Logic Programming Methods for Searching the Web

(Preliminary Report)

Helmut Prendinger*
University of California, Irvine
Irvine, CA 92717-4555
Email: hprendin@benfranklin.hnet.uci.edu

Abstract

In this paper, we propose a method for approximate reasoning within an abductive logic programming setting. Our investigation is motivated by applying abductive reasoning to the problem of searching the web. Retrieval of user-relevant information is modeled as the task of generating abductive solutions (explanations) to a search query. In order to rule out (logically relevant) solutions (web sites) if they are uninteresting to the user, factual relevance criteria are applied. The main contribution of our paper is a well-founded method of restricting the set of abductive solutions in a way that user-relevant information (i) can be retrieved even if the knowledge base is partly explored, and (ii) is preserved over activated hyperlinks.

Topic areas: LP-based net search tools and spiders.

1 Introduction and Motivation

As AI research shifts from solving 'toy problems' towards real-world applications, methods are called for that reduce the computational complexity inherent in virtually all interesting problem solving tasks. If knowledge bases are allowed to contain arbitrarily large amounts of information such as the world-wide web, forms of *limited* or *approximate* reasoning will play a predominant rôle. The method of approximate (relevant) reasoning is foundational work as part of a larger enterprise, the design and implementation of a *searching softbot* (*soft*ware ro*bot*) for the web. The softbot is intended to post queries to multiple search engines (for instance, Lycos and InfoSeek) and gather the returned pages. Retrieved information is modeled as abductive solutions to a query. In the spirit of MetaCrawler [24], the softbot is a meta-service relying on standard search engines and concentrates solely on the retrieval task. The prime target of the softbot is the selection of the most *interesting* information relative to user-needs; that is, in addition to the logical notion of relevance (via reachability relations) we consider forms of *factual* relevance that depend on user-interests and the domain under consideration.

In this paper, we suggest a method for *approximate relevant* reasoning that combines (most) advantages of both limited [23] and relevant reasoning [13]. The idea of limited (or approximate) reasoning is to make reasoning easier by restricting the classical notion of entailment to a subset $S \subseteq P^0$ of the language [12, 23]; propositional variables $p \in P^0 \backslash S$

*On leave from University of Salzburg, Austria.

are 'ignored' in the derivation of the goal. Put more positively, limited reasoning enables partial *views* comparable to 'focussing attention' in inductive tasks [27]. Computational savings are immediate, *provided* a favorable set S is readily found; yet in [23] no way is shown how to generate the set S of predicates *relevant* for proving or disproving a given goal. By employing *reachability relations* [3] it is guaranteed that only those parts of the knowledge base are accessed that are relevant for (dis)proving a given goal.

We will develop our method within an *abductive* logic programming framework. In short, the method approximates the set of all explanations by incrementally refining the answer to a query. Abduction is an inference of the form "from q and $q \leftarrow p$ infer p", that is, from q and the implication $q \leftarrow p$ abduction generates p as a possible explanation for q. Generally, given a knowledge base KB, sentences I (a set of *integrity constraints*), and a sentence G, the abductive process searches for sets Δ_i of sentences such that [10]: (i) $KB \cup \Delta_i \vdash G$; (ii) $KB \cup \Delta_i \cup I$ is consistent; (iii) Δ_i respects further restrictions that make it an 'interesting' abductive explanation for G. Multiple explanations $\Delta_1, ..., \Delta_n$ is a general phenomenon in abductive reasoning. In addition to criteria which help to filter out 'preferred' hypotheses (item ii), explanations are often restricted to some pre-specified set $\mathcal{A}$ of abducible predicates (item iii). In the literature, $\mathcal{A}$ is usually conceived as a set of causes. Since we want to be neutral w.r.t. the status of the outcome of the abductive process, we will call results 'abductive solutions' rather than explanations.

The paper is organized as follows. In Section 2, we report on a (logical) concept of relevance due to [3] and relate it to the proof procedure of logic programming. In Section 3, we first introduce the notions of abduction problem and abductive solution; the abductive framework will be illustrated by means of a web searching problem. Then an approximation of the abductive proof procedure is proposed that generates a subset of all (logically) relevant abductive solutions. Section 4 is dedicated to the problem of selecting the (factually) relevant, that is, interesting solutions to a query. In section 5, we briefly describe the implementation of a prototype agent in Prolog. In the last section we discuss related work and suggest some refinements of our framework.

2 Relevant and Limited Reasoning

Most work on relevance in deductive reasoning [27, 13] relies on processing the 'complete' knowledge base. For instance, the relevancy detection algorithm described by Levy and Sagiv [13] identifies irrelevant clauses, but considerations of relevance are made *after* preprocessing a fixed knowledge base. When searching the web, here conceived as a knowledge base, the situation is entirely different since the complete knowledge base is never given in advance. The relevance concept of Brüning and Schaub [3] is robust for incompletely processed knowledge bases. This is accomplished by so-called *reachability relations*. Intuitively, reachability relations select those parts of the knowledge base that are relevant for (dis)proving a given query, that is, a literal K is reachable from a literal L if the derivation of a clause containing K might contribute to a refutation of L. It will turn out that this notion of relevance is sufficient for our representation of a web searching problem; if we proceed to more complex domain representations (for instance, Horn formulas with variables or recursive Horn formulas), we need to formulate relevance by utilizing, for instance, the *query tree* idea of Levy and Sagiv [13]. Moreover, Brüning and Schaub propose to compute values of further concepts which are intended to estimate the relevance of a literal K to another literal L; for instance, starting with a literal L the concept of *distance* measures how many steps must be taken in order reach a clause with some literal K. The following statement applies to the language of (propositional) Horn clauses.

Observation 2.1
The set of literals reachable from a literal p is identical to the set of literals resolved upon in the refutation mechanism of SLD-resolution with query $?-p$, provided the whole search tree is explored.

What this observation actually tells us is that instead of computing the reachability relation, we can execute a logic program and be confident that only relevant literals are resolved upon. SLD-resolution is a well-known procedure to prove that a given query follows from a Horn program [15]; the SLD mechanism constructs a refutation $Q^0, ..., Q^n$ where Q^0 is the original query and Q^n is the empty clause $\Box$, denoting a contradiction. The next observation relates the concept of distance to SLD-resolution.

Observation 2.2
The value of distance $d(p, q)$ corresponds to the minimum number of resolution steps taken to resolve upon q, where $?-p$ is the query.

As opposed to the arbitrary choice of 'interesting' predicates in the limited reasoning approach of [23], reasoning with reachability relations is query-sensitive. The same holds for the resolution mechanism of SLD-resolution. Below we will show how to approximate the set of abductive solutions by increasing the distance of literals (relative to a query) considered for computation. In the terminology of [23], this may count as a *principled* way to find sets $S \subseteq P^0$ of relevant letters.

3 Approximating Abduction

In this section we first introduce some terminology relevant to abductive logic programming. Then we show how the results of a searching process can be represented as an abductive logic program. In the main part of the section we suggest an approximate version of the abductive framework.

3.1 Basic Notions

Let $\mathcal{L}$ be a first order language, with the notions of alphabet, term, atom, and literal introduced as in [15]. Variables are denoted by capital letters $X, Y, Z, ...$; functors by $f/n, g/n, ...$ (n for the number of arguments, the arity); constants are treated as 0-ary function symbols. Predicate symbols are denoted by $p/n, q/n, ...$; propositional variables $p, q, ...$ are treated as 0-ary predicate symbols. Terms will be denoted by $s, t, ...$. For convenience, tuples of variables $(X_1, ..., X_n)$ and $(Y_1, ..., Y_m)$ are denoted by $\overline{X}$ and $\overline{Y}$, respectively.

A *definite (or Horn) clause* (based on $\mathcal{L}$) is a formula of the form $A :- B_1, ..., B_n$ where $A, B_1, ..., B_n$ are atoms. If $n > 0$, the formula is called a *rule*, else a *fact*. An *atomic definite query* is a formula of the form $?-B$ where B is an atom. The *definition* for a predicate p/n of $\mathcal{L}$ are all clauses with consequent (head) p/n. A (definite) *logic program* P is a set of definitions for predicates (not including "="). If a predicate has no definition in P, it is called *abducible*. $\mathcal{A}$ is the set of abducible predicates (in P). A logic program P with predicates which have no definition in P is called *abductive*.

The problem of searching the web can be related to the classical AI problem of searching in a graph. We perceive of each web page as a node and of the hypertext links to other web pages as edges. Note that the web is not of the tree type in which each node has at most one parent node since more than one web page may set a link to a particular page; moreover, the graph corresponding to the web is not acyclic, that is, some web pages may have links to ancestor pages. For simplicity, we assume that the graph structure is transformed to *acyclic*[1] tree form by means of pruning nodes that have been visited before.

Definition 3.1 (abduction problem)
An *abduction problem* is a quadrupel $\langle P, Q, I, \mathcal{A}\rangle$ where (i) P is an abductive acyclic (definite) program, (ii) Q is an atomic (definite) query not containing an abducible predicate, (iii) I is a set of integrity constraints, and (iv) $\mathcal{A}$ is a set of abducible predicates.

The *abductive* extension of SLD-resolution is performing the following task [17]: Given a query Q and a program P, the abductive procedure computes a set Δ of ground abducible facts and an answer substitution θ such that $P + \Delta \vdash \theta(Q)$.

Definition 3.2 (abductive solution)
Let $\langle P, Q, I, \mathcal{A}\rangle$ be an abduction problem. An *abductive solution* for Q (in P) is the resulting set $\Delta_i \subseteq \mathcal{L}(\mathcal{A})$ of a SLDA-refutation that satisfies I.

Our searching tasks will allow for very simple representations (see Table 1 below). First, programs are ground, that is, they do not contain variables; so we need not bother with unification. For different terms s and t, we have $s \neq t$. This is the special case $n = m = 0$ of the freeness axiom $\forall \overline{X}, \overline{Y}(f(\overline{X}) \neq g(\overline{Y}))$ for each pair of different functors f/n and g/m in $\mathcal{L}$ $(n, m \geq 0)$. Second, since search results can be represented as definite programs, we do not have to deal with negative information. In effect, the representational complexity is on the level of propositional definite programs.

In fact, we will not deal with abduction problems such that $I \neq \emptyset$ until Section 4 where integrity constraints are introduced formally. Intuitively, integrity constraints are used to select among abductive solutions.

3.2 A Simple Searching Example

Given a *query* (a search string entered by the user), a search engine returns a scored list with references, called *hits*; for instance, the list page of Lycos displays the first lines of web pages relevant to the query. Each reference (item in the list) contains a click-able string which, when clicked by the user, loads the respective web page. Web pages are created with HTML (*Hypertext Markup Language*).

Assume our searching softbot receives the following hits after putting the (generic) query $Q = search\_query$ to some standard search engines. Note that we only allow for atomic (logical) queries; atomic or complex *search* queries such as $search\_query_1$ & ... & $search\_query_n$ are handled differently, by stating integrity constraints corresponding to a (possibly singleton) conjunction of search queries (see, for instance, the *musts* conditions in Section 4). Table 1 shows the result of the search process as an abductive logic program. We assume that the HTML document retrieved by a search engine is transformed to a definite program. Observe that the (two) links contained in the second hit (Rule (3)) have been activated since their contents (keywords) and URL (*Uniform Resource Locator*) are in the program.

Let $\mathcal{A}$ be those predicates that have no definition in P, that is, $\mathcal{A} = \{key\_words, url\}$.

[1] We use the property *acyclic* as introduced in [1], as a subclass of locally stratified programs.

Table 1: Sample search result to a query.

```
(1) search_query :- hit.
(2) hit :- key_words([q, k11, k12, k13]), url(u1).
(3) hit :- key_words([q, k21, k22, k23]), url(u2), link(u2).
(4) hit :- key_words([q, k31, k32, k33]), url(u3).
(5) link(u2) :- key_words([q, k211, k212, k213]), url(u21).
(6) link(u2) :- key_words([q, k221, k222, k223]), url(u22).
```

The argument of the predicate *key_words* is a list of keywords occurring on a web page, including the search query q. In the simplest case, the list is generated by scanning HTML documents for nouns. Associated with each page is a unique internet address (predicate *url*). The predicate *link* records hypertext links referring to other web pages. To discriminate links found by standard search engines from links to be activated by users, the former are called 'hits' while the latter are uniformely called 'links'. As indicated in the Introduction, abductive solutions to a search query are tuples of an address, (a list of) keywords, and typically (a list of) links. This conception of web pages is sufficient for our present purposes; it is similar to LogicWeb modules [16] (but do not contain Prolog code).

A naive searching softbot will start with the retrieval of all web pages that are found by available search engines and then follow all hyperlinks (references to other web pages) of each retrieved page. This agent will meet at least two obstacles: the *first* is a consequence of the sheer size of the web. Since web pages typically feature many hyperlinks to other pages, the process of loading these pages will be very time-consuming; *second*, not all references in web pages are related to the query the page was originally retrieved or references are not available any more.

In our paper, the first problem will be addressed by an approximation technique, that is, we approximate the set of abductive solutions rather than generating them all. Thereby only relevant (in the sense of reachable) solutions will be created. The second problem is essentially one of factual relevance: the problem of preserving interesting information (relative to user interests) over activated hypertext links.

3.3 Bounded SLDA

In order to approximate the abductive process SLDA, we employ consecutively bounded depth-first search [26]. The algorithm performs exhaustive depth-first search for increasing depth bounds $1, 2, ..., m$. The following idea does not depend on this strategy; for instance, Lieberman [14] and Davison and Loke [5] employ a bounded breadth-first search strategy instead. SLDA (SLD-resolution plus abduction inference rule) constructs a refutation $\langle Q^0, \Delta^0\rangle, ..., \langle Q^i, \Delta^i\rangle, ..., \langle Q^n, \Delta^n\rangle$ where $Q^0 = Q$, $\Delta^0 = \emptyset$, $G^n = \Box$ and Δ^n is an abductive solution, that is, a conjunction of abducible predicates.

Definition 3.3 (search depth)
The *depth* d of a literal L is the number of (or-)nodes on the path between the root node of the search tree (the original query) and the node that contains L. The depth of the root

node of the tree is set to 0. Suppose $A :- B_1, ..., B_n$ is a definite clause. If the depth of atom A is m, then the depth of each B_i is $m + 1$.

Let Q^i be a conjunction of atoms $L_1, ..., L_j, ..., L_k$ and m the depth bound. Suppose $d(Q^i) < m$. One atom L_j is selected, for instance, the left-most in Prolog, and one of two steps is performed: a *resolution step* or an *abduction step*.

- **Resolution step.** The atom L_j is resolved with one of the clauses in P. Note that since P and Q are ground, we do not have to consider the case where L_j is resolved with a non-ground fact in Δ^i. Suppose $A :- B_1, ..., B_u$ is the selected clause. In Prolog, this is the top-most clause where L_j and A match. Then
  - $Q^{i+1} = L_1, ..., B_1, ..., B_u, ..., L_k$
  - $\Delta^{i+1} = \Delta^i$
- **Abduction step.** If L_j is abducible, then
  - $Q^{i+1} = L_1, ..., L_{j-1}, L_{j+1}, ..., L_k$
  - $\Delta^{i+1} = \Delta^i \cup \{L_j\}$

If the depth bound is met, that is, $d(L_j) \geq m$, the procedure halts and reports the abductive solutions constructed so far. In Fig. 1, the procedure is illustrated graphically. Numbered arcs indicate resolution steps (compare to the numbers of the rules in Table 1), selected literals are underlined.

Assume the depth of the query *search_query* (Table 1) is set 0. We fix 2 as the depth bound. Then we obtain the following abductive solutions (where $I = \emptyset$).

$\Delta_1 = \{$ key_words([q, k11, k12, k13]), url(u1) $\}$

$\Delta_2 = \{$ key_words([q, k21, k22, k23]), url(u2) $\}$

$\Delta_3 = \{$ key_words([q, k31, k32, k33]), url(u3) $\}$

By way of example, we show how Δ_1 is generated. The given query *search_query* is resolved with Rule (1) whereby we obtain the goal *hit*. The new goal can be resolved with one of the Rules (2), (3) or (4). In Prolog, it is resolved with the first one occurring in the knowledge base, that is, Rule (2). As a result, we have the new goal $key\_words([q, k11, k12, k13]), url(u1)$. The first literal, $key\_words([q, k11, k12, k13])$, is selected but cannot be resolved with a rule in the knowledge base. Since *key_words* is abducible, $key\_words([q, k11, k12, k13])$ is solved through an abduction step by adding it to Δ. The remaining literal $url(u1)$ is abduced by analogous considerations. Now, all goals are solved and $\Delta = \{key\_words([q, k11, k12, k13]), url(u1)\}$.

By increasing the depth bound by 1, that is, to 3, all abductive solutions are produced (compare to Fig. 1).

$\Delta_1 = \{$ key_words([q, k11, k12, k13]), url(u1) $\}$

$\Delta_2 = \{$ key_words([q, k21, k22, k23]), url(u2),
key_words([q, k211, k212, k213]), url(u21) $\}$

$\Delta_3 = \{$ key_words([q, k21, k22, k23]), url(u2),
key_words([q, k221, k222, k223]), url(u22) $\}$

$\Delta_4 = \{$ key_words([q, k31, k32, k33]), url(u3) $\}$

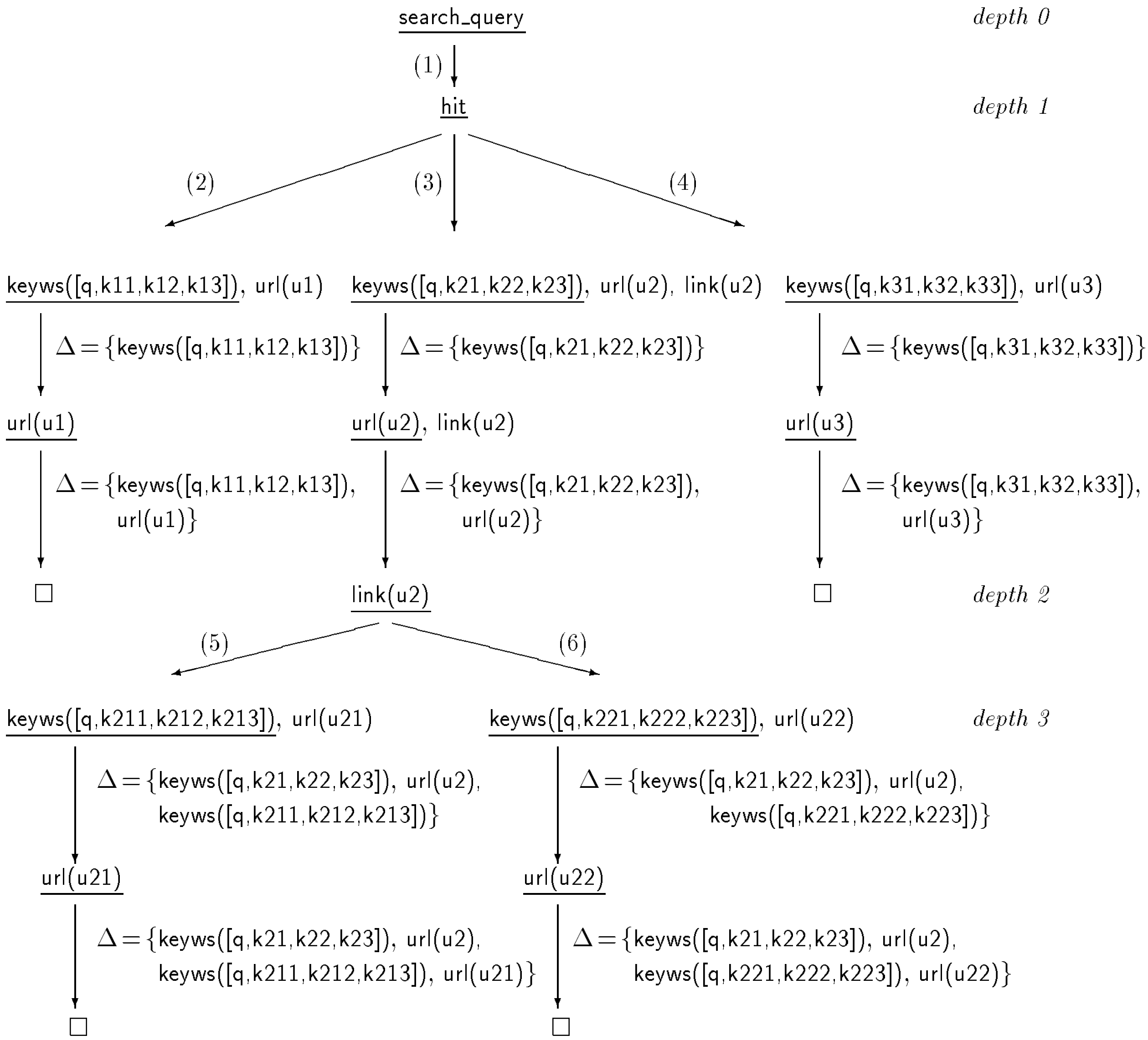

Figure 1: Proof tree of the searching problem for query $Q = search\_query$.

The depth-bounded extension of SLDA will be called SLDbA. The following result is immediate since SLDbA restricts the search space to a subset of SLDA's search space.

Theorem 3.1 (soundness)
If Δ is the result of a SLDbA refutation, then Δ is also the result of a SLDA refutation.

Of course, the opposite direction (completeness) is not valid.

4 Selection among Abductive Solutions

If the depth bound is not already met, the searching softbot will follow all hypertext links occurring in the web pages (that are retrieved by standard search engines like InfoSeek or Lycos). In this way a new list of pages is created that contains further links to yet other web pages. In the *logical* sense, the generated tree can be characterized as *solution-rich* since all nodes are abductive solutions to the search query. As activating hypertext links is a time consuming process and may lead to pages irrelevant to the user, we have to apply *factual* selection strategies. We proceed as follows. First, all abductive solutions at the first level are generated (depth 2 in the example above). At this level, we only expand the best-scored nodes; recall that scoring is a service provided by standard search engines. In addition, the selection can be refined by associating *costs* with hypertext links, by comparing the internet domains of client and server. Then, links (to other web pages) in the selected pages are activated whereby notions of factual irrelevance help to prune the search tree. In this section, we introduce types of factual irrelevance and show how they can be integrated within the abductive framework. We distinguish two kinds of factual irrelevance: *user-specified* irrelevance and *domain-specific* irrelevance.

The *user* may specify

- **Musts.** A web page is irrelevant if it does not contain certain keywords. In the first place, the page *must* contain the query (the search string entered by the user); moreover, the user may specify further notions that have to appear on every retrieved page.

  Example: the URL of a link must have an "html" extension (indicating a text file), so that graphic files "gif", "jpg", and "jpeg" are ruled out as irrelevant.

- **No-nos.** A web page is irrelevant if it contains certain keywords. If the user does not want a notion or phrase to be contained on any page, it is called a *no-no*.

  Example: if *agents* is the query we might want to rule out all pages that contain the word "travel"; thereby we suggest our interest in (intelligent) agents as opposed to travel agents.

On the other hand, certain phrases can be pre-determined as instances of *domain-specific* irrelevance. Examples include the phrase "URL not found" occurring on a page as well as outdated pages.

Integrity Constraints. As mentioned above, we encode factual irrelevance by means of integrity constraints: $I = I_m \cup I_{no}$ where I_m are musts conditions and I_{no} are no-no conditions (including cases of domain-specific irrelevance).

- **Must constraints.** I_m is of format

  $$p([..., t_1, ...]) \wedge ... \wedge p([...t_n...])$$

  such that p is an abducible predicate (here *key_words*[2]) and for each k $(1 \leq k \leq n)$

  $$\Delta_i \vdash p([..., t_k, ...])$$

  where $\vdash$ is the deducibility relation of SLD-resolution; for convenience, we use $p([..., t, ...])$ to denote that term t occurs somewhere in the list argument of p.

- **No-no constraints.** I_{no} is of format

  $$\neg(p([..., t_1, ...]) \vee ... \vee p([..., t_m, ...]))$$

[2]Constraints for URLs are defined analoguously and typically restrict certain parts of the internet address.

such that p is abducible and for each l,

$$\Delta_i \cup \neg p([..., t_l, ...]) \text{ is consistent.}$$

For instance, let the bound be 3, q and $k33$ musts, and $I_{no} = \emptyset$ (compare to Fig. 1); then Δ_4 is the only abductive solution. If $k22$ is a no-no (bound is 3, q a must), then Δ_1 and Δ_4 remain as solutions.

Example 4.1
As a concrete example, assume the user is interested in text documents related to intelligent agents for the world-wide web, but not in robotic agents. We have

$$\begin{aligned} I_m &= \{\mathsf{key\_words}([..., \text{"intelligent agents"}, ...]) \wedge \\ &\quad \mathsf{key\_words}([..., \mathsf{web}, ...]) \wedge \mathsf{url}(\mathsf{*.html})\} \\ I_{no} &= \{\mathsf{key\_words}([..., \mathsf{robotics}, ...])\} \end{aligned}$$

We should stress that all links of a web page are skipped if this page does not satisfy the integrity constraints. Put differently, we assume that pages associated with links occurring on some irrelevant (uninteresting) page are also irrelevant. This assumption will turn out as too restrictive (compare to the discussion of LogicWeb [5] in Section 6). Suppose we are interested in papers related to intelligent agents. If both *intelligent_agents* and *papers* are specified as musts, no page will be retrieved that does not contain both phrases. More likely, we will find papers describing intelligent agents in links that start from a web site in which only *agents* occurs as a keyword, usually in departmental research pages or homepages of individual researchers.

In order to deal with those cases we introduce

- **Conditional constraints.** I_c is of format

$$p([..., t_{1,1}, ...]) \vee ... \vee p([..., t_{1,k_1}, ...]) \rightarrow p([..., t_{1,l_1}, ...]) \quad ...$$
$$p([..., t_{n,1}, ...]) \vee ... \vee p([..., t_{n,k_n}, ...]) \rightarrow p([..., t_{n,l_n}, ...])$$

such that p is abducible, and for each j $(1 \leq j \leq n)$

$$\Delta_i \vdash p([..., t_{j,1}, ...]) \vee ... \vee p([..., t_{j,k_j}, ...]) \rightarrow p([..., t_{j,l_j}, ...])$$

Conditional constraints have to be provided by the designer of the searching softbot. If the antecedent of a conditional constraint is not satisfied, the conditional is (vacuously) satisfied; on the other hand, when the antecedent *is* satisfied, the respective keyword occurring in the conclusion must be contained in the web page. Non-vacuous satisfaction of a conditional constraint is reported to the system, resulting in a higher score for the abductive solution. We intend to incorporate a function $InterestingLink : CurrentLink \times Constraint \rightarrow [0, 1]$ to the system where $CurrentLink$ is the score of the hyperlink and $Constraint$ encodes (conditional) constraint applications. The value of $InterestingLink$ is the new score of the page. In this way we prevent the system to retrieve too many pages without a handle on how interesting they are for the user.

EXAMPLE 4.2
Suppose the user indicates a general interest in intelligent agents and a special interest in papers related to intelligent agents. The system maps the user's queries into the following constraints:

$$\begin{aligned} I_m &= \{\mathsf{key\_words}([..., \text{“intelligent agents”}, ...])\} \\ I_c &= \{\mathsf{key\_words}([..., \mathsf{research}, ...]) \vee \mathsf{key\_words}([..., \mathsf{homepage}, ...]) \\ &\quad \rightarrow \mathsf{key\_words}([..., \mathsf{papers}, ...])\} \end{aligned}$$

Another use of the pre-defined constraints is to map the (general) query together with the respective disjuncts of the antecedent (of the conditional constraint) into musts conditions. For instance:

$$\begin{aligned} I_m &= \{\mathsf{key\_words}([..., \text{“intelligent agents”}, ...]) \\ &\quad \wedge \mathsf{key\_words}([..., \mathsf{homepage}, ...])\} \\ I_c &= \{\mathsf{key\_words}([..., \mathsf{research}, ...]) \vee \mathsf{key\_words}([..., \mathsf{homepage}, ...]) \\ &\quad \rightarrow \mathsf{key\_words}([..., \mathsf{papers}, ...])\} \end{aligned}$$

This approach assumes that, for instance, homepages and research pages are more easily identifiable by search engines than pages containing *papers*.

Advanced methods for analyzing HTML documents. The search process discussed in this paper conceives of web pages as a list of keywords $[k_1, ..., k_n]$ where a keyword is a sequence of letters, delimited by non-letters, and all common English words are deleted. Work on *information retrieval* [22, 18] develops methods that allow for a more fine-grained analysis of text documents: for instance, *stemming* reflects the relation between “agent” and “agents” by finding the root forms of words; the *term-frequency* $TF(w, d)$ (count of times a word w occurs in a document d) indicates how relevant a document is for a certain keyword. In our system, keywords in web pages are treated simply as Boolean features (a word is present or absent). Many techniques of information retrieval are employed by machine learning approaches to identifying interesting web pages (see Pazzani *et al.* [19] and Joachims *et al.* [9]).

5 Implementation

We have implemented a prototype of our agent in Prolog; the core program is about sixty lines of code. The approximate abductive procedure can be readily encoded by a Prolog meta-interpreter [25]. The problem of (possibly) cyclic programs is solved by a simple book-keeping policy. If some predicate $p(t)$ is abduced, this is recorded by asserting the fact $is\_abduced(p(t))$ to the knowledge base. In case this fact is to be abduced *again*, the procedure fails and prunes the remaining computation branch. Moreover, the policy guarantees that all abductive solutions are generated only once; consider the case where different web pages set hyperlinks to the same web page. Checking the integrity constraints amounts to showing that a certain word or phrase is or is not an element of the list argument of the *key_words* predicate.

We are currently looking for a paradigm to integrate the searching agent with the world-wide web. One candidate is the LogicWeb approach of Loke and Davison [16]. Moreover, Cabeza *et al.* [4] describe a rich internet/www programming library for (constraint) logic programming systems, called PiLLoW, that can be used for various web-related tasks. We have to concede that our searching agent is a rather theoretical entity, as is stands.

6 Discussion and Future Work

In this paper, we propose approximate (relevant) reasoning within an abductive framework as the methodology to design a searching softbot. Both the method of approximate reasoning and the abductive proof procedure are readily implemented in Prolog. Although we did not prove this yet, we have strong evidence that reasoning this way is tractable. A distinguished feature of our intended application domain, the web, is that the knowledge base cannot given in advance because it is too expensive to load all hypertext links occurring on web pages. Hence we opted for incrementally building the knowledge base, thereby keeping track of relevant clauses (relative to user queries).

Machine learning. When applied to searching problems in the web domain, the success of our approach will heavily rely on sophisticated strategies for eliminating abductive solutions. The crucial point is to select most promising links on web pages. Here, investigations from the *learning* community could give valuable insights:

- *WebWatcher.* Amstrong *et al.* [2] suggest to compute a function $LinkUtility : Page \times Goal \times User \times Link \to [0,1]$ such that $Page$ is the current web page, $Goal$ is the search query, $User$ is the model of the user, and $Link$ is a hypertext link occurring on the page. The value of $LinkUtility$ is the probability that activating the $Link$ on $Page$ is a step of the shortest path to a page satisfying $Goal$, relative to $User$. The system learns by observing users' successful paths from a given web page to some goal page.

- *Letizia.* The user interface agent of Lieberman [14] is similar to WebWatcher in the sense that it makes suggestions to the user what hyperlinks to activate. The agent may justify its recommendations by observing the user's past browsing behavior; for instance, bookmarking a page counts as a strong indicator of interest.

- *Syskill & Webert.* Pazzani *et al.* [19] describe an agent that may rate web pages based on a user profile. The profile is learned by analyzing information on pages that the user evaluates as either 'hot', 'lukewarm', or 'cold'. The profile can also be used to generate a Lycos query that encodes the user's interests.

The solutions generated by our system could be annotated with the value of $LinkUtility$ (WebWatcher) or with the estimated probability that a user would rate a page as 'hot' (Syskill & Webert). We leave the possibility to recast our framework in probabilistic terms (Poole [20]) for future research.

Web search and pages augmented by Prolog code. The LogicWeb system [5] views the web as a collection of pages connected by hypertext links. In this abstract sense their conception is very similar to our view of the web. Yet, web pages in LogicWeb (called *modules*) can contain further information in the form of Prolog code that enables more sophisticated forms of search. For instance, take the example where a user wants to find papers relevant to agents. The problem is how to handle relatively uninteresting pages (only containing *agents*) when there is some chance that they lead to interesting pages (that contain both *agents* and *papers*).[3] Here LogicWeb exploits the fact that certain *classes* of Web pages such as department web sites typically exhibit a standardized page

[3] I am indebted to Seng Loke for clarification on this point.

type hierarchy that may guide the search process. The class of department web pages is defined by means of a predicate *composed_of* that has two page types as arguments.

composed_of(dept, research)

composed_of(research, project

composed_of(project, proj_members)

composed_of(proj_members, researcher)

If the LogicWeb system arrives at an agent-related research page, it explores the hierarchy to look for a page type that contains the papers, utilizing depth-bounded breadth-first search. Although we do not have the concept of a page class in our system, we may formulate (standardized) integrity constraints that encode knowledge about likely places where, for instance, papers are found. We consider to formulate conditions that mirror page type hierarchies on the level of integrity constraints.

Web query languages. The *Information Manifold* (**IM**) of Kirk *et al.* [11] is intended to access multiple information sources on the web. The query processor of **IM** guarantees that only those sources are visited that are relevant to the query. By utilizing integrity constraints in the query modus, the system can identify relevant knowledge bases by their description only and does not have to actually access them. However, it seems to us that most of the advantages of **IM** rely on the fact that information sources are structured. This is quite different to querying the web that is highly unstructured. The integrated active/deductive/object-oriented data model (ADOOD) of Gianotti and Manco [7] seems more suitable to handle poorly structured information.

Visualization. The activity of the softbot could be monitored by the Hy^+ visualization system of Hasan *et al.* [8].

Abductive procedures for normal abductive programs. The representation language employed in this paper is propositional and Horn. We argue that this language fragment is sufficient for basic searching tasks. On the other hand, if the software agent has to solve tasks such as "print all available papers related to intelligent agents", the expressiveness of the language has to be extended in order to allow *planing* a sequence of actions (first find the papers, then download, and so on). It should be stressed that planning can be performed within the abductive framework for the Horn subset of first-order logic augmented by negation-as-failure (Denecker and co-workers [6, 17]). An *approximate* version of an abductive reasoner within the extended language is discussed in [21].

Acknowledgements

The author is indebted to the anonymous referees for their stimulating and guiding comments. The author is supported by the exchange program Salzburg/Austria-Irvine/USA, and by grants from the *Österreichische Forschungsgemeinschaft* and the *Stiftungs- und Förderungsgesellschaft*, Salzburg.

References

[1] K.R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.

[2] R. Armstrong, D. Freitag, Th. Joachims, and T. Mitchell. WebWatcher: A learning apprentice for the World Wide Web. In *Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogenous, Distributed Resources*, 1995.

[3] St. Brüning and T. Schaub. Using classical theorem-proving techniques for approximate reasoning: A revised report. In B. Bouchon-Meunier, R. Yager, and L. Zadeh, editors, *Advances in Intelligent Computing*, pages 389–398. Springer, 1995.

[4] D. Cabeza, M. Hermenegildo, and S. Varma. The PiLLoW/CIAO library for INTERNET/WWW programming using computational logic systems. *1st Workshop on Logic Programming Tools for INTERNET Applications*, in conjunction with *JICSLP-96*, 1996.

[5] A. Davison and S.W. Loke. LogicWeb: Enhancing the web with logic programming. Submitted to the Journal of Logic Programming, 1996.

[6] M. Denecker and D. De Schreye. SLDNFA: An abductive procedure for normal abductive programs. *Proceedings International Joint Conference and Symposium on Logic Programming*, 1992.

[7] F. Giannotti and G. Manco. A deductive data model for representing and querying semistructured data. In *Proceedings 2nd International Workshop on Logic Programming Tools for Internet Applications*, in conjunction with *ICLP-97*, 1997.

[8] M.Z. Hasan, A.O. Mendelzon, and D. Vista. Visual web surfing with Hy$^+$. In *Proceedings of the CASCON-95, Toronto*, 1995.

[9] T. Joachims, D. Freitag, and T. Mitchell. WebWatcher: A tour guide for the world wide web. Technical Report, CMU-CS-96, School of Computer Science, Carnegie Mellon University, 1996.

[10] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[11] Th. Kirk, A.Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogenous, Distributed Resources*, 1995.

[12] H.J. Levesque. A knowledge-level account of abduction. In *IJCAI-89*, pages 1061–1067, 1989.

[13] A.Y. Levy and Y. Sagiv. Exploiting irrelevance reasoning to guide problem solving. In *Proceedings Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 138–144, 1993.

[14] H. Lieberman. Letizia: An agent that assists web browsing. In *Proceedings International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[15] J.W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, New York, second, extended edition, 1987.

[16] S.W. Loke and A. Davison. Logic programming with the world-wide web. In *Proceedings of the 7th ACM Conference on Hypertext (available at http://www.cs.unc.edu/barman/HT96/P14/lpwww.html)*, pages 235–245, 1996.

[17] L. Missiaen, M. Bruynooghe, and M. Denecker. CHICA, an abductive planning system based on event calculus. *Journal of Logic and Computation*, 5(5), 1995.

[18] M. Pazzani. Machine learning and intelligent information access. Handout for ICS 279, University of California, Irvine, Fall 1996.

[19] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting web sites. In *Proceedings AAAI-96*, 1996.

[20] D. Poole. Logic programming, abduction and probability. *Proceedings International Conference on Fifth Generation Computer Systems (FGCS-92)*, pages 530–538, 1992.

[21] H. Prendinger. Approximate abductive reasoning. Submitted to *Conceptus*, 1997.

[22] G. Salton. Developments in automatic text retrieval. *Science*, 253:947–979, 1991.

[23] M. Schaerf and M. Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.

[24] E. Selberg and O. Etzioni. Multi-service search and comparision using the MetaCrawler. In *Proceedings of the 1995 World Wide Web Conference*, 1995.

[25] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press, 1994.

[26] M.E. Stickel and W.M. Tyson. An analysis of consecutively bounded depth-first search with applications in automated deduction. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1073–1075, 1985.

[27] D. Subramanian and M.R. Genesereth. The relevance of irrelevance. In *Proceedings International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 416–422, 1987.