Automated Information Management via Abductive Logic Agents

F.Toni [1]

Imperial College of Science, Technology and Medicine
Department of Computing
180 Queen's Gate, London SW7 2BZ, UK

Abstract

This paper describes an approach for managing information organised in networks of interconnected nodes. The nodes may be heterogeneous and distributed over a computer network, and nodes and links between them may be added and deleted dynamically as the information networks evolve over time.

The highly dynamic and distributed nature of these information networks poses problems of effective generation and maintenance, so that given properties (or meta-data) are satisfied as the networks evolve. The paper investigates the use of abductive logic (software) agents for the generation and maintenance of such networks. The agents rely upon a symbolic representation of networks and meta-data via abductive logic programs and make use of the reasoning engine abductive logic programs are equipped with for checking as well as enforcing the properties represented by the meta-data.

1 Introduction

This paper addresses the problem of managing information organised in networks of interconnected nodes. Nodes in such networks may be heterogeneous and distributed over a computer network. The links between nodes of the information networks represent symbolic connections between the information held at the nodes. The networks can grow dynamically, as nodes and links are added and deleted.

These networks support the complexity of collectively generated information. Indeed, they may be generated and evolved by ordinary, computing semi-

[1] E-mail: `ft@doc.ic.ac.uk`

literate, individuals interacting in a community, and may be the outcome of complex social interactions. Examples of these networks include the World Wide Web and networks for collective memory management [11].

The highly dynamic and distributed nature of these networks poses problems of effective generation and maintenance, so that given properties are satisfied. These properties can be thought of as *meta-data* on the (object-level) data held at the nodes of the network as well as on the structure of the network. For example, one might require that every node holding title and author of a book should link to two nodes, one holding a brief review of the content of the book, the other holding a list of addresses of libraries where copies of the book can be borrowed from. In addition to domain-specific meta-data of the above kind, one might require that any given network satisfies *structural properties*, e.g. that it contains no dangling links and no unreferenced nodes.

The paper investigates the use of *abductive logic (software) agents* that act on behalf of the individuals in the community to support the the generation and evolution of the information networks. Agents can be informally described as (software or hardware) computing artifacts capable of autonomous behaviour to achieve their given objectives while taking into account and reacting to the environment in which they are situated [10]. Abductive logic agents are general-purpose agents that can be employed for a number of other applications beyond the one considered in this paper [8,12]. They rely upon a symbolic, logic-based representation of the "knowledge" of the agents, 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 this paper, abductive logic programs in abductive logic agents are used to represent networks, meta-data and structural properties, and the agents are used for *checking* as well as *enforcing* the properties the meta-data represent as well as the required structural properties. The proposed techniques allow not only to identify that the networks exhibit unwanted features, with respect to the given properties, but also suggest possible *repair* strategies, in the form of actions that can be taken to put the given network into a format satisfying the properties. For example, the addition of new nodes and/or links might be recommended as a side-effect of user-given additions and/or deletions.

The paper is organised as follows. Section 2 provides the motivating examples we will concentrate on in the paper. Section 3 describes abductive logic programs and proof procedures. Section 4 decries the use of abductive logic programs and proof procedures for the problem of *checking* compliance of information networks with given properties. Section 5 describes the use of abductive logic programs and proof procedures for the problem of *repairing* information networks which do not comply with given properties, so that the properties are enforced. Section 6 illustrates the use of abductive logic agents in general and a concrete abductive logic agent architecture in particular for the maintenance of information networks. Section 7 evaluates the contribution of the paper, discusses directions for future research and presents comparisons with some related work. Section 8 concludes.

2 Motivating examples

In this section we illustrate some of the problems we want to address in this paper. Figure 1(a) exemplifies a simple information network with nodes n_0, n_1,

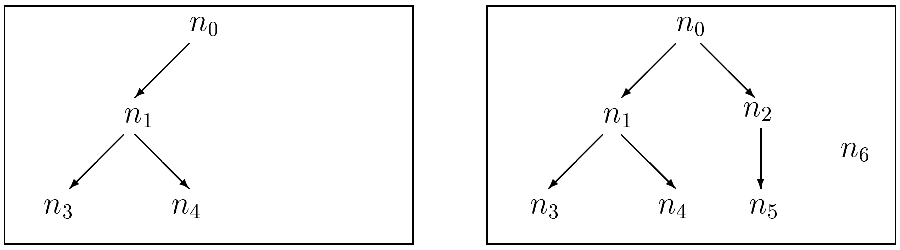

Fig. 1. (a) A simple information network. (b) Example of unreferenced node.

n_3, n_4 and links between nodes given by the directed edges. Here, we abstract away from the content of the nodes, and use identifiers n_i, $i = 1, \ldots, 4$, to represent them. Concretely, in a WWW setting, the n_i could be URLs of pages. The links represent symbolic connections between the information held at the nodes connected by the links. Concretely, in a WWW setting, the links could be pairs of URLs of pages.

For example, the network might hold information about books reviewed by individuals in a given community, to be accessed by other individuals. The node n_0 could be the entry node to the network, node n_1 could be about some concrete book1, node n_3 could contain a brief review of book1, and node n_4 could hold a list of addresses of libraries where copies of book1 can be borrowed.

The network evolves over time, as individuals add and delete nodes and links, and modify information held at nodes. Figure 1(b) shows a possible extension of the simple network in figure 1(a) by the addition of nodes n_2, n_5, n_6 and links from n_0 to n_2 and from n_2 to n_5. The extension violates the **structural property** that networks contain *no unreferenced nodes*. The extensions can be modified so that the property is satisfied, by adding a link from some node amongst $n_0, \ldots, n_5$ to n_6.

Figure 2(a) shows an alternative extension of the simple network in figure 1(a) by the addition of nodes n_2, n_5 and links from n_0 to n_2, from n_2 to n_5 and from

n_2 to some unspecified node. The extension violates the structural property that networks contain *no dangling links*. The extensions can be modified so that the property is satisfied, by adding a node to which the dangling link points.

Note that, even if the network is structurally correct, as in figure 2(b), it might violate some properties the information held by the network must satisfy. These properties can be seen as **meta-data**, because they refer to the data

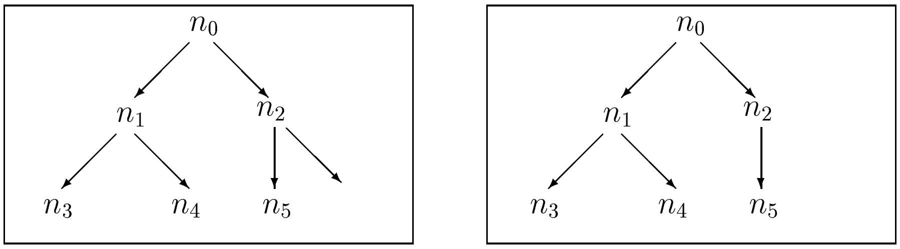

Fig. 2. (a) Example of dangling link. (b) Example of violation of meta-data.

(or information) held by the network. For example, one such property could be that for every book reviewed in the network both a brief review and a list of libraries holding copies of the book must be provided. Then, in figure 2(b), if nodes n_1, n_2 hold information about book1 and book2, respectively, nodes n_3, n_5 describe book1 and book2, respectively, and node n_4 holds a list of libraries from which book1 can be borrowed, then the network violates the above meta-data. The network can be repaired by adding a node n_6 holding the information about the required list of libraries and adding a link from n_2 to n_6.

In the remainder of the paper we will illustrate the use of existing general-purpose logic-based techniques [6,9,8,12,1] to detect violation of structural properties and meta-data in information networks and to suggest repairs automatically. These techniques rely upon the representation of networks, structural properties and meta-data via abductive logic programs, described in the next section.

3 Abductive logic programs and proof procedures

Abductive logic programs are extensions of logic programs by abduction [7]. **Abduction** is the reasoning mechanism allowing to infer premises of logical implications from their conclusions. For example, given the implication `A if B`, abduction allows to infer `B` from `A`: the premise `B` can be seen as an explanation for the observation `A`. Abduction can be seen as the "opposite" of **deduction**, which allows to infer conclusions of implications from their premises, for example `A` given that `B` is true, in the simple case above. Whereas deduction is a sound form of reasoning, abduction is not, as it amounts to adopting hypotheses, as explanations for observations, that might turn out to be invalid. It is however a very useful kind of reasoning to perform many human tasks, and it has been studied extensively both in philosophy and artificial intelligence [7].

3.1 Abductive logic programs

Information networks, structural properties and meta-data can be represented via **abductive logic programs**. These are triples $\langle P, A, I\rangle$ such that

- P is a **logic program**, i.e. a set of if-rules of the form
  `Head if Body`
  where `Head` is an atom, `Body` is a conjunction of atoms and negations of atoms, possibly *true*, and all variables in `Head` and `Body` are implicitly universally quantified from the outside.
  P is understood as a (possibly incomplete) set of data or beliefs by an agent.
- A is a set of variable-free atoms, referred to as **abducibles**.
  The atoms in A are understood as hypotheses that can be used to expand the logic program.
- I is a set of **integrity constraints**, i.e. if-then-rules of the form
  `if Conditions then Conclusion`
  where `Conditions` is a conjunction of atoms and negations of atoms, possibly *true*, `Conclusion` is an atom, possibly *false*, and all variables in `Conditions` and `Conclusion` are implicitly universally quantified from the outside.
  Sentences in I are understood as sentences that must be "satisfied" by any sets of abducibles expanding the logic program, in the same way integrity constraints in databases are understood as sentences that must be "satisfied" in every database state.

To illustrate the concept of abductive logic programs $\langle P, A, I\rangle$ in the context of this paper, consider the following simple example. The logic program P consists of the if-rules

```
node(n1) if true
node(n2) if true
white(n1) if true
white-node(X) if node(X) and white(X)
black-node(X) if node(X) and black(X)
```

representing the data that $\mathtt{n}_1$ is a node (in some network), that it is white, that every node that is white is called a white-node and that every node that is black is called a black-node. Here and elsewhere in the paper we adopt the convention that constant and predicate symbols start with a lower-case letter, whereas variables start with upper-case letters.

Note that we often write an if-rule with *true* as its `Body` simply by means of the `Head`. Namely, the first two if-rules in P above can be written simply as

```
node(n1)
node(n2)
white(n1)
```

We will adopt a similar convention for if-then-rules in the integrity constraints, by representing an if-then-rule with *true* as its `Conditions` simply by means of the `Conclusion`.

Continuing the example, the set of integrity constraints I might consist of the if-then-rules

```
if link(X,Y) and white-node(X) then black-node(Y)
if white(X) and black(X) then false
```

representing the properties that every link from a white node must end in a black node, and that no object can be both white and black at the same time. Finally, the set of abducibles A might consist of the set of all variable-free atoms in the predicates `link` and `black`, i.e., assuming $\mathtt{n}_1$ and $\mathtt{n}_2$ are the only two terms allowed in the language, $A = \{\mathtt{black(n_1)},\ \mathtt{black(n_2)},\ \mathtt{link(n_1,n_2)},\ \mathtt{link(n_2,n_1)}\}$.

Given an abductive logic program as above and an **observation** (conjunction of literals) O, an **(abductive) explanation** for O is a pair (D, θ) consisting of a (possibly empty) set of abducibles D, and a (possibly empty) variable substitution θ for the variables in O, such that

(1) $P \cup D$ **entails** $O\,\theta$, and
(2) $P \cup D$ **satisfies** I.

Various notions of entailment and satisfaction can be adopted, for example entailment could be entailment in first-order, classical logic and satisfaction could be consistency in first-order, classical logic. However, the provision of such notions is beyond the scope of this paper, see [7] for a detailed discussion.

For the simple example given earlier in this section

- the observation O_1=`white-node(n1)` (to be understood as a query) admits the single explanation $(\{\}, \{\})$, as O_1 is entailed by P alone and $P \cup \{\}$ satisfies the integrity constraints;
- the observation O_2=`link(n1,n2)` (to be understood as an input) admits

the single explanation ({`link(`n_1`,`n_2`)`, `black(`n_2`)`}, {}); note that the pair ({`link(`n_1`,`n_2`)`}, {}) does not constitute an explanation as it does not satisfy the first integrity constraint in I;
- the observation O_3=`link(`n_1`,`n_2`) and white(`n_2`)` (to be understood as an input) admits no explanation, as no candidate set entailing the observation can possibly satisfy both integrity constraints in I;
- the observation O_4=`link(X,`n_2`) and node(X) and white(X)` admits the single explanation ({`link(`n_1`,`n_2`)`},{X=n_1}); if the logic program is P' given by P together with the if-rules

  `node(`n_3`)`

  `white(`n_3`)`

  then O_4 admits two alternative explanations, namely the one given earlier and ({`link(`n_3`,`n_2`)`},{X=n_3});
- the observation O_5=`link(X,`n_2`) and node(X) and not white(X)` admits the single explanation ({`link(`n_2`,`n_2`)`},{X=n_2}).

3.2 Abductive proof procedures

An abductive proof procedure is an engine for the automatic computation of explanations for observations. Examples of abductive proof procedures can be found in [7,6,9,12]. Typically, abductive proof procedures interleave **backward reasoning** with if-rules in the logic program (reduction of `Head` to `Body`) and **forward reasoning** with if-then-rules in the integrity constraints (addition of `Conclusion` to `Conditions`). Backward reasoning with the if-rules allows to generate the kernel of candidate explanations, whereas forward reasoning with the if-then-rules allows to rule out unwanted candidate explanations and expand the kernel of promising ones. Indeed, some kernel explanations might allow to explain the observation but might fail to take into account the integrity constraints, whereas others might need to be augmented in order to do so. The interleaving terminates when no more backward or forward reasoning can be performed to a branch in the tree of candidate explanations: the explanation can be extracted from the branch.

The abductive logic agents described in section 6 rely upon the proof procedure of [6]. In this proof procedure, all if-rules defining a predicate are combined in **if-and-only-if definitions** (**iff-definitions** in short) before backward reasoning is applied. For the simple example given earlier in this section, the iff-definitions are

```
node(X) iff X=n1 or X=n2
white(X) iff X=n1
white-node(X) iff node(X) and white(X)
black-node(X) iff node(X) and black(X)
```

Note that there are no iff-definitions for the predicates `link` and `black` as they are abducible predicates.

In general, the set of iff-definitions can be generated automatically from the logic program by applying the **selective completion** to it [12].

A formal description of the proof procedure in [6] is beyond the scope of this paper. Here we (loosely) illustrate its behaviour for the simple example and observations given earlier in this section: [2]

- given O_1, the proof procedure applies backward reasoning with the selective completion of P to O_1 to generate first `node(n`$_1$`) and white(n`$_1$`)`, and then *true*, corresponding to the single (empty) explanation given earlier;
- given O_2, the proof procedure first applies forward reasoning with the first integrity constraint in I to O_2, to generate the residual integrity constraint `if white-node(n`$_1$`) then black-node(n`$_2$`)`, then backward reasoning with the selective completion of P to the condition of the residual integrity constraint to generate the further residual `if` *true* `then black-node(n`$_2$`)`, then forward reasoning to generate `black-node(n`$_2$`)`, and finally (two steps of) backward reasoning from it to generate `black(n`$_2$`)`, thus computing the single explanation given earlier;
- given O_3, the proof procedure interleaves forward reasoning with the first integrity constraint in I and its residuals and backward reasoning with the selective completion of P to O_3 as for O_2, and then applies further steps of forward reasoning with the second integrity constraint in I giving $false$, corresponding to no explanation;
- given O_4, the proof procedure applies backward reasoning with the selective completion of P to O_4 to generate first
  `[link(n`$_1$`,n`$_2$`) and white(n`$_1$`)] or [link(n`$_2$`,n`$_2$`) and white(n`$_2$`)]`
  and then `link(n`$_1$`,n`$_2$`)`, corresponding to the explanation given earlier on;
- given O_5, the proof procedure applies backward reasoning with the selective completion of P to O_5 to generate first
  `[link(n`$_1$`,n`$_2$`) and (if white(n`$_1$`)then` $false$`)] or`
  `[link(n`$_2$`,n`$_2$`) and (if white(n`$_2$`) then` $false$`)]`
  and then
  `[link(n`$_1$`,n`$_2$`) and (if n`$_1$`=n`$_1$` then` $false$`)] or`
  `[link(n`$_2$`,n`$_2$`) and (if n`$_2$`=n`$_1$` then` $false$`)]`
  then forward reasoning to reject the first ($false$) disjunct and generate `link(n`$_2$`,n`$_2$`)`, corresponding to the explanation given earlier on.

[2] Note that backward reasoning as described and used in this paper corresponds to a combination of operations in [6], namely *unfolding, equality rewriting, splitting*, whereas forward reasoning corresponds to the single operation of *propagation.*

4 Checking integrity of information networks

Information networks can be represented via sets of atoms in the predicates *link* and *node*. For example, the simple network in figure 1(a) can be represented by $P_{1(a)}$ consisting of the if-rules:

```
link(n0,n1)
link(n1,n3)
link(n1,n4)
node(n0)
node(n1)
node(n3)
node(n4)
```

Note that we do not suggest to replace existing networks by representations such as the above, but rather to couple the networks with such representations, so that it is possible to reason and operate on them.

The representation of networks as suggested above is equivalent to a relational database, and if-rules in logic programs as we will introduce later amount to views in deductive databases. Semantic query optimisation technology in deductive databases [2] could indeed be adopted to check integrity of information networks. However, as we will discuss later on in the conclusions, it is not clear how such technology could be used to repair and maintain the networks dynamically (see next section).

The structural property that there are no dangling links can be represented by means of the set of integrity constraints I_{dang} consisting of the if-then-rules:

```
if link(X,Y) then node(X)
if link(X,Y) then node(Y)
```

expressing that every link in a network connects two existing nodes.

The additional set of if-rules $P_{1(a),type}$c consisting of:

```
book(n1)
review(n3)
libraries(n4)
```

represents the "type" of the nodes, namely the sort of information held at the nodes. In particular, the node referred to as n_1 holds the description of a book, the node referred to as n_3 holds a review of the book described in n_1 (as there is a link between n_1 and n_3) and the node referred to as n_4 holds a list of libraries the book described in n_1 (as there is a link between n_1 and n_4)

can be borrowed from.

Note that this representation is satisfactory in the context of the motivating examples in section 2. However, a richer representation might be required for other examples, for example one might adopt

```
book(N, Title, Publisher, Year, ISBN)
```

where `N, Title, Publisher, Year, ISBN` are variable names hinting at their possible values. This richer representation would be valuable if the description of the book at `N` contains title, publisher, year and ISBN of the book, and this information needs to be accessed in order to enforce some of the meta-data of interest. An alternative representation for the same purpose would be

```
book(N)
title(N,Title)
publisher(N,Publisher)
year(N,Year)
isbn(N,ISBN)
```

introducing multiple predicates.

The meta-data that every "book" node should be linked to a "review" node and a "libraries" node can be expressed by I_{meta} consisting of the if-then-rule:

```
if book(X) then two-links(X)
```

where the predicate `two-links` is defined by the logic program P_{meta} consisting of the if-rule:

```
two-links(X)  if link(X,Y) and review(Y)
              and link(X,Z) and libraries(Z)
```

Then, given the abductive logic program $\langle P_{1(a)} \cup P_{1(a),type}, \{\}, I_{dang} \cup I_{meta} \rangle$, where the set of abducibles is empty, any correct abductive proof procedure returns ($\{\}$,$\{\}$) as an explanation for the empty observation. Indeed, the given network satisfies the structural property as well as the meta-data on the information it contains.

However, let $P_{2(b)}$ be the logic program consisting of the if-rules:

$\texttt{node(}n_2\texttt{)}$
$\texttt{node(}n_5\texttt{)}$
$\texttt{link(}n_0\texttt{,}n_2\texttt{)}$
$\texttt{link(}n_2\texttt{,}n_5\texttt{)}$

and $P_{2(b),type}$ be the logic program consisting of the if-rules:

```
book(n_2)
review(n_5)
```

Note that $P_{1(a)} \cup P_{2(b)}$ represents the network in figure 2(b). Moreover, $P_{2(b),type}$ represent the "type" of the new nodes in the network in figure 2(b). Then, given the abductive logic program $\langle P_{1(a)} \cup P_{2(b)} \cup P_{1(a),type} \cup P_{2(b),type}, \{\}, I_{dang} \cup I_{meta} \rangle$, where the set of abducibles is again empty, any correct abductive proof procedure returns $false$ as an explanation for the empty observation. Indeed, the given network violates the meta-data.

So far we have illustrated how abductive proof procedures can be used to detect compliance of given information networks with required structural properties and meta-data, based upon the representation of networks, properties and meta-data via abductive logic programs with an empty set of abducibles. In the next section we will illustrate how the same techniques can be used to suggest repairs to non-compliant networks, by modifying the knowledge representation (namely the abductive logic program).

5 Repairing information networks

We introduce predicates `derived-node`, `add-node`, `derived-link`, `add-link`, and a new logic program P^{new} consisting of if-rules:

```
derived-node(X) if node(X)
derived-node(X) if add-node(X)
derived-link(X,Y) if link(X,Y)
derived-link(X,Y) if add-link(X,Y)
```

Intuitively, predicates `node` and `links` are used to represent given networks as above, and `add-node` and `add-link` can be seen as repair actions. Indeed, all atoms in the predicates `add-node` and `add-link` are *abducibles*, namely they can be added to the given logic program if needed, e.g. to avoid the integrity constraints to be violated.

The additional integrity constraints I^{false}:

```
if add-link(X,Y) and link(X,Y) then false
if add-node(X) and node(X) then false
```

guarantee that no repair action is taken unless absolutely necessary.

The predicates `derived-node` and `derived-link` are used to describe "intentionally" repaired networks, before the networks are physically created. The system architecture described in the next section can be used as a support to revise networks physically according to recommended repairs.

Similarly, we introduce predicates `derived-book`, `add-book`, `derived-review`, `add-review`, `derived-libraries`, `add-libraries`, and add to P^{new} analogous definitions as the ones above.

The integrity constraints I_{dang} and I_{meta} are rewritten as I^{new}_{dang} and I^{new}_{meta}, respectively, where I^{new}_{dang} consists of if-then-rules:

```
if derived-link(X,Y) then derived-node(X)
if derived-link(X,Y) then derived-node(Y)
```

and I^{new}_{meta} consists of the if-then-rule

```
if derived-book(X) then two-links(X)
```

and the predicate `two-links` is defined by P^{new}_{meta}:

```
two-links(X) if derived-link(X,Y)
             and derived-review(Y)
             and derived-link(X,Z)
             and derived-libraries(Z)
```

Let A be the set of abducibles

{`add-node(t)`, `add-link(t,t')`, `add-book(t)`,
`add-review(t)`, `add-libraries(t)` |
`t,t'` are variable-free terms in the vocabulary of the network}.

Then, given the abductive logic program

$\langle P_{1(a)} \cup P_{1(a),type} \cup P^{new} \cup P^{new}_{meta}, A, I^{new}_{dang} \cup I^{new}_{meta} \rangle$,

and an observation O (set of input actions)

add-link(n_0,n_2) and
add-link(n_2,n_5) and
add-node(n_2) and
add-node(n_5) and
add-book(n_2) and
add-review(n_5)

representing the extension of the network in figure 1(a) to that in figure2(b), any correct abductive proof procedure computes the explanation

$(\{\texttt{add-link(n}_2\texttt{,X)},\ \texttt{add-libraries(X)}\} \cup O, \{\})$

where the variable `X` is existentially quantified. The subset of this explanation not in the original O can be seen as a recommended repair to the network.

6 Abductive logic agents

Abductive logic programs and proof procedures can be used to represent (software and hardware) agents, and abductive proof procedures can serve as the engine underlying the reasoning process of these agents [8,12]. In particular,

- logic programs can be thought of as (possibly incomplete) beliefs of agents, describing, for example, how to construct plans to achieve given objectives;
- abducibles can be thought of as candidate data that can be used to extend the beliefs of agents as well as (physical and communicative) actions that agents may need or choose to perform in order to achieve their own objectives and to react to inputs they receive from the environment in which they are situated;
- integrity constraints can be thought of as rules of behaviour, condition-action and commitment rules that agents should obey to in order to construct their plans dynamically, as well as obligations, prohibitions, and communication protocols the agent's behaviour should conform to;
- observations can be thought of as agents' objectives as well as incoming inputs agents need to explain and react to;
- explanations can be thought of as data to be actually added to the beliefs of the agents and actions that, if successfully performed by the agents, would allow for the agents to achieve the given objectives while taking into account the agents' beliefs, prohibitions, obligations, rules of behaviour and so on, in the circumstances given by the current inputs.

In abductive logic agents, the proof procedure is executed within an "observe-reason-act" cycle that allows the agent to be alert to the environment and react to it as well as think and devise plans [8]. This cycle interleaves the execution of the abductive proof procedure (*reason*) for the computation of explanations for observations, the absorption of inputs from the environment, to be added dynamically to the observation to be explained (*observe*) and the selection of actions (i.e. abducibles in explanations) to be performed in the environment (*act*). Note that, in order for the agents to be *reactive* to the environment in which they are situated, agents might need to select actions and perform them before the computation of explanations/plans is finalised. This also allows to get feedback from the environment before too much time is spent on an unpromising plan.

More concretely, the observe-reason-act cycle can be described as follows.

To cycle at time t,

(i) observe any input at time t,
(ii) record any such input,
(iii) resume the IFF procedure by reasoning forwards from the inputs,
(iv) continue applying the IFF procedure,
using for steps (iii) and (iv) a total of r units of time,
(v) select an atomic action which can be executed at time $t + r + 2$,
(vi) execute the selected action at time $t + r + 2$ and record the result,
(vii) cycle at time $t + r + 3$.

The cycle starts at time t by observing and recording any inputs from the environment (steps (i) and (ii)). Time t is the clock of the agent. Steps (i) and (ii) are assumed to take one unit of time each. Then, the proof procedure is applied for r units of time (steps (iii) and (iv)). The amount of resources r available in steps (iii) and (iv) is bounded by some predefined amount n. By decreasing n the agent is more reactive, by increasing n the agent is more rational. Forward reasoning is applied first (step (iii)), in order to allow for an appropriate reaction to the inputs. Afterwards, an action is selected and executed, taking care of recording the result (steps (v) and (vi)). Steps (v) and (vi) conjoined are assumed to take one unit of time. Selected actions can be thought of as outputs into the environment, and observations as inputs from the environment. From every agent's viewpoint, the environment contains all other agents. Selected actions correspond to abducibles in an explanation computed by the proof procedure.

This understanding of abductive logic programs and proof procedures is at the heart of the system architecture underlying the prototype implementation described in [1,13]. In the implementation, the Java programming language is used for implementing the agents and their interactions with the environment, and Voyager (see `http://www.objectspace.com/`) for implementing the inter-agent features (such as communication). Java has been chosen for its portability and platform-independence, as well as because it allows to design clean interfaces. Voyager has been chosen because it integrates well with Java and is free. Of course, other choices would have been possible. Note that the adoption of abductive logic programming for the representation and reasoning of agents does not impose the choice of PROLOG as the underlying implementation language, nor would such choice render the implementation simpler overall, due to the need to incorporate inter-agent features via some additional language to be integrated with it.

In the implementation, the execution of the abductive proof procedure presented in [6] is interleaved with the analysis of incoming observations and the

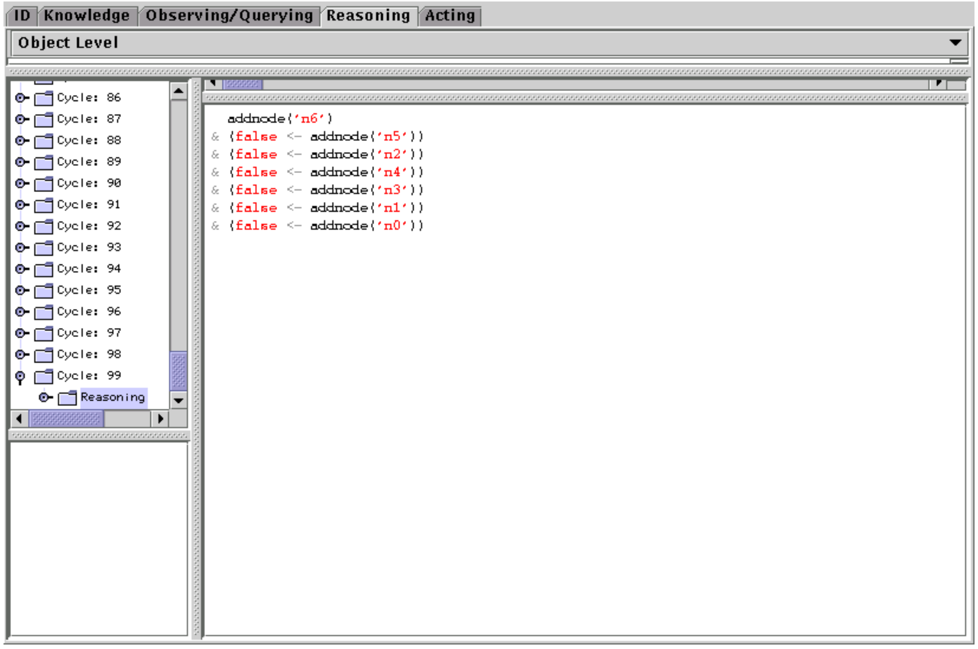

Fig. 3. Monitoring the reasoning of the agent

execution of selected actions, via a resource-bounded sense-reason-act cycle. The level of rationality/reactivity of agents depends on the amount of resources assigned to the reasoning part of the cycle (the more the resources, the more rational, but less reactive, the agent).

Figure 3 illustrates the component of the system implementation responsible for viewing and monitoring the Reasoning activity of the agent. The identifier (ID) of an agent is the name other agents use to interact (via communication) with the given agent. The Knowledge of the agent is given by an abductive logic program. New observations (and queries) to the agent can be inputed via the Observing/Querying component of the implementation. Actions are abducibles in selected explanations and are dynamically selected, as they are generated, by the Acting component. The Reasoning component repeatedly applies, a pre-defined number of times, amounting to the level of rationality, the operations of the underlying abductive proof procedure.

In the concrete case shown in figure 3, the knowledge component (not shown) is the abductive logic program representing the network in figure 2(a) and the structural property that there are no dangling links in information network, in the spirit of section 5. The right-most box on the screen shows the explanation computed for the empty observation. Note that the suggested repair (correctly) amounts to adding an additional node, n_6. The remaining conjuncts in the explanation are "residues" of the integrity constraint that no node should be added if already existent in the network.

7 Discussion

The advantages of the techniques described in the paper are threefold.

First, abductive logic programming is a rich formalism for knowledge representation that lends itself to representing a number of features and properties of networks. Its application has already been studied in many different settings with interesting results [7,9], and its theoretical foundations are solid. As discussed in section 4, deductive databases could be used for representing networks and their properties, and semantic query optimisation [2] for checking the properties. Indeed, abductive logic programming can be seen as a generalisation of deductive databases [9]. However, abductive logic programming has the advantage of having being encapsulated within an agent setting, thus being a good candidate for automatisation. This is especially beneficial when the original networks are generated by ordinary individuals, since in this setting the integrity of the network is likely to be violated often and in a decentralised manner. Note that we have not advocated the re-formulations of networks in terms of abductive logic programs to *replace* the existing networks, but rather to *adjoin* them, for the purpose of checking and enforcing properties on them, in a way similar to XML code.

Second, abductive logic programs are equipped with proof procedures for reasoning automatically with them. We have seen that these tools can be used to check for properties to be satisfied and to suggest repairs when they are not. As the proof procedures are sound tools whose behaviour is well-determined, any outcome they provide, in terms of property satisfaction and network repair, is guaranteed to be a "good" one. Thus, if the recommended repair is indeed performed, the resulting network is guaranteed to satisfy the properties. Note that this is simply a by-product of the soundness of the proof procedures.

Third, abductive proof procedures can be executed within an agent architecture, thus serving as a basis for a completely autonomous tool for network handling. This tool would check for properties to be satisfied, would suggest repairs when needed, and would either implement repairs directly, whenever possible, or guide the individuals creating the network through the process of repair. Moreover, the agent formulation paves the way to a distributed, multi-agent modeling of the problem, where different agents are responsible for checking and enforcing different properties and/or different parts of the network. This would allow higher modularity and reliability, as well as efficiency. The provision of a fully-automated, multi-agent system for network handling is a subject for future research.

Our proposal is somewhat related, in its aims, to the proposal of [4]. However, we propose to adapt general-purpose techniques, namely abductive logic programs and proof procedures, rather than define ad-hoc tools for the problem at hand. Moreover, we propose to use these techniques within an agent-oriented setting, to allow for automatisation.

Future research also includes comparison and possibly integration with tools such as the ones described in [3,5].

8 Conclusions

We have described how

(1) abductive logic programs, an existing general-purpose tool for knowledge representation in artificial intelligence, can be used to specify desired properties of networks of information
(2) abductive proof procedures, existing automated reasoning tools abductive logic programs are equipped with, can be used to check and enforce these properties,
(3) abductive logic agents, namely autonomous entities whose beliefs are represented by abductive logic programs and whose behaviour is achieved by interleaving the execution of abductive proof procedures, can be used to fully automatise the enforcement of the properties on the networks.

We have argued that an agent architectures relying upon the above mentioned logic-based techniques would pave the way to autonomous and distributed tools for network handling.

Acknowledgements

This research was supported by the EPSRC project "Logic-based multi-agent systems". The author is grateful to Kostas Stathis and anonymous referees for many useful suggestions.

References

[1] Y. Carer, *Distributed communicating agents.* Individual Project, Department of Computing, Imperial College, 1998/1999.

[2] U.S. Chakravarthy, J. Grant, J. Minker, Foundations of Semantic Query Optimization for Deductive Databases. *Foundations of Deductive Databases and Logic Programming*, pages 243–273, Minker, (ed.), Morgan Kaufmann, 1988.

[3] M.L. Creech, Author-oriented link management. *Computer Networks and ISDN Systems*, 28(7–11):1015, 1996.

[4] M. Fernández, D. Florescu, A. Levy, D. Suciu, Verifying integrity constraints on web sites. *Proc. IJCAI'99, 16th International Joint Conference on Artificial Intelligence*, pages 36–41, T. Dean (ed.), Morgan Kaufmann, 1999.

[5] R. Fielding, Mantaining distributed hypertext infostructures: welcome to MOMSpider's web. *Proc. First International Conference on the World Wide Web*, 1994.

[6] T.H. Fung, R.A. Kowalski, The iff procedure for abductive logic programming. *Journal of Logic Programming* 33(2):151–165, Elsevier.

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

[8] R.A. Kowalski, F. Sadri, From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, Baltzer Science Publishers, Editors: J. Dix, J. Lobo, volume 25(3,4):391–420, 1999.

[9] R.A. Kowalski, F. Toni, G. Wetzel, Executing suspended logic programs. *Fundamenta Informaticae* 34(3):203–224, ISO Press, 1998.

[10] N.R. Jennings, K. Sycara, M. Wooldridge, A Roadmap of Agent Research and Development, *International Journal of Autonomous Agents and Multi-Agent Systems* 1(1):7-38, 1998.

[11] A. Mamdani, J. Pitt, K. Stathis, Connected communities from the standpoint of multi-agent systems. *New Generation Computing Journal*, Special issue on New Challenges in Intelligent Systems, 17(4), T. Nishida (ed.), 1999.

[12] F. Sadri, F. Toni, Abduction with negation as failure for active and reactive rules. *Proc. AI*IA 99, 6th Congress of the Italian Association for Artificial Intelligence*, E. Lamma and P. Mello eds, Springer Verlag LNAI 1792, pages 49-60, 2000.

[13] F. Sadri, F. Toni, A Logic-Based Multi-Agent System, Technical Report, 1999.