Interleaving belief revision and reasoning: preliminary report

F. Sadri and F. Toni

Department of Computing, Imperial College London, UK
Email: {fs,ft}@doc.ic.ac.uk

Abstract. Most existing work on belief revision and knowledge representation and reasoning tacitly assumes that belief revision is performed off-line, and that reasoning from the beliefs is performed either before or after the beliefs are changed. This imposes that, if a revision occurs while reasoning is performed, reasoning has to be stopped and re-started anew so that the revision is taken into account, with an obvious wastage of reasoning effort. In this paper, we tackle the problem of performing belief revision on-line, while reasoning is taking place by means of an abductive proof procedure.

1 Introduction

Traditionally, the field of belief revision (e.g. [1]) has concentrated upon identifying principled ways to incorporate new information or delete existing information from sets of beliefs (or belief bases). On the other hand, the field of knowledge representation and reasoning has concentrated upon identifying techniques for drawing correct conclusions efficiently from beliefs. Beliefs could be held by intelligent agents, and reasoning could aim at identifying plans for the agents' goals or answering queries on the agents' beliefs.

If the revision of beliefs takes place while reasoning is performed, the problem arises as to which bits of the already performed reasoning can be "saved", because it has used up beliefs which have not been affected by the revision. Conventional knowledge representation and reasoning systems overlook this problem, and assume that no bits of reasoning can be saved, by restarting the reasoning process anew.

In this paper, we show how belief revision and reasoning can be interleaved, so that reasoning with beliefs that are not affected by the revision can be saved and, if necessary, used after the revision. We assume that reasoning is performed by means of the abductive proof procedure IFF [4] and that beliefs are represented by means of abductive logic programs [7]. In this paper, we also assume that belief revision amounts to the addition or deletion of rules or facts to the logic programming component of the abductive logic program, or to the addition of integrity constraints in the corresponding component of the abductive logic program. For simplicitly, we do not consider deletion of integrity constraints.

These rules and facts could be generated by learning, or by any other means. In this paper, we do not focus on the belief revision part of the process, but rather on how such revision affects the reasoning process. In order to save any reasoning effort not affected by revisions, we adopt a labeling technique that maintains dependencies. We define a new variant of the IFF procedure, called *LIFF*, with labels occurring in derivations. We formalise the soundness of LIFF, wrt the semantics underlying the IFF procedure.

In this paper, we concentrate on the propositional case only for the definition of LIFF and the results.

Our work is closely related to the work of [5, 6], which share our aims. However, we extend an abductive proof procedure (IFF) for beliefs in the form of abductive logic programs (consisting of logic programs and integrity constraints) whereas [5] extends conventional SLDNF, for ordinary logic programs. Our knowledge representation and reasoning choice paves the way to the use of our techniques for agent applications, following the abductive logic agent approach of [10, 14, 8]. Moreover, we use a labeling technique rather than the (arguably more space consuming) technique based upon full data structures associated to atoms as [5].

Amongst agent applications for which our approach is particularly useful are those where the agents are situated in dynamic environments and need to adjust quickly to the changes in that environment. For example, a situated pro-active agent can plan for its goals while it interleaves the planning with action execution and observations from the environment; such an agent has, for example been described in [12]. The observations that it makes and records from the environment and the success or failure of the actions that it attempts to execute can lead to the agent revising its beliefs. LIFF allows the agent to keep as much of its partial plan as is possible and to replan only those parts that are affected by the changes in its beliefs.

Another agent application that would benefit for our work is information integration. In this application a mediator agent receives the user query specified in some user-oriented high-level language. It then constructs a query plan translating the user query into a form understandable by the sources of information it has at its disposal. It contacts the sources for the necessary information, integrates their answers, and translates the answer into the language of the user. Such an information integration approach based on abductive logic programming and, in particular the IFF proof procedure, has been described in [21, 16]. Now, while the agent is constructing the query plan for the user query, it may receive information about semantic content or access availability changes to the information sources. Here again LIFF would allow the agent to accommodate these changes within its reasoning without having to discard all the work it has done in constructing the (possibly partial) query plan.

The paper is organised as follows. In section 2 we summarise necessary background on abductive logic programming and the abductive proof procedure IFF, in the propositional case. In section 3 we define the effect of (propositional) updates upon given abductive logic programs. In section 4 we give a simple example of the working of our procedure LIFF, interleaving belief revision and reasoning. In section 5 we define LIFF, in the propositional case. In section 6 we give our results. In section 7 we conclude.

2 Preliminaries

Abductive logic programming is a general-purpose knowledge representation and reasoning framework that can be used for a number of applications and tasks [7, 10, 3, 13, 15, 14, 19, 20]. It relies upon a symbolic, logic-based representation of beliefs, for any given domain of application, via *abductive logic programs*, and the execution of logic-based reasoning engines, called *abductive proof procedures*, for reasoning with such representations. In the propositional case, an abductive logic program consists of

– A **logic program**, P, namely a set of if-rules of the form $Head \leftarrow Body$, where $Head$ is an atom and $Body$ is either $true$ (in which case we write the if-rule simply as $Head$) or a conjunction of (negations of) atoms. P is understood as a (possibly incomplete) set of beliefs.
– A set of **abducible atoms**, $\mathcal{A}$. are assumed not to occur in the $Head$ of any if-rule in P. Abducible atoms can be used to "complete" the (beliefs held within the) logic program, subject to the satisfaction of the integrity constraints (see below).
– A set of **integrity constraints**, I, namely if-then-rules of the form $Condition \Rightarrow Conclusion$, where $Condition$ is either $true$ (in which case we write the if-then-rule simply as $Conclusion$) or a conjunction of (negations of) atoms, and $Conclusion$ is $false$ or an atom. The integrity constraints are understood as properties that must be "satisfied" by any "acceptable" extension of the logic program by means of atoms of abducibles, in the same way integrity constraints in databases are understood as properties that must be "satisfied" in every database state. We will assume that all integrity constraints in I are satisfiable, namely there exists some extension of P via sets of abducibles that satisfies them. [1]

In the sequel, $\overline{\mathcal{A}}$ will refer to the complement of the set $\mathcal{A}$ wrt the set of all predicates in the vocabulary of P, i.e. $\overline{\mathcal{A}}$ is the set of non-abducible predicates.

Both IFF and or new proof procedure LIFF rely upon the **completion** [2] of logic programs. The completion of a predicate p defined, within a propositional logic program, by the if-rules

$p \leftarrow D_1, \ldots, p \leftarrow D_k$

is the **iff-definition**

$p \leftrightarrow D_1 \vee \ldots \vee D_k$

The left and right-hand side are called the *head* and the *body*, respectively. The completion of a predicate p not defined in a given logic program is

[1] Otherwise, no explanation may possibly exist for any query, see below.

$p \leftrightarrow false.$

The **selective completion** $comp_S(P)$ of a logic program P wrt a set of predicates S in the vocabulary of P is the union of the completions of all the predicates in S. Both IFF and LIFF use $comp_{\overline{\mathcal{A}}}(P)$.

Abductive proof procedures aim at computing "explanations" (or "abductive answers") for "queries", given an abductive logic program representing knowledge about some underlying domain. A **query** is a (possibly empty) conjunction of literals. Given an abductive logic program $\langle P, \mathcal{A}, I\rangle$, in the propositional case, an **explanation** (or **abductive answer**) for a query Q is a (possibly empty) set E of ground abducibles such that $P \cup E$ *entails* Q and $P \cup E$ *satisfies* I. Various notions of entailment and satisfaction can be adopted, for example entailment and satisfaction could be entailment and consistency, respectively, in first-order, classical logic. In this paper, we propose an adaptation of IFF as the chosen abductive proof procedure, that we refer to as LIFF. We thus adopt truth wrt the 3-valued completion semantics of [11] as the underlying notion of entailment and satisfaction, inherited from [4].

IFF generates a **derivation** consisting of a **sequence of goals**, starting from an **initial goal**, which is the given query conjoined with the integrity constraints in I. Moreover, IFF computes explanations (referred to as **extracted answers**) for the given query extracting them from disjuncts in a goal in a derivation from the initial goal, such that no further inference rule can be applied to these disjuncts.

Goals in a derivation are obtained by applying **inference rules**. In the simplest case, goals derived by the IFF proof procedure are disjunctions of **simple goals**, which are conjunctions of the form

$A_1 \wedge \ldots \wedge A_n \wedge I_1 \wedge \ldots \wedge I_m \wedge D_1 \wedge \ldots \wedge D_k$

where $n, m, k \geq 0$, $n+m+k > 0$, the A_i are atoms, the I_i are **implications**, with the same syntax of integrity constraints, except that in addition to universally quantified variables, they can also contain existentially quantified or free variables, occurring elsewhere in the goal, and the D_i are disjunctions of conjunctions of literals and implications. Implications are obtained by repeatedly applying the inference rules of the proof procedure to either integrity constraints in the given $\langle P, \mathcal{A}, I\rangle$ or to the result of rewriting negative literals $not\, A$ as $A \Rightarrow false$.

IFF assumes that the inference rules are applied in such a way that every goal in a sequence derived by the proof procedure is a *disjunction of simple goals*. (Note that every initial goal is a simple goal with the integrity constraints and the rewriting of negative literals as the only implications.) LIFF will make the same assumption.

We omit here the definition of the inference rules of IFF, as they can be reconstructed from the inference rules of LIFF, given in section 5, by dropping the labels. In the next section, we illustrate the behaviour of the IFF procedure and of LIFF with a simple example. The example illustrates the main difference between IFF and LIFF, namely the fact that the latter associates labels to literals and implications in goals, to be exploited when predicate definitions are updated.

3 Assimilating updates in the abductive logic program

In this paper we do not address the issue of belief revision. As mentioned in the section 1, the belief revision can be done in a number of different ways, for example through learning, or through a process of assimilation similar to the one described in [9]. However the belief revision is handled, at the end of the process, items (facts, rules, integrity constraints) will need to be added or deleted from the abductive logic program.

In this section we define how a given abductive logic program $\langle P, \mathcal{A}, I\rangle$ is revised after (any number of) the following kinds of updates:

- the addition to P of facts (atoms) or if-rules,
- the deletion from P of facts or if-rules,
- the addition to I of if-then-rules.

Note that we ignore the deletion of if-then-rules from I, in order to keep the definition of LIFF as simple as possible. This deletion could be easily accommodated with the aid of appropriate additional labels in LIFF. We leave this as a future extension of our work. Note also that we do not directly allow for *modifying* if-rules and integrity constraints, as this can be achieved by suitable combinations of deletions and additions.

We will implicitly assume that the integrity constraints I' in the revised abductive logic program $\langle P', \mathcal{A}', I'\rangle$, after the updates have taken place, are satisfiable by the revised logic program P', as earlier in section 2.

There are a number of different ways that we can accommodate addition and deletion of facts and rules in an ALP. For example, we can decide whether or not a predicate that is originally abducible will, in effect, remain abducible after any updates. Similarly we can decide whether or not a predicate that is originally non-abducible will always be non-abducible, even if all its definitions are deleted from the logic program. LIFF is independent of these choices, and it can deal uniformly with any combination of these choices. Below, to set the scene, we, arbitrarily, make the concrete choice where abducible predicates always remain abducible and non-abducible predicates always remain non-abducible, no matter how many updates they are subjected to.

3.1 Addition of facts or if-rules

Let the update be $p \leftarrow B$, with $B \neq false$, [2] and with B possibly $true$ (in which case p is a ground fact). Below, we refer to $p \leftarrow B$ simply as U.

Let $\langle P, \mathcal{A}, I\rangle$ be the abductive logic program before the update. We will refer to $\mathcal{A}_B$ as the set of all predicates occurring in B in the update but not already in the vocabulary of $\langle P, \mathcal{A}, I\rangle$.

We distinguish three cases:

[2] The addition of $p \leftarrow false$ is not allowed directly, but it can be achieved by deleting all if-rules with head p.

1. p is an abducible in $\mathcal{A}$; then revising $\langle P, \mathcal{A}, I\rangle$ by U gives $U(\langle P, \mathcal{A}, I\rangle)$=$\langle P', \mathcal{A}', I\rangle$, obtained as follows:
   - $P' = P \cup \{U\} \cup \{p \leftarrow p^*\}$,
   - $\mathcal{A}' = ((\mathcal{A} \cup \{p^*\}) - \{p\}) \cup \mathcal{A}_B$,

   where p^* is a new predicate not occurring in the vocabulary of $\langle P, \mathcal{A}, I\rangle$;
2. p is in the vocabulary of $\langle P, \mathcal{A}, I\rangle$ but is not abducible; then revising $\langle P, \mathcal{A}, I\rangle$ by U gives $U(\langle P, \mathcal{A}, I\rangle)$=$\langle P', \mathcal{A}', I\rangle$, obtained as follows:
   - $P' = P \cup \{U\}$,
   - $\mathcal{A}' = \mathcal{A} \cup \mathcal{A}_B$;
3. p is not in the vocabulary of $\langle P, \mathcal{A}, I\rangle$; then revising $\langle P, \mathcal{A}, I\rangle$ by U gives $U(\langle P, \mathcal{A}, I\rangle)$=$\langle P', \mathcal{A}', I\rangle$, obtained as follows:
   - $P' = P \cup \{U\}$,
   - $\mathcal{A}' = \mathcal{A} \cup \mathcal{A}_B$.

Note that we assume that any predicate occurring in the body of any added rule but not already present in the vocabulary of the abductive logic program prior to the update is treated as a new abducible (in $\mathcal{A}_B$). This choice sees new undefined predicates as "open", and thus abducible.

Note that we do not allow the addition of new abducibles explicitly, from the outside. However, new abducibles are added as a by-product of adding definitions for abducibles (case 1 above) or definitions of predicates containing the new abducibles in the body, as predicates that did not occur before in the abductive logic program (cases 1-3 above).

3.2 Deletion of facts or if-rules

Let $\langle P, \mathcal{A}, I\rangle$ be the abductive logic program before the update. Let the update be the deletion of $p \leftarrow B \in P$ (B possibly *true*, in which case p is a ground fact), referred to simply as U. We will assume that B is different from a p^* atom, namely an atom introduced in case 1 in section 3.1. This means that predicates that are abducible in the initial abductive logic program always remain "abducible", in the sense that they can be assumed to hold by abduction (of atoms in p^*).

Revising $\langle P, \mathcal{A}, I\rangle$ by U gives $U(\langle P, \mathcal{A}, I\rangle)$=$\langle P', \mathcal{A}', I\rangle$, obtained as follows:

- $P' = P - \{U\}$;
- $\mathcal{A}' = \mathcal{A} - \{a | a$ is a predicate occurring in B and nowhere else in P or $I\}$.

Note that it is arguable whether or not it is sensible whether to delete all integrity constraints that only mention the deleted abducibles. We assume here that such integrity constraints are not deleted.

3.3 Addition of integrity constraints

Let $\langle P, \mathcal{A}, I\rangle$ be the abductive logic program before the update. Let the update be the addition of $C \Rightarrow D$, referred to simply as U. Let $\mathcal{A}_U$ be the set of all predicates occurring in C, D in the update but not already in the vocabulary of $\langle P, \mathcal{A}, I\rangle$.

Revising $\langle P, \mathcal{A}, I\rangle$ by U gives $U(\langle P, \mathcal{A}, I\rangle)$=$\langle P, \mathcal{A}', I'\rangle$, obtained as follows:

- $I' = I \cup \{U\}$;
- $\mathcal{A}' = \mathcal{A} \cup \mathcal{A}_U$.

3.4 Multiple updates

Here we define the effect of multiple updates upon an initially given abductive logic program $\langle P, \mathcal{A}, I\rangle$. Let the sequence of updates be $U_1, \ldots, U_n$, with $n > 1$. Then, $U_1, \ldots, U_n$ applied to $\langle P, \mathcal{A}, I\rangle$ gives

$$U_n \circ U_{n-1} \circ \ldots \circ U_1(\langle P, \mathcal{A}, I\rangle) = U_n(U_{n-1}(\ldots(U_1(\langle P, \mathcal{A}, I\rangle)))).$$

Later on, in section 5.2, we will allow empty updates to occur in sequences of updates. In this setting, an empty update stands for no update at all. However, empty updates are useful in the formal definition of LIFF derivations, to accommodate updates at the correct place during derivations.

4 An illustrative example

In this section we illustrate the application of LIFF and its relationship to IFF via a concrete example.

Given the abductive logic program $\langle P, \mathcal{A}, I\rangle$ with

$$\begin{aligned}
P\colon\ & p \leftarrow q\\
& p \leftarrow a\\
& q \leftarrow d \wedge c\\
& n \leftarrow e\\
\mathcal{A}\colon\ & a, b, c, d, e\\
I\colon\ & a \Rightarrow b\\
& c \wedge n \Rightarrow false
\end{aligned}$$

and a query p, IFF would derive the following sequence of goals:

$p \wedge I$

by unfolding p with $Comp_{\overline{\mathcal{A}}}(P)$:

$(q \vee a) \wedge I$

by splitting (distributing $\vee$ over $\wedge$):

$[q \wedge I] \vee [a \wedge I]$

by unfolding q with $Comp_{\overline{\mathcal{A}}}(P)$:

$[d \wedge c \wedge I] \vee [a \wedge I]$
by propagation (with $a \Rightarrow b \in I$ in the second disjunct):
$[d \wedge c \wedge I] \vee [a \wedge b \wedge I]$
by propagation (with $c \wedge n \Rightarrow \mathit{false} \in I$ in the first disjunct):
$[d \wedge c \wedge [n \Rightarrow \mathit{false}] \wedge I] \vee [a \wedge b \wedge I]$
by unfolding n with $Comp_{\overline{\mathcal{A}}}(P)$:
$[d \wedge c \wedge [e \Rightarrow \mathit{false}] \wedge I] \vee [a \wedge b \wedge I]$

From the final goal in this sequence of goals, IFF would extract the explanations $\{d, c\}$ (from the first disjunct/simple goal) and $\{a, b\}$ (from the second disjunct/simple goal) for the given query p. From the goal before last in this sequence, IFF would extract the explanation $\{a, b\}$ (from the second simple goal). No explanation could be extracted from the first simple goal in that goal, as inference rules can still be applied to it.

Given the same $\langle P, \mathcal{A}, I\rangle$ and query, LIFF would derive the following sequence of goals with labels, with updates as indicated. Note that a label $\{\langle \mathtt{X_1}; \mathtt{Y_1}\rangle, \ldots \langle \mathtt{X_n}; \mathtt{Y_n}\rangle\}$ associated with an atom or an implication Z in a goal, with $\mathtt{X_i}$ a predicate, intuitively indicates that "if the definition of $\mathtt{X_i}$ is updated (by adding or deleting if-rules for it), then Z should be replaced by $\mathtt{Y_i}$". Also, $\emptyset$ stands for the empty set of sentences.

$p \wedge I$
by unfolding and splitting:
$[q : \{\langle \mathtt{p}; \mathtt{p}\rangle\} \wedge I] \vee [a : \{\langle \mathtt{p}; \mathtt{p}\rangle\} \wedge I]$
by unfolding:
$[d : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge c : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge I] \vee [a : \{\langle \mathtt{p}; \mathtt{p}\rangle\} \wedge I]$
by propagation:
$[d : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge c : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge I] \vee [a : \{\langle \mathtt{p}; \mathtt{p}\rangle\} \wedge b : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{a}; \emptyset\rangle\} \wedge I]$
by propagation:
$[d : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge c : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge [n \Rightarrow \mathit{false}] : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{c}; \emptyset\rangle, \langle \mathtt{q}; \emptyset\rangle\} \wedge I] \vee$
$[a : \{\langle \mathtt{p}; \mathtt{p}\rangle\} \wedge b : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{a}; \emptyset\rangle\} \wedge I]$
by unfolding:
$[d : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge c : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge$
$[e \Rightarrow \mathit{false}] : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{q}; \emptyset\rangle, \langle \mathtt{c}; \emptyset\rangle, \langle \mathtt{n}; \mathtt{n} \Rightarrow \mathtt{false} : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{q}; \emptyset\rangle, \langle \mathtt{c}; \emptyset\rangle\}\rangle\} \wedge I] \vee$
$[a : \{\langle \mathtt{p}; \mathtt{p}\rangle\} \wedge b : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{a}; \emptyset\rangle\} \wedge I]$

Note that the same answers could be extracted from the two simple goals here as in the case of IFF. However, if, before answer extraction takes place, the definition of p is updated, then LIFF would generate, as the next goal in the sequence,

$$[p \wedge p \wedge I] \vee [p \wedge I]$$

which collapses to the initial goal $p \wedge I$. So, in this case, LIFF would start the reasoning process from scratch, as would happen if applying IFF (or any other conventional abductive proof procedure).

Suppose n is updated instead. Then, our procedure would obtain

$$[d : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge c : \{\langle \mathtt{p}; \mathtt{p}\rangle, \langle \mathtt{q}; \mathtt{q}\rangle\} \wedge [n \Rightarrow false] : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{q}; \emptyset\rangle, \langle \mathtt{c}; \emptyset\rangle\} \wedge I] \vee$$
$$[a : \{\langle \mathtt{p}; \mathtt{p}\rangle\} \wedge b : \{\langle \mathtt{p}; \emptyset\rangle, \langle \mathtt{a}; \emptyset\rangle\} \wedge I]$$

thus avoiding starting the reasoning process from scratch.

5 The LIFF proof procedure

We first define the notion of label.

Definition 1. *A* **label** *(for an atom or implication) is a set*

$$\{\langle \mathtt{X_1}; \mathtt{Y_1}\rangle, \ldots, \langle \mathtt{X_n}; \mathtt{Y_n}\rangle\}$$

$n \geq 0$*, with each* $\mathtt{X_i}$ *a predicate and each* $\mathtt{Y_i}$ *either*

- *the empty sentence* $\emptyset$*, or*
- *an atom (possibly with a label), or*
- *an implication (possibly with a label).*

Note that labels can be empty (if $n = 0$). Below (and in the earlier example in section 4), absence of a label corresponds to an empty label. Intuitively, each $\langle \mathtt{X_i}; \mathtt{Y_i}\rangle$ in a label for Z indicates what Z should become if any update is made upon $\mathtt{X_i}$, namely if a new definition is added to P for $\mathtt{X_i}$ or if an existing definition for $\mathtt{X_i}$ is deleted from P.

We will refer to labelled atoms/implications simply as atoms/implications. Moreover, the terminology of goals and simple goals will carry through in the presence of labels.

5.1 LIFF inference rules

Given an abductive logic program $\langle P, \mathcal{A}, I\rangle$ and an initial query Q, we will define *LIFF derivations* for Q in terms of sequences of goals, $G_1, \ldots, G_n$, such that $G_1 = G \wedge I$ [3] and each G_{i+1} is obtained from the previous goal G_i either by enforcing an update (see section 5.2) or by application of one of the inference rules below. The intuition behind each of the labels is to identify the components that have contributed to the derivation of the new element(s).

Unfolding: given an atom $p : \mathtt{l}$ and $p \leftrightarrow D_1 \vee \ldots \vee D_n$ in $comp_{\overline{\mathcal{A}}}(P)$

- if the atom is a conjunct of a simple goal in G_i, then G_{i+1} is G_i with the atom replaced by $(D_1 \vee \ldots \vee D_n) : \mathtt{l} \cup \{\langle \mathtt{p}; \mathtt{p} : \mathtt{l}\rangle\}$;

[3] In the sequel, $(A_1 \wedge \ldots \wedge A_k) : \mathtt{l}$ $(k > 1)$ stands for $A_1 : \mathtt{l} \wedge \ldots \wedge A_k : \mathtt{l}$ and $(D_1 \vee \ldots \vee D_n) : \mathtt{l}$ stands for $D_1 : \mathtt{l} \vee \ldots \vee D_n : \mathtt{l}$, where $\mathtt{l}$ is a label.

- if p is a conjunct L_i in the body of an implication
  $[L_1 \wedge \ldots \wedge L_m \Rightarrow A] : \mathtt{l}'$, $m \geq 1$
  which is a conjunct of a simple goal of G_i, then G_{i+1} is G_i with the implication replaced by the conjunction
  $[L_1 \wedge \ldots \wedge D_1 \wedge \ldots \wedge L_m \Rightarrow A] : \mathtt{l}'' \wedge \ldots \wedge$
  $[L_1 \wedge \ldots \wedge D_n \wedge \ldots \wedge L_m \Rightarrow A] : \mathtt{l}''$,
  where $\mathtt{l}'' = \mathtt{l}' \cup \{\langle \mathtt{p}; \mathtt{L_1} \wedge \ldots \wedge \mathtt{L_m} \Rightarrow \mathtt{A} : \mathtt{l}' \rangle\}$

Propagation: given an atom $p : \mathtt{l}$ and an implication
$[L_1 \wedge \ldots \wedge L_m \Rightarrow A] : \mathtt{l}'$, $m \geq 1$
with $L_i = p$, for some $1 \leq i \leq m$, both conjuncts of the same simple goal in G_i, then, if the implication
$[L_1 \wedge \ldots L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_m \Rightarrow A] : \{\langle \mathtt{p}; \emptyset \rangle\} \cup \{\langle \mathtt{q}; \emptyset \rangle | \langle \mathtt{q}; \mathtt{r} \rangle \in \mathtt{l} \cup \mathtt{l}'\}$
is not already a conjunct of the simple goal, then G_{i+1} is G_i with the new implication conjoined to the simple goal.

Negation elimination: given an implication $\mathit{not}\, A_1 \wedge \ldots \wedge \mathit{not}\, A_m \Rightarrow A : \mathtt{l}$ which is a conjunct of a simple goal in G_i, then G_{i+1} is G_i with the implication replaced by the disjunction $[A \vee A_1 \vee \ldots \vee A_m] : \mathtt{l}$. [4]

Logical simplification replaces:
- $[B : \mathtt{l} \vee C : \mathtt{l}'] \wedge E : \mathtt{l}''$ by
  $[B : \mathtt{l} \wedge E : \mathtt{l}''] \vee [C : \mathtt{l}' \wedge E : \mathtt{l}'']$ (**splitting**);
- $B : \mathtt{l} \wedge B : \mathtt{l}$ by
  $B : \mathtt{l}$ where B is an atom or an implication;
- $B : \mathtt{l} \vee B : \mathtt{l}$ by
  $B : \mathtt{l}$ where B is an atom or an implication.

Note that we are not including some simplification steps present in IFF, e.g. $A \wedge \mathit{false} \equiv \mathit{false}$. Such simplification steps only affect the "efficiency" of IFF, rather than its correctness, in that they prevent "working" on a disjunct in a goal from which no answer can ever be extracted (e.g. for $A \wedge \mathit{false} \equiv \mathit{false}$, due to the inconsistency of that disjunct). In the case of LIFF, though, these inference steps could undermine correctness, as they would prevent us from keeping track of intermediate steps that might be affected by updates (e.g. $A \wedge \mathit{false}$, if what lead to false is updated).

Note also that we are not allowing "merging" of identical conjuncts but with different labels, as this would prevent us from incorporating all updates correctly. This is illustrated by the following example. Assume $\langle P, \mathcal{A}, I \rangle$ with

P: $p \leftarrow a$
$\mathcal{A}$: $\{a\}$
I: $a \Rightarrow a$

[4] In [4], negation elimination is defined as follows: $\mathit{not}\, A_1 \wedge \mathit{Rest} \Rightarrow A$ is replaced by $\mathit{Rest} \Rightarrow A \vee A_1$. Operationally, our definition (ignoring the labels) is equivalent to the one in [4].

then, given the query p, the following sequence of goals is generated by IFF:

$p \wedge I$
  by unfolding p with $Comp_{\overline{\mathcal{A}}}(P)$:
$a \wedge I$
  by propagation:
$a \wedge I \wedge a$
  by simplification
$a \wedge I$

from which the answer $\{a\}$ can be extracted. The following is the LIFF counterpart of this derivation:

$p \wedge I$
  by unfolding p with $Comp_{\overline{\mathcal{A}}}(P)$:
$a : \{\langle \mathtt{p}; \mathtt{p} \rangle\} \wedge I$
  by propagation:
$a : \{\langle \mathtt{p}; \mathtt{p} \rangle\} \wedge I \wedge a : \{\langle \mathtt{p}; \emptyset \rangle, \langle \mathtt{a}; \emptyset \rangle\}$

from which the answer $\{a\}$ can be extracted, as we will see in section 5.3. If we allowed the merging of the two occurrences of a and their labels to obtain:

$a : \{\langle \mathtt{p}; \mathtt{p} \rangle, \langle \mathtt{a}; \emptyset \rangle\} \wedge I$

the resulting label would give an incorrect indication as to how the goal should be revised if the definition of a weer revised.

5.2 LIFF derivations

In LIFF derivations, the application of inference rules (as given in section 5.1) is interleaved with the assimilation of updates within the abductive logic program (as given in section 3) and the application to these updates to goals, defined as follows:

Definition 2. *Given a goal G and an update U, the* **updated goal wrt G and U** *is a goal G' obtained as follows:*

- *if U is the addition of a fact or rule $p \leftarrow B$, then*
  - *if p is not abducible, then G' is G where every conjunct with label containing $\langle p; Q \rangle$ is replaced by Q;*
  - *if p is abducible, then G' is identical to G.*
- *if U is the deletion of a fact or rule $p \leftarrow B$, then G' is G where every conjunct with label containing $\langle p; Q \rangle$ is replaced by Q;*
- *if U is the addition of an integrity constraint X then G' is G where X is conjoined to each simple goal.*

Definition 3. *Given a query Q, an abductive logic program $\langle P, \mathcal{A}, I\rangle$, and a sequence of updates $U_1, \ldots, U_n$, $n > 0$, a* **LIFF derivation** *is a sequence of tuples:*

$$(G_1, U_1, R_1, ALP_1), \ldots, (G_n, U_n, R_n, ALP_n)$$

where, for each $1 \leq i \leq n$,

- *G_i is a goal,*
- *U_i is a (possibly empty) update,*
- *R_i is (the application of) an inference rule or is empty,*
- *exactly one of U_i and R_i is non-empty,*
- *ALP_i is an abductive logic program,*

and $G_1 = Q \wedge I$, ALP_1=$\langle P, \mathcal{A}, I\rangle$, and, for each $1 < i \leq n$,

- *if U_{i-1} is empty then*
  - *G_i is obtained from G_{i-1} by applying R_{i-1}*
  - *ALP_i=ALP_{i-1}*
- *if U_{i-1} is non-empty then*
  - *G_i is the updated goal wrt G_{i-1} and U_{i-1}*
  - *ALP_i=$U_{i-1}(ALP_{i-1})$.*

The **end goal** *and* **end abductive logic program** *in a LIFF derivation are G and ALP, respectively, such that:*

- *if U_n is empty then*
  - *G is obtained from G_n by applying R_n*
  - *ALP=ALP_n*
- *if U_n is non-empty then*
  - *G is the updated goal wrt G_n and U_n*
  - *ALP=$U_n(ALP_n)$.*

Note that for goals in LIFF derivations to be guaranteed to be disjunctions of simple goals, it is sufficient that every step of unfolding and negation elimination is followed by a step of splitting. We will assume this is the case in all derivations.

We will refer to a LIFF derivation whose updates are all empty as a **static LIFF derivation**.

5.3 Successful LIFF derivations and extracted answers

In this section we formalise the notion of answer extraction for LIFF, by adapting the notion of answer extraction for IFF to take into account labels.

Given a LIFF derivation for a query Q, abductive logic program $\langle P, \mathcal{A}, I\rangle$ and a given sequence of updates, let G be the end goal of the derivation. Let D be a disjunct of G:

- if no inference rule can be applied to D, then this is called **conclusive**;
- if $false$: `l`, for any label `l`, is a conjunct in D, then this is called **failed**;

– if D is conclusive and not failed then it is called **successful**.

Then, a derivation is a **successful derivation** iff there exists a successful disjunct D in G.

Given such a successful derivation, an **answer extracted from** D is the set of all abducible atoms, without labels, in D. Basically, our notion of extracted answer is the same as that of IFF, but ignoring the labels.

6 Soundness of the LIFF proof procedure

Theorem 1. *(*Soundness*) Let us assume that E is an answer extracted from a successful LIFF-derivation wrt Q and $\langle P, \mathcal{A}, I\rangle$, after any number of updates. Let $\langle P', \mathcal{A}', I'\rangle$ be the abductive logic program resulting after the updates. Then E is an explanation for Q, wrt $\langle P', \mathcal{A}', I'\rangle$.*

The proof of this theorem relies upon the following lemmas and the correctness of the IFF proof procedure.

Lemma 1. *Every non-static LIFF derivation wrt Q, $\langle P, \mathcal{A}, I\rangle$, $U_1, \ldots, U_m$, with end goal G and end abductive logic program $\langle P', \mathcal{A}', I'\rangle$, can be mapped onto a static LIFF derivation wrt Q, $\langle P', \mathcal{A}', I'\rangle$, ending at (a simplified version of) G.*

Lemma 2. *Every static LIFF derivation wrt Q, $\langle P, \mathcal{A}, I\rangle$, can be mapped onto an IFF derivation wrt Q.*

7 Conclusions and future work

In this paper we have proposed a dynamic abductive logic programming proof procedure, called LIFF. LIFF modifies the existing proof procedure IFF by adding labels to goals. The labels keep track of dependencies amongst the items (atoms and implications) in goals and between the items and the logic program. By keeping track of these dependencies, after updating the abductive logic program, LIFF can keep parts of the reasoning that have not been affected by the updates and determine how best to replace those parts that have been affected. It thus allows reasoning in dynamic environments and contexts without having to discard earlier reasoning when changes occur. We have considered updates consisting of addition and deletion of facts and rules and addition of integrity constraints.

We have not considered updates in the form of deletion of integrity constraints. These could be easily accommodated by adopting additional labels, associated to items (implications and atoms) in goals, to keep dependencies with any integrity constraints that have originated those items. It would be interesting also to consider the consequences of making different choices, for example enlarging the set of abducibles without adding new rules or integrity constraints.

LIFF manages to do most of its saving of earlier work when the definitions of relatively lower level predicates are modified (by additions and deletions). This kind of update is particularly prominent in the case when we use abductive logic programming and agent-based techniques for information integration [21, 16]. In such an application the higher level predicates are user-level and auxiliary predicates, and the lower level predicates are related to information sources. Changes in various aspects of the information sources, for example their content or availability, amount to updating these lower level predicates.

Our work on LIFF shares the same objectives as that of [5]. However whereas we adapt IFF for abductive logic programming, they adapt conventional SLDNF for ordinary logic programs. We also share some of the objectives of [17, 18]. In their work on speculative computation they allow reasoning to proceed with default values for specific facts and then accommodate updates which explicitly replace the default values, attempting to keep some of the earlier computation. They, however, use SLD derivation for Horn theories (no negation in rules and no integrity constraints). They keep a form of dependency, but it is at a much coarser level compared to ours. They keep dependency information for the whole goal, rather than for the individual items in goals, thus allowing fewer savings in computations.

We have described LIFF and a soundness result for propositional cases. Work is currently in progress for the predicate case, including the additional rules of factoring, case analysis and equality rewriting. Completeness results with respect to completeness results of IFF are subject of future work. Another interesting line of thought is to explore how we can parameterise the extent of savings in reasoning according to what is updated, to identify optimal and non-optimal cases. It would also be worth exploring how this work scales up to substantial applications that would require large knowledge bases and frequent updates.

Acknowledgements

This work has been supported by the IST programme of the EC, FET under the IST-2001-32530 SOCS project, within the GC proactive initiative.

References

1. C. E. Alchourron, P. Gardenfors, and D. Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.
2. K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
3. P. Dell'Acqua, F. Sadri, and F. Toni. Combining introspection and communication with rationality and reactivity in agents. In J. Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence, European Workshop, JELIA'98, Dagstuhl, Germany, October 12–15, 1998, Proceedings*, volume 1489 of *Lecture Notes in Computer Science*, pages 17–32. Springer-Verlag, 1998.

4. T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, 1997.
5. H. Hayashi. Replanning in robotics by dynamic sldnf. In *Proc. Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World*, 1999.
6. H. Hayashi, K. Cho, and A. Ohsuga. Integrating planning, action execution, knowledge updates and plan modifications via logic programming. In *Proc. ICLP2002*, 2002.
7. A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.
8. A.C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proc. ECAI-2004*, 2004.
9. R. A. Kowalski. *Logic for Problem Solving.* North-Holland, 1979.
10. R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25(3/4):391–419, 1999.
11. K. Kunen. Negation in logic programming. In *Journal of Logic Programming*, volume 4, pages 289–308, 1987.
12. P. Mancarella, F. Sadri, G. Terreni, and F. Toni. Planning partially for situated agents. In João Leite and Paolo Torroni, editors, *5th Workshop on Computational Logic in Multi-Agent Systems (CLIMA V)*, September 29–30 2004.
13. F. Sadri and F. Toni. Abduction with negation as failure for active and reactive rules. In E. Lamma and P. Mello, editors, *AI*IA'99: Advances in Artificial Intelligence, Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence, Bologna*, number 1792 in Lecture Notes in Artificial Intelligence, pages 49–60. Springer-Verlag, 2000.
14. F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In S. Greco and N. Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 419–431. Springer-Verlag, 2002.
15. F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In *Intelligent Agents VIII: 8th International Workshop, ATAL 2001, Seattle, WA, USA, Revised Papers*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 405–421. Springer-Verlag, 2002.
16. F. Sadri, F. Toni, and I. Xanthakos. An abductive framework for semantic integration of information. Technical report, Department of Computing, Imperial College London, 2004.
17. K. Satoh, K. Inoue, K. Iwanuma, and C Sakama. Speculative computation by abduction under incomplete communication environments. In *Proc. ICMAS2000*, pages 263–270, 2000.
18. K. Satoh and K. Yamamoto. Speculative computation with multi-agent belief revision. In *Proc. AAMAS2002*, pages 897–904, 2002.
19. F. Toni. Automated information management via abductive logic agents. *Journal of Telematics and Informatics*, 18(1):89–104, 2001.
20. F. Toni and K. Stathis. Access-as-you-need: a computational logic framework for flexible resource access in artificial societies. In *Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW'02)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2002.
21. I. Xanthakos. *Semantic Integration of Information by Abduction.* PhD thesis, Imperial College London, 2003.