k = \mathbf{HAP}' \cup \{\mathbf{H}(E_2)\}$$

Confirmation **E** builds two nodes, N^1 and N^2; in node N^1 we assume that the expectation and the happened event unify, and in node N^2 we hypothesize that the two do not unify:

N^1			N^2		
$\mathbf{EXP}^1_{k+1}$	=	$\mathbf{EXP}'$	$\mathbf{EXP}^2_{k+1}$	=	$\mathbf{EXP}_k$
$\mathbf{CONF}^1_{k+1}$	=	$\mathbf{CONF}_k \cup \{\mathbf{E}(E_1)\}$	$\mathbf{CONF}^2_{k+1}$	=	$\mathbf{CONF}_k$
CS^1_{k+1}	=	$CS_k \cup \{E_1 = E_2\}$	CS^2_{k+1}	=	$CS_k \cup \{E_1 \neq E_2\}$

In this case, N^2 is not a failure node, as $\mathbf{E}(E_1)$ might be confirmed by other events.

Disconfirmation **E** In order to check the disconfirmation of a positive expectation **E**, we need to assume that there will never be a matching event in the external queue. We can, e.g., exploit the semantics of *time*. If we make the hypothesis of temporal ordering (Sect. 2), we can infer that an expected event for which the deadline is passed, is disconfirmed.

Given a node:

- $\mathbf{EXP}_k = \{\mathbf{E}(X,T)\} \cup \mathbf{EXP}'$
- $\mathbf{HAP}_k = \{\mathbf{H}(Y,T_c)\} \cup \mathbf{HAP}'$
- $\forall E_2, T_2 : \mathbf{H}(E_2,T_2) \in \mathbf{HAP}_k,\ CS_k \cup \{(E_2,T_2) = (X,T)\} \models \bot$
- $CS_k \models T < T_c$

transition Disconfirmation **E** is applicable and creates the following node:

- $\mathbf{EXP}_{k+1} = \mathbf{EXP}'$
- $\mathbf{DISC}_{k+1} = \mathbf{DISC}_k \cup \{\mathbf{E}(X,T)\}$.

Operationally, one can often avoid checking that (X,T) does not unify with every event in the history by choosing a preferred order of application of the transitions. By applying Disconfirmation **E** only if no other transition is applicable, the check can be safely avoided, as the test of confirmation is already performed by *Confirmation* **E**.

Notice that this transition infers the current time from happened event; i.e., it infers that the current time cannot be less than the time of a happened event.

Symmetrically to Disconfirmation **E**, we also have a transition *Confirmation* **EN**, which we do not report here for lack of space; the interested reader is referred to [13].

Note that the entailment of constraints from a constraint store is, in general, not easy to verify. In the particular case of $CS_k \models T < T_c$, however, we have that the constraint $T < T_c$ is unary (T_c is always ground), thus a CLP for finite domains solver CLP(FD) is able to verify the entailment very easily if the store contains only unary constraints (it is enough to check the maximum value in the domain of T). Moreover, even if the store contains non-unary constraints (thus the solver performs, in general, incomplete propagation), the transition will not compromise the soundness and completeness of the proof procedure. If the solver performs a powerful propagation (including pruning, in CLP(FD)), the disconfirmation will be early detected, otherwise, it will be detected later on.

CLP Here we adopt the same transitions as in CLP [7]. Therefore, the symbols $=$ and $\neq$ are in the constraint language. Note that a constraint solver works on a constraint domain which has an associated interpretation. In addition, the constraint solver should handle the constraints among terms derived from the unification. Therefore, beside the specific constraint propagation on the constraint domain, we need further inference rules for coping with the unification. For space limitations, we cannot show them here: but they can be found in [11].

4.3 Soundness

The following proposition relates the operational notion of successful derivation with the corresponding declarative notion of goal provability.

Proposition 1. *Given an instance* $ALP_{\mathbf{HAP}^i}$ *of an ALP program and a ground goal* G*, if* $DALP_{\mathbf{HAP}^i} \vdash^{\mathbf{HAP}^f}_{\mathbf{EXP}\cup\mathbf{CONF}} G$ *then* $DALP_{\mathbf{HAP}^f} \models_{\mathbf{EXP}\cup\mathbf{CONF}} G$.

This property has been proven for some notable classes of ALP programs. In particular, a proof of soundness can be found in [13] for *allowed* ALPs (for a definition of allowedness see [11]). The proof is based on a correspondence drawn between the *S*CIFF and IFF transitions, and exploits the soundness results of the IFF proof procedure [4].

5 Using the *S*CIFF for agent compliance verification

Abduction has been used for various applications, and many of them (e.g. diagnosis) could benefit from an extension featuring hypotheses confirmation, such as the one depicted in this paper. We have applied the language to a multi-agent setting, in the context of the SOCS project [14].

In order to combine autonomous agents and have them operate in a coordinated fashion, protocols are often defined. Protocols show, in a way, the ideal behaviour of agents. But, since agents are often assumed to be autonomous, and societies open and heterogeneous, agent compliance to protocols is rarely a reasonable assumption to make. Agents may violate the protocols due to malicious

intentions, to wrong design or, for instance, to failure to keep the pace with tight deadlines.

With our language, protocols can be easily formalized, and the *S*CIFF proof procedure can be used then to check whether the agents comply to protocols. For instance, let us consider the (very simple) *query-ref* protocol: an agent requests a piece of information to another agent, which, by a given deadline, should either provide it or refuse it, but not both. The protocol can be expressed by the following three integrity constraints (where D represents a dialogue identifier):

$\mathbf{IC_1}$: $\mathbf{H}(tell(A, B, \textit{query-ref}(\textit{Info}), D), T) \wedge \textit{deadline}(T_\mathrm{d}) \rightarrow$
$\mathbf{E}(tell(B, A, \textit{inform}(\textit{Info}, \textit{Answer}), D), T_1) \wedge T_1 \leq T + T_\mathrm{d} \vee$
$\mathbf{E}(tell(B, A, \textit{refuse}(\textit{Info}), D), T_1) \wedge T_1 \leq T + T_\mathrm{d}$

$\mathbf{IC_2}$: $\mathbf{H}(tell(A, B, \textit{inform}(\textit{Info}, \textit{Answer}), D), T) \rightarrow$
$\mathbf{EN}(tell(A, B, \textit{refuse}(\textit{Info}), D), T_1) \wedge T_1 \geq T$

$\mathbf{IC_3}$: $\mathbf{H}(tell(A, B, \textit{refuse}(\textit{Info}), D), T) \rightarrow$
$\mathbf{EN}(tell(A, B, \textit{inform}(\textit{Info}, \textit{Answer}), D), T_1) \wedge T_1 \geq T$

IC_1 expresses that an agent that receives a *query-ref* must reply with either an *inform* or a *refuse* by T_d time units; IC_2 and IC_3 state that an agent that performs an *inform* cannot perform a *refuse* later, and *vice-versa*. Predicate $\textit{deadline}/1$ is defined in the KB: in this example, let it be defined by the one fact $\textit{deadline}(10)$.

Let us suppose that the following events happen:

$\mathbf{H_1}$: $\mathbf{H}(tell(alice, bob, \textit{query-ref}(\textit{what-time}), d_0), 10)$

$\mathbf{H_2}$: $\mathbf{H}(tell(bob, alice, \textit{refuse}(\textit{what-time}), d_0), 15)$

and consider how *S*CIFF verifies that such an history is compliant to the interaction protocol.

The $\mathbf{H_1}$ event, by propagation of $\mathbf{IC_1}$ (where, by unfolding, T_d has been bound to 10), will cause a disjunction of two expectations to be generated, which will split the proof tree into two branches:

1. In the first branch, $\mathbf{EXP} = \{\mathbf{E}(tell(bob, alice, \textit{inform}(\textit{what-time}, \textit{Answer}), d_0), T_1)\}$ and $\mathbf{CS} = \{T_1 \leq 20\}$. When $\mathbf{H_2}$ happens, by propagation of $\mathbf{IC_2}$, $\mathbf{EXP} = \{\mathbf{E}(tell(bob, alice, \textit{inform}(\textit{what-time}, \textit{Answer}), d_0), T_1),$ $\mathbf{EN}(tell(bob, alice, \textit{inform}(\textit{what-time}, \textit{Answer}), d_0), T_2)\}$, with $\mathbf{CS} = \{T_1 \leq 20, T_2 \geq 15\}$. Then, by enforcing *E*-consistency, the domain of T_1 is reduced: $\mathbf{CS} = \{T_1 \leq 14, T_2 \geq 15\}$. Now, *Disconfirmation*-$\mathbf{E}$ can be applied: $\mathbf{E}(tell(bob, alice, \textit{inform}(\textit{what-time}, \textit{Answer}), d_0), T_1)$ is moved from **EXP** to **DISC**; this means that this branch cannot be successful.
2. In the second branch, $\mathbf{EXP} = \{\mathbf{E}(tell(bob, alice, \textit{refuse}(\textit{what-time}), d_0), T_1)\}$ and $\mathbf{CS} = \{T_1 \leq 20\}$. By propagation of $\mathbf{IC_2}$, after $\mathbf{H_2}$, $\mathbf{EXP} = \{\mathbf{E}(tell(bob, alice, \textit{refuse}(\textit{what-time}), d_0), T_1),$ $\mathbf{EN}(tell(bob, alice, \textit{inform}(\textit{what-time}, \textit{Answer}), d_0), T_2)\}$, with $\mathbf{CS} = \{T_1 \leq 20, T_2 \geq 15\}$. By *Confirmation* $\mathbf{E}$, $\mathbf{H_2}$ also causes $\mathbf{E}(tell(bob, alice, \textit{refuse}(\textit{what-time}), d_0), T_1)$ to be moved from **EXP** to **CONF**, with $T_1 = 15$. If no more events happen, this branch is successful.

Through this simple example, we showed how a protocol can be easily cast in our model. More rules can be easily added, to accomplish more complex protocols. In other work, we have shown the application of this formalism to a range of protocols [15, 16]. The use of expectations generated by the *S*CIFF could be

manifold: by associating Confirmation/Disconfirmation with a notion of Fulfillment/Violation, we can verify at run-time the compliance of agents to protocols. Moreover, expectations, if made public, could be used by agents planning their activities, helping their choices if they aim at exhibiting a compliant behaviour.

6 Related Work

This work is mostly related to the IFF proof procedure [4], which it extends in several directions, as stated in the introduction.

Other proof procedures deal with constraints; in particular we mention ACLP [5] and the $\mathcal{A}$-system [6], which are deeply focussed on efficiency issues. Both of these proof procedures use integrity constraints in the form of denials (e.g., $A, B, C \rightarrow \bot$), instead of forward rules as the IFF (and SCIFF). Both of these proof procedures only abduce existentially quantified atoms, and do not consider quantifier restrictions, which make the SCIFF in this sense more expressive.

Some conspicuous work has been done with the integration of the IFF proof procedure with constraints [17]; however the integration is more focussed on a theoretical uniform view of abducibles and constraints than to an implementation of a proof procedure with constraints.

In [18], Endriss et al. present an implementation of an abductive proof procedure that extends IFF [4] in two ways: by dealing with constraint predicates and with non-allowed abductive logic programs. The cited work, however, does not deal with confirmation and disconfirmation of hypotheses and universally quantified variables in abducibles (**EN**), as ours does. The two works also differ in their implementation: Endriss et al.'s is a metainterpreter which exploits a built-in constraint solver, whereas we implement the proof transitions and variable unification by means of CHR and attributed variables. Both works have been conducted in the context of the SOCS project [14]: the main application of Endriss et al.'s is the implementation of the internal agent reasoning, while ours is the compliance check of the observable (external) agent behaviour.

Abdual [19] is a systems for performing abduction from extended logic programs adopting the well-founded semantics. It handles only ground programs. It relies on tabled evaluation and is inspired to SLG resolution [20].

Many other abductive proof procedures have been proposed in the past; the interested reader can refer to the exhaustive survey by Kakas et al. [12].

In [21], Sergot proposed a general framework, called *query-the-user*, in which some of the predicates are labelled as "askable"; the truth of askable atoms can be asked to the user. The framework provides, thus, the possibility of gathering new information during the computation. Our **E** predicates may in a sense be seen as asking information, while **H** atoms may be considered as new information provided during search. However, as we have shown in the paper, **E** atoms may also mean *expected* behavior, and the SCIFF can cope with abducibles containing universally quantified variables.

The concept of hypotheses confirmation has been studied also by Kakas and Evans [22], where hypotheses can be corroborated or refuted by matching them with observable atoms: an explanation fails to be corroborated if some of its

logical consequences are not observed. The authors suggest that their framework could be extended to take into account dynamic events, eventually, queried to the user: *"this form of reasoning might benefit for the use of a query-the-user facility"*.

In a sense, our work can be considered as an extension of these works: it provides the concept of confirmation of hypotheses, as in corroboration, and it provides an operational semantics for dynamically incoming events. Moreover, we extend the work by imposing integrity constraints to better define the feasible combinations of hypotheses, and we let the program abduce non-ground atoms.

The dinamicity of incoming events can be considered as an instance of an Evolving Logic Program [23]. In EvoLP, the knowledge base can be dynamically modified depending both on events that come from the external world and on the result of a previous computation. The *S*CIFF proof procedure does not generate new events, but only expectations about external events. The focus of the work is more on the expressivity of the generated expectations (which can contain variables universally quantified and constrained) than on the generality of the evolution of the knowledge base. An interesting extension of our work would contain evolution of the whole knowledge bases, not only of the set of happened events.

In Speculative Computation [24, 25] hypotheses are abduced and can be confirmed later on. It is a framework for a multi-agent setting with unreliable communication. When an agent asks a query to another agent, it also abduces its (default) answer; if the real answer arrives within a deadline, the hypothesis is confirmed or disconfirmed; otherwise the computation continues with the default. In our work, expectations can be confirmed by events, but the scope is different. In our work, if a deadline is missed the computation fails, as an hypothesis has been disconfirmed.

Other implementations have been given of abductive proof procedures in Constraint Handling Rules [26, 27]. Our implementation is more adherent to the theoretical operational semantics (in fact, every transition is mapped onto CHR rules) and exploits the uniform understanding of constraints and abducibles noted by Kowalski et al. [17].

Finally, in Section 5 we considered multi-agent systems to show an application of the *S*CIFF. Some discussion about other formal approaches to protocol verification can be found in [13].

7 Conclusions

In this paper, we proposed an abductive logic programming framework which extends most previous work in several directions. The two main features of this framework are: the possibility to account for new dynamically upcoming facts, and the possibility to have hypotheses confirmed/disconfirmed by following observations and evidence. We proposed a language, and described its declarative and operational semantics. We implemented the proof-procedure for a system verifying the compliance of agents to protocols; the implementation can be downloaded from `http://lia.deis.unibo.it/Research/sciff/` [8].

Acknowledgements

This work has been supported by the European Commission within the SOCS project (IST-2001-32530), funded within the Global Computing programme and by the MIUR COFIN 2003 projects *La Gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici* and *Sviluppo e verifica di sistemi multiagente basati sulla logica.*

References

1. Poole, D.L.: A logical framework for default reasoning. Artificial Intelligence **36** (1988) 27–47
2. Kakas, A.C., Mancarella, P.: On the relation between Truth Maintenance and Abduction. In Fukumura, T., ed.: Proceedings of PRICAI-90, Nagoya, Japan, Ohmsha Ltd. (1990) 438–443
3. Console, L., Dupré, D.T., Torasso, P.: On the relationship between abduction and deduction. Journal of Logic and Computation **1** (1991) 661–690
4. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33** (1997) 151–165
5. Kakas, A.C., Michael, A., Mourlas, C.: ACLP: Abductive Constraint Logic Programming. Journal of Logic Programming **44** (2000) 129–177
6. Kakas, A.C., van Nuffelen, B., Denecker, M.: A-System: Problem solving through abduction. In Nebel, B., ed.: Proceedings of the 17th IJCAI, Seattle, Washington, USA, Morgan Kaufmann Publishers (2001) 591–596
7. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19-20** (1994) 503–582
8. : (The SCIFF abductive proof procedure) `http://lia.deis.unibo.it/Research/sciff/`.
9. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming **37** (1998) 95–138
10. Alberti, M., Chesani, F.: The implementation of a system for generation and confirmation of hypotheses. Technical Report CS-2004-2, Dipartimento di Ingegneria, Università degli Studi di Ferrara, Ferrara, Italy (2004) Available at `http://www.ing.unife.it/informatica/tr/CS-2004-02.pdf`.
11. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Abduction with hypotheses confirmation. Technical Report CS-2004-01, Department of Engineering, University of Ferrara, Ferrara, Italy (2004) Available at `/http://www.ing.unife.it/informatica/tr/CS-2004-01.pdf`.
12. Kakas, A.C., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. In Gabbay, D.M., Hogger, C.J., Robinson, J.A., eds.: Handbook of Logic in Artificial Intelligence and Logic Programming. Volume 5., Oxford University Press (1998) 235–324
13. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of interaction protocols: a computational logic approach based on abduction. Technical Report CS-2003-03, DI, University of Ferrara, Italy (2003) `http://www.ing.unife.it/informatica/tr/cs-2003-03.pdf`.
14. : (Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees) `http://lia.deis.unibo.it/Research/SOCS/`.

15. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Modeling interactions using *Social Integrity Constraints*: a resource sharing case study. (In Leite, J.A., Omicini, A., Sterling, L., Torroni, P., eds.: DALT 2003. Melbourne, Victoria. Workshop Notes) 81–96
16. Alberti, M., Daolio, D., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interaction protocols in a logic-based system. In: Proceedings of the 19th ACM SAC. (AIMS Track), Nicosa, Cyprus, ACM Press (2004) to appear.
17. Kowalski, R., Toni, F., Wetzel, G.: Executing suspended logic programs. Fundamenta Informaticae **34** (1998) 203–224 `http://www-lp.doc.ic.ac.uk/UserPages/staff/ft/PAPERS/slp.ps.Z`.
18. Endriss, U., Mancarella, P., Sadri, F., Terreni, G., Toni, F.: Abductive Logic Programming with CIFF. In Bennett, B., ed.: Proceedings of the 11th Workshop on Automated Reasoning, Bridging the Gap between Theory and Practice, University of Leeds (2004) Extended Abstract. To appear.
19. Alferes, J., Pereira, L.M., Swift, T.: Abduction in well-founded semantics and generalized stable models via tabled dual programs. Theory and Practice of Logic Programming (2003)
20. Chen, W., Warren, D.S.: Tabled evaluation with delaying for general logic programs. Journal of the ACM **43** (1996) 20–74
21. Sergot, M.J.: A query-the-user facility of logic programming. In Degano, P., Sandwell, E., eds.: Integrated Interactive Computer Systems, North Holland (1983) 27–41
22. Evans, C., Kakas, A.: Hypotheticodeductive reasoning. In: Proc. International Conference on Fifth Generation Computer Systems, Tokyo (1992) 546–554
23. Alferes, J.J., Brogi, A., Leite, J.A., Pereira, L.M.: Evolving logic programs. In Flesca, S., Greco, S., Leone, N., Ianni, G., eds.: Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02). Volume 2424 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2002) 50–61
24. Satoh, K., Inoue, K., Iwanuma, K., Sakama, C.: Speculative computation by abduction under incomplete communication environments. In: Proceedings of the 4th International Conference on Multi-Agent Systems, Boston, USA, IEEE Press (2000) 263–270
25. Satoh, K., Yamamoto, K.: Speculative computation with multi-agent belief revision. In Castelfranchi, C., Lewis Johnson, W., eds.: Proceedings of AAMAS-2002, Part II, Bologna, Italy, ACM Press (2002) 897–904
26. Abdennadher, S., Christiansen, H.: An experimental CLP platform for integrity constraints and abduction. In Larsen, H., Kacprzyk, J., Zadrozny, S., Andreasen, T., Christiansen, H., eds.: FQAS, Flexible Query Answering Systems. LNCS, Warsaw, Poland, Springer-Verlag (2000) 141–152
27. Gavanelli, M., Lamma, E., Mello, P., Milano, M., Torroni, P.: Interpreting abduction in CLP. In Buccafurri, F., ed.: AGP, Reggio Calabria, Italy (2003) 25–35