Appears in *Proc. of PACLP'2000, Manchester, U.K., April 10-12, 2000*

Virtual Answers for Query Refinement in Information Retrieval

Jean-Marc Andreoli [1] and Uwe M. Borghoff [2]
[1] Xerox Research Centre Europe, Grenoble, France
[2] Institut für Softwaretechnologie, Universität der Bundeswehr, München, Germany.

Abstract

Query refinement in information retrieval over repositories on the Web faces problems when trying to adapt a purist database approach. Needed semantics and brokering facilities differ significantly when compared, for instance, with traditional SQL-like query schemes. Therefore, we propose an interaction model, adapted to Web sources, and based on constraints and on the notion of so-called "virtual answers". Our model extends both the basic pull and push models, but proposes virtual answers as a powerful means for further interactions from a given answer: typically, query refinement. A major contribution of this paper is the discussion of the issues of information combination (brokering) and of soundness and completeness of information retrieval engines (semantics) in the framework of this new model. We make the simplifying assumption that sources evolve monotonically. Strictly speaking, this may look irrealistic in the context of the Web, but it approximates many real situations, where changes in the sources occur on a larger time-scale than user sessions, or where clients check the validity of returned items before actually making use of them.

Currently, the model is being evaluated for integration into the Constraint Based Knowledge Broker system (CBKB[1]). The CBKB system aims at heterogeneous information retrieval, schema integration, and knowledge fusion.

1 Introduction

The huge amount of data nowadays available online on many desktops has lead to the need for efficient information retrieval models and tools to support the user's permanent hunt for the right information at the right time and the right place. A major source of inspiration to deal with such problems comes from the database research community, which has acquired over time a rich experience in efficient information retrieval techniques from possibly very large repositories. However, many of the assumptions on which such techniques have been developed do not hold in the case where the repository is the Web. In other words, the Web cannot, in many respects, be treated as a huge (distributed!) database from which users could extract information using traditional query mechanisms and languages such as SQL.
In particular, one major characteristic of the Web is its extraordinary heterogeneity. Only the lowest level communication protocol (HTTP) has been homogenized (and this is in fact a major reason for its success). Concerning content, heterogeneity is still the rule, and will remain so in the foreseeable future. In fact, it concerns two different aspects: the format of the exchanged data on the one hand, and the model of interaction with the sources on the other hand.

- Concerning data format, most information sources on the Web give access to and present their data through HTML pages and forms, in a human readable format. HTML is indeed directly interpreted by most Web browsers, and thus provides the basis for cheap and portable human interfaces. The HTML pages produced by one given source generally have a uniform structure, since they are generated by the same program or set of programs. Consequently, once this structure is known for one source, it is possible to write specific "wrappers" for that source, i.e., mediator programs capable of (*i*) translating queries in a uniform abstract query language into the specific HTML forms required by the source, and, conversely, (*ii*) translating the returned HTML pages in the specific format of the source into knowledge items in a uniform abstract knowledge representation language. In other words, data format heterogeneity can generally be dealt with by building sets of wrappers for the various data sources, encapsulating them in a uniform interface supporting an abstract data manipulation language.

[1] `http://www.xrce.xerox.com/research/ct/projects/cbkb`

The task is possible, but not necessarily easy and several issues have to be tackled, in particular: minimizing the redundancies in the design of wrappers (when they are to be mass produced), maintaining them in presence of changes in the data source interfaces (data sources generally do not incorporate mechanisms to notify changes, esp. minor changes), and, most importantly, dealing with extended features incorporated into HTML pages, such as applets, scripts etc. See [8, 3, 7] for an overview of these problems, which will not be tackled here.

- But information sources on the Web present another kind of heterogeneity which makes the Web even more distant from the database world. It concerns the interaction protocol between a client and a source. In the database world, this protocol is generally limited to the basic "pull" model, in which the client submits a query (say, an SQL query) and receives an answer in return, consisting of a list of items matching the query. (In this paper, we are only concerned with interactions which extract information, and ignore those with side-effects aiming at modifying, deleting or adding items in a database.)

  On the Web, the situation is quite different. Of course, the basic (HTTP) protocol directly supports the pull model. But most sources use it to implement more sophisticated models of interaction. For example, in a highly unpredictable environment like the Web, sources may have to deal both with very specific requests returning only a few answers and with very general requests returning a huge number of answers. To deal with such disparity, many sources define a threshold on the amount of information allowed to be returned for one request. If the answers to a request, as computed by the source, contain less information than the threshold, they are all passed to the client. If the computed answers exceed the threshold, only some (or a summary) of them, which remain within the threshold, are passed to the client, together with a warning that more answers exist, and possibly a hook to access the remaining ones by refining the query. This leads to a more complex interaction model where the exploration of the space of answers is based on a collaboration between the system and the user, where the system recursively suggests query refinements and the user decides which ones to follow. Another example of non-standard interaction models developed on top of the basic HTTP protocol concerns sources which evolve in time (we assume here a purely monotonic evolution). For such sources, the "pull" model, where the client first specifies the items s/he is interested in and expects in return the list of matching items at the moment of the query, is favorably replaced by the "push" model where the matching items are sent to the client over a long (possibly infinite) period of time, according to the evolution of the source, and at the source's initiative.

Heterogeneity in interaction models and formats makes database concepts unsuitable, without adaptations, to the Web. We are interested here in two particular aspects:

Semantics: In a database context, an information retrieval process may be measured with respect to two general properties which characterize the difference (if any) between the actual behavior of the process and its intended behavior:

- *Soundness:* the process is sound if any item in the list of answers returned for a query actually corresponds to a database item which satisfies the query.
- *Completeness:* the process is complete if any item in the database which satisfies a query is always included in the list of answers returned for that query.

As defined here, these notions are deeply rooted in the pull model, which, as explained before, is not really suited to Web information retrieval. Adaptations of these notions are needed for extended interaction models.

It is often argued that Web information retrieval processes need not be complete nor even sound, but simply "helpful". This simplification, while very pragmatic, makes it rather difficult to measure and compare information retrieval processes and is mainly justified by the lack of a general model to express and test the soundness and completeness properties. This paper is an attempt to provide such a model.

Brokering: Brokers are information sources which do not contain explicit knowledge items, but only a mechanism deriving new "compound" knowledge items from existing ones, which may have to be retrieved from other sources [4, 2]. Hence, brokers appear alternatively as client and server in the retrieval process. In this sense, the so called meta-search engines are a form of brokers.

In the database context, brokers typically implement "join" operations. Various approaches have been proposed to specify and implement joins, for example as algebraic operations on tuple sets or as logical operations a la Datalog on predicates [10]. The specification of a join generally leads to "naive" implementations (nested loops in the algebraic case, deduction — forward and/or backward chaining — in the logical case) which have been refined to achieve efficiency, for example by taking into account index information.

On the Web, both specification and implementation of information combination performed by brokers have to be adapted to extended interaction models. One solution, of course, consists of devising ad-hoc combination mechanisms for specific sets of sources, but such specific brokers would hardly be re-usable outside the context where they have been designed. The model proposed in this paper allows the definition of generic combination mechanisms.

Section 2 presents an abstract interaction model adapted to information sources on the Web, and based on the notion of "virtual" answers. It encompasses both the basic pull and push models, but proposes virtual answers as an abstraction for various mechanisms allowing further interactions from a given answer: typically, query refinement. Many interaction models of heterogeneous sources on the Web can be viewed as instances of this abstract model. Section 3 discusses the issues of information combination (brokering) and of soundness and completeness of information retrieval engines (semantics) in the framework of this new model.

2 The Virtual Answer Interaction Model

2.1 Hypotheses

Information retrieval is the activity of making explicit a set of knowledge items matching a client's requirements. The knowledge items are implicitly contained in distributed information sources over a wide area network such as the Web. The matching procedure between knowledge items and user requirements may be:

- *Binary:* in this case, an item either matches the requirements or it does not, and the client is interested in those items which match and only these ones.
- *Fuzzy:* in this case, any item matches all requirements, but at different degrees, and the client is interested in "best" matches, i.e. items which match most closely the requirements.

None of these two classes subsumes the other, and they correspond in fact to rather different kinds of information retrieval activities. We are mainly interested here in the first class (binary matching), which is closer to the traditional database world, although our approach may apply to the second class. Each information source has a double characterization.

Descriptive characterization: An information source holds a set of publicly accessible knowledge items, called the extension of the source. No assumption is made on the detailed nature of these items (tuples, records, objects, etc.), but we assume that their meaning is un-ambiguous for the user and can be represented in a uniform language. We assume that the extension of a source, if not static, evolves monotonically (i.e., the extension may grow in time, but never shrink). In fact, non monotonic sources can also be accommodated, as long as users do not mind about stale items (or check validity before using them), or changes in the source occur on a larger time scale than user sessions.

Interactive characterization: The extension of an information source is not directly visible from the outside world but is encapsulated within an interface, accessible from the network, and defining an interaction model for the extraction of the knowledge items of the source. We ignore issues of access control or access costs in the interaction model. Access operations are supposed to be side-effect free: in particular, we ignore, in this paper, destructive operations on knowledge items, which would contradict the previous hypothesis of monotonicity of the sources.

While the descriptive aspect of a source can straightforwardly be formalized as a set (the extension of the source), the formalization of the interactive aspect is not so obvious. Given the HTTP infrastructure of the Web, we stick to a client-server approach and describe an interaction model by the valid sequences of messages it allows between a client and a server.

2.2 A Constraint Based Description Language

To deal with the heterogeneity of the sources, captured by the absence of assumptions about the detailed nature of the knowledge items, we make use of a general purpose language $\mathcal{C}$ capable of expressing abstract statements about knowledge items, which are supposed to range over a common abstract domain $\mathcal{D}$. In other words, we make the minimal assumption that there exists a common language, understood by all the sources, capable of expressing properties of the elements of $\mathcal{D}$ but we do not derive this language from a common, fixed representation of the knowledge items themselves. Generic knowledge description languages have already been proposed in the literature and intensively studied, e.g. KL1 [5, 6].

The elements of $\mathcal{C}$ are called constraints. Given a constraint c and an element x of the domain, we write $c(x)$ the statement c applied to x. The denotation of c is the subset $\|c\|$ of the domain defined by

$$\|c\| \quad = \quad \{x \in \mathcal{D} \ / \ c(x)\}$$

An example of domain is the set of "feature structures" [1], on which constraints can be represented as unary formulae in feature logic. For example, the formula

$$\exists t \qquad x : \texttt{book} \ \wedge \ x : \texttt{title} \rightarrow t \ \wedge \ \texttt{contains}(t, \texttt{"Wuthering"})$$

is a constraint identifying bibliography entries of books the title of which contains the word "Wuthering".
A given source s is characterized by its extension Ω_s which is a subset of $\mathcal{D}$. The set of all the valid knowledge items is therefore characterized by the subset Ω of $\mathcal{D}$ defined by

$$\Omega = \bigcup_s \Omega_s$$

A user can then specify a subset of valid items of interest, for enumeration, by a constraint c, characterizing the subset $\Omega \cap \|c\|$. The size of this subset can vary between 0 (e.g. if constraint c is inconsistent) and a huge number: if constraint c is tautological, the specified subset contains the union of the extensions of all the knowledge sources; enumerating such a subset is neither possible, nor tractable, nor even desirable, since the user cannot process such an amount of items.

2.3 A Constraint Based Interaction Model

The "pull" interaction model is the simplest protocol to access an information source. In this interaction model, a valid sequence of messages is reduced to a single invocation-reply interaction: a query, i.e. a constraint c, is passed in the invocation and the whole list of matching items, i.e. $\Omega \cap \|c\|$, is passed in the reply. Figure 1-a illustrates this situation.
The "push" interaction model in which the list of answers may be returned over a long period of time by a source, is obtained by a straightforward extension of the previous protocol. In the push model, a valid sequence of messages is initiated by an invocation followed by an unbounded stream of replies. A query, i.e. a constraint c, is passed in the initial invocation and each reply holds a matching item, i.e. one element of $\Omega \cap \|c\|$. Figure 1-b illustrates this situation.
The "virtual answer" interaction model elaborates on the previous ones. Like the push model, it assumes that answers are returned in a possibly unbounded stream to the client. But it also assumes that the client is not passive in the enumeration of this stream. The simplest way to achieve this is by returning to the client a "handle" to the stream of answers, from which individual answers can be separately retrieved, by explicit invocation of a "get-next" operation on the handle, as suggested, for example, in the KQML protocol [9]. Although this allows control by the client of the enumeration rate of the elements of the stream of answers, it does not fundamentally modify the visibility of the content of the stream by the client. For example, for a given query, a source may compute relevant information such as the number of answers or a classification of the answers, without computing them all. This kind of information can be very useful to navigate through the stream of answers in an "intelligent" way, avoiding for example huge chunks of the stream which the client knows are irrelevant to his purpose. The aim of the virtual answer protocol is to allow relevant information about answers which have not been computed explicitly by the source to be returned in a manageable way to the client.
In the virtual answer interaction model, a valid sequence of messages is initiated, as in the push model, by an invocation, i.e. a query specified by a constraint c, followed by an unbounded stream of replies. The only difference with the push model is that each reply now holds either a matching item (called a "plain" answer), or a constraint c' entailing c. Constraint c' is called a "virtual" answer and must be interpreted as an implicit handle to the sub-stream of items matching c' (and hence, by entailment, c). The client may then choose to submit c', or some more specific constraint, to obtain more (and more refined) answers to the original query. Note that:

- Virtual answers co-exist with plain answers and may arise on the stream in any order; order is irrelevant.
- Several virtual answers may be returned, corresponding to different ways to refine the query.

The situation is illustrated on Figure 1-c.
Let's stress here the difference between a plain answer given by an element of the domain a and a virtual answer, the denotation of which contains (or is reduced to) a. The plain answer asserts that element a actually is a valid item, while the virtual answer asserts that the element a may be a valid item, and one way to confirm that is to submit the virtual answer as a refined query, which, then, may (or may not) produce a as a plain answer. Hence, the difference between plain and virtual answers is not just the size of their denotation (the former denotes exactly one item while the later may denote zero, one or more items) but also the nature of the statement they make (the former states a necessity while the later states a possibility).

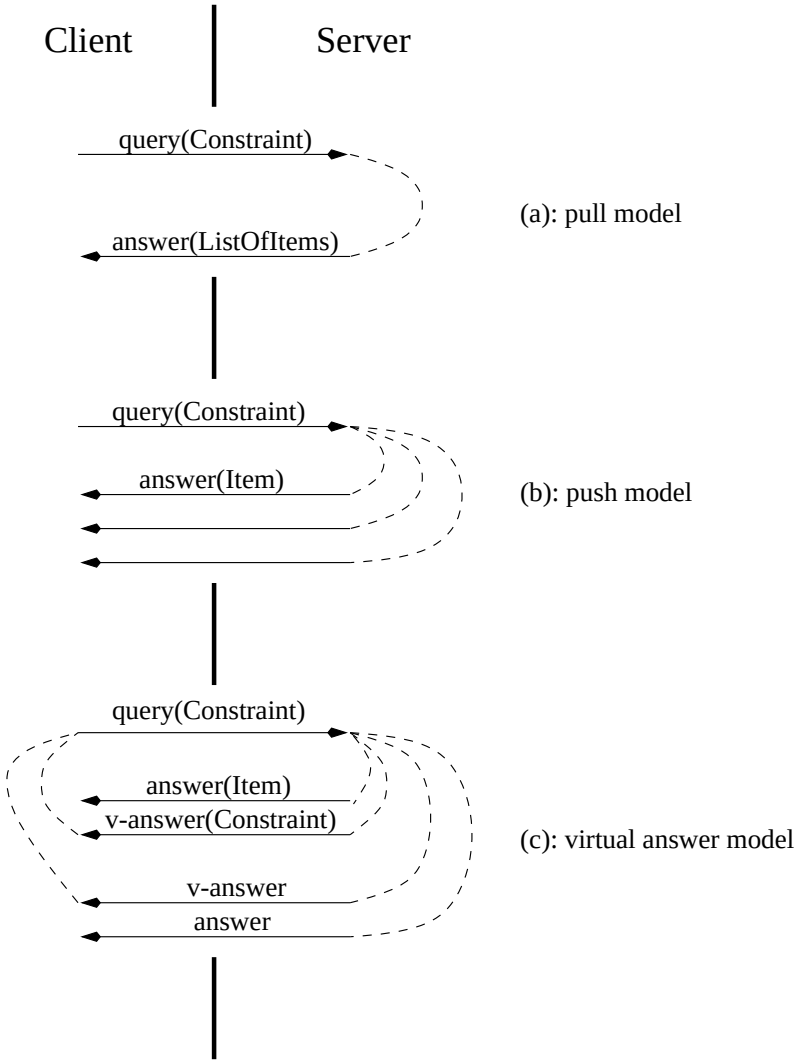

Figure 1: The different interaction models

2.4 Examples

Consider for example a source like Alta-Vista[2]. If a rather "wide-scope" key-word such as "broker" is submitted, the source returns the URLs of the first 20 of more than a million pages containing this key-word. The probability is high that the user is not interested in all these pages, which pertain to stock-exchange brokers as well as mortgage brokers and many other things. But, without computing explicitly all the answers, the Alta-Vista engine is capable of making explicit a couple of refinement proposals for the query.

As depicted in Figure 2, one proposal is "stock broker", another "business brokers", etc. These proposals correspond to virtual answers enriching the original constraint (keyword "broker") with a new set of constraints (the new keywords). By clicking on it, a refined query is launched and the number of answers is reduced to a bit more than ten thousands in the case of "stock broker". If the user is not satisfied yet, a new refinement is then possible and the system proposes, in addition to a new set of URLs (i.e., new plain answers), a new set of key-words (i.e., new virtual answers) to choose from. For example, by clicking on "online stock brokers", the number of answers is reduced to a few dozens. Each time, the decision (and the way) to refine the query is under the user's control, based on information provided by the system.

Another earlier attempt, also in the Alta-Vista source, was the provision of a network of "topics" under which to classify the answers. These topics typically correspond to virtual answers. Figure 3, shows the resulting topic network for the search term "knowledge broker".

Similar to the refinement proposal as given by Alta-Vista, so-called custom search folders have been introduced

[2] `http://www.altavista.com`

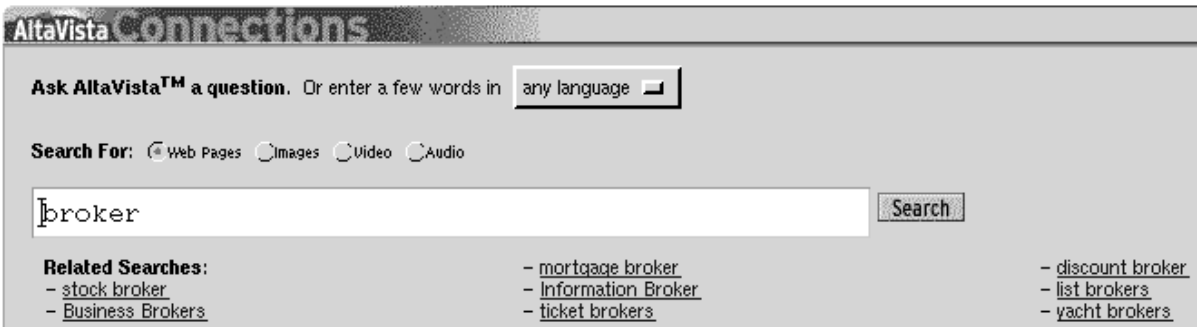

Figure 2: Refinement proposals in Alta-Vista

by the source Northernlight[3]. Again the user is confronted with means to further refine his query. As shown in Figure 4, the answers are packed into a folder scheme that allows to refine the query under the user's control, based on dynamically calculated information. By clicking on a folder, new subfolders open up. More precise information is displayed in the main answer window where a couple of plain answers are always given.
In the same way, many other query refinement mechanisms, available with various information sources with more or less sophistication, can also be captured at the abstract level by the notion of virtual answers. Simpler mechanisms to explore streams of answers can also be captured, such as the most simple one mentioned above, allowing the client to enumerate answers one by one (or by blocks) from a handle. For example, in the simple case of a query returning the first 20 answers together with a "next" button to retrieve more, the next button can be viewed as a virtual answer. It corresponds to the initial constraint augmented with a new constraint of the form "$\texttt{rank}(x) > 20$", so that if the same query is submitted with this new constraint explicitly stated (or infered), the next 20 plain answers (and a new virtual answer corresponding to the new next button) are returned, and so on. Ranks are here arbitrary numbers assigned to items by the source (for a specific user session).
Query refinement based on virtual answers can also be used to handle exceptions in the retrieval process. For example, consider a source of bibliography entries which rejects insufficiently informed queries that do not specify the literary genre (poetry, theater, novel, etc.). If a client submits a query concerning only the title (see the sample feature constraint of section 2.2), virtual answers may be returned with the initial constraint (on the title) augmented with a new constraint of the form, e.g.

$$x : \texttt{genre} \rightarrow \texttt{"novel"}$$

The client may then re-submit a refined query specifying the appropriate genre. Note here that a separate virtual answer is returned for each literary genre, which is reasonable as long as their number is not too big. Now, if the source enforces, say, that the author's name be given, it is clear that the system cannot produce a virtual answer for each possible author name, even if the model allows infinite streams of answers. This case would require a variation of our model where (possibly infinite) sets of constraints, specified by second-order constraints, can be returned in virtual answers. For example, we could have the following second-order constraint, where c is a second order variable standing for a first-order constraint:

$$\exists a \forall x \quad [c(x) \ \supset \ x : \texttt{author} \rightarrow a]$$

In this paper, we do not explore this extension of the model.
Another case of exception which could be treated with virtual answers occurs when an information retrieval agent faces connection problems with a given source. The agent may be programmed to automatically retry connection at regular intervals for some time, but after a while, it may give-up. In this case, it could return a virtual answer, augmenting the initial constraint with a new constraint of the form "$\texttt{date}(x) > T + \Delta$" where T is the date of the last connection attempt and Δ the time span during which the connection should not be retried. Thus, only if this constraint is explicitly stated (or rather, in this case, infered) in a refined query triggered by the client, will a new connection be attempted again.

[3] `http://www.northernlight.com`

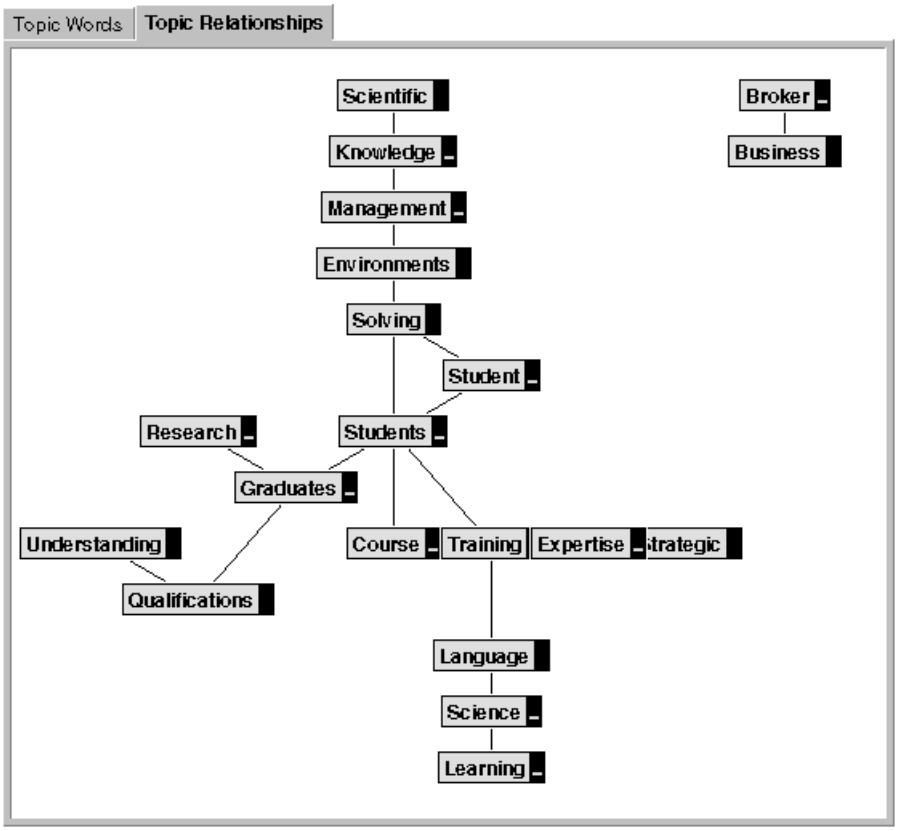

Figure 3: Network of topics in Alta-Vista

3 Utilization of the Virtual Answer Model

The virtual answer interaction model provides an abstraction for dealing with sources supporting long-lived interaction protocols including in particular a notion of "query refinement" in a very wide sense. But of course, it is also capable of dealing with simpler sources, and thus provides a uniform framework in which to model heterogeneous sources on the Web. This is particularly crucial to understand the semantics of an information retrieval engine and to enrich it with brokering capabilities.

3.1 Semantics of Information Retrieval Engines

An information retrieval engine is a mapping

$$\begin{array}{rccccc} E: & \mathcal{C} & \mapsto & \wp(\mathcal{D}) & \times & \wp(\mathcal{C}) \\ & c & & \lfloor E \rfloor(c) & & \lceil E \rceil(c) \end{array}$$

such that

$$\forall c \in \mathcal{C} \quad \lfloor E \rfloor(c) \subset \|c\|$$
$$\forall c, c' \in \mathcal{C} \ \textbf{if}\ c' \in \lceil E \rceil(c)\ \textbf{then}\ \|c'\| \subset \|c\|$$

The first (resp. second) component $\lfloor E \rfloor(c)$ (resp. $\lceil E \rceil(c)$) corresponds the plain (resp. virtual) answers for the query holding constraint c. Both components are sets, which need not be returned in one block but which may be enumerated over time. The conditions imposed in the definition state an intrinsic consistency property of the engine: a plain answer must match the query from which it originates and a virtual answer must at least entail the query from which it originates.
Intuitively, given an information retrieval engine, the extension of a constraint c is the set of all the plain answers returned for c for all the refined queries obtained by recursively following the virtual answers from c. This is formalized by the following definition and properties.

Definition 1 *Let E be a retrieval engine and Γ a set of constraints. Γ is said to be E-stable (denoted $E \wr \Gamma$) iff $\forall d \in \Gamma\ \lceil E \rceil(d) \subset \Gamma$. Let c be a constraint. We define the virtual (resp. plain) E-extension of c, denoted $\lceil E \rceil_\star(c)$*

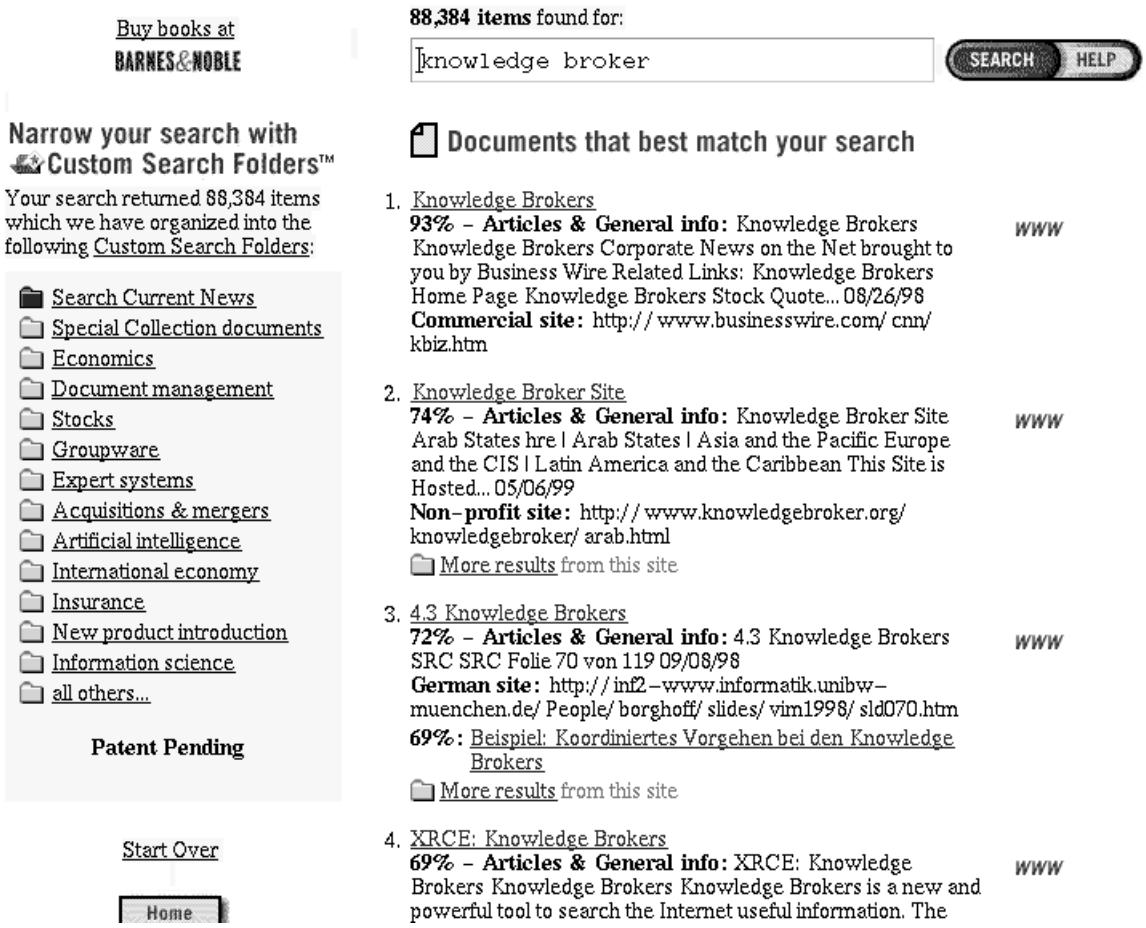

Figure 4: Custom search folders in Northernlight

(resp. $\lfloor E \rfloor_\star(c)$) as follows:

$$\begin{array}{lcl} \lceil E \rceil_\star(c) & = & \bigcap_{\Gamma \,/\, c \in \Gamma \,\wedge\, E \rceil \Gamma} \Gamma \\ \lfloor E \rfloor_\star(c) & = & \bigcup_{d \in \lceil E \rceil_\star(c)} \lfloor E \rfloor(d) \end{array}$$

It is easily provable that the intersection of E-stable sets is E-stable so that the virtual E-extension of a constraint is E-stable (more precisely, it is the smallest E-stable set containing that constraint). From this, it is easy to prove that the plain E-extension of a constraint c is the union of the set of plain answers for c and the union of the plain extensions of the virtual answers for c, i.e.

$$\forall c \in \mathcal{C} \quad \lfloor E \rfloor_\star(c) = \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)} \lfloor E \rfloor_\star(d)$$

These properties, as well as all those stated below, are proved in Annex A. We can now formalize the notions of soundness and completeness:

Definition 2 *Let E be a retrieval engine and let Ω be the set of valid knowledge items.*

- *E is sound w.r.t. Ω iff* $\quad \lfloor E \rfloor_\star(c) \subset \|c\| \cap \Omega$
- *E is complete w.r.t. Ω iff* $\quad \|c\| \cap \Omega \subset \lfloor E \rfloor_\star(c)$

Thus, these definitions formalize in a logical framework the notions of soundness and completeness for information sources accessible through the virtual answer interaction model. To summarize, the main points are:

- Sources content defines a set of valid items as a subset of an abstract domain.
- The extraction of these items is modeled as a mapping returning for each query (constraint), both plain answers (items) and virtual answers (constraints).

- Soundness and completeness characterize the difference between the expected answers for a constraints c (i.e., the set of valid items satisfying the constraint) and the retrieved answers (obtained by merging all the plain answers obtained from c and by recursively following all the virtual answers).

Definitions 2 of the soundness and completeness properties, although close to intuition, are not operational in the sense that they involve the mapping $\lfloor E \rfloor_\star$ defined globally from E. This means that checking such properties for a given engine requires global reasoning about the programs implementing E, which may not be obvious. Under certain conditions (for completeness), the properties can be "localized", making the checks simpler.

Definition 3 *"Local" versions of soundness and completeness:*

- *E is locally-sound iff* $\quad \lfloor E \rfloor(c) \subset \|c\| \cap \Omega$
- *E is locally-complete iff* $\quad \|c\| \cap \Omega \subset \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)} \|d\|$

Local soundness means that the plain answers for a query (instead of the whole extension of the query) are valid items which satisfy this query. Local completeness means that the valid items which satisfy a query either can be found among the plain answers or match some virtual answers for that query (but are not necessarily found in the whole extension of the query).

Proposition 1 *Soundness (resp. completeness) implies local soundness (resp. completeness).*

The converse of the above proposition is true in general for soundness, but not for completeness. Take for example the trivial retrieval engine given by

$$\lfloor E \rfloor(c) = \emptyset \qquad \lceil E \rceil(c) = \{c\}$$

This engine is not very informative: for any query, it replies that answers may exist and propose the same query as its own refinement. By simple application of the definition, this engine turns out to be locally complete. But it is obviously incomplete w.r.t. any (non trivial) set of sources since $\lfloor E \rfloor_\star(c) = \emptyset$ for any c.
However, with some conditions on the engine, local completeness can be proved equivalent to completeness. Basically, the condition states that the refinements proposed by the virtual answers to a query are "true" refinements, which really augment the original query, excluding for example the dumb refinement proposed by the trivial retrieval engine described above. To formalize this notion, we say that a binary relation $\succ$ over constraints is a true refinement if any infinite $\succ$-chain converges to the inconsistency, i.e.

$$\forall c \in \mathcal{C}^\aleph \ \textbf{if} \ \forall n \ \ c_{n+1} \succ c_n \ \textbf{then} \bigcap_n \|c_n\| = \emptyset$$

The following is then true:

Proposition 2 *Let E be a retrieval engine.*

- *E is locally sound if and only if E is sound.*
- *If the relation "$c' \in \lceil E \rceil(c)$" is a true refinement, then E is locally complete if and only if E is complete.*

3.2 Brokering with the virtual answers model

In general, a broker can be modeled as a mapping on subsets of the domain $\mathcal{D}$, of the form

$$\Pi : \wp(\mathcal{D}) \mapsto \wp(\mathcal{D})$$

The problem we are interested in is

> Given a (locally) Ω-sound and complete retrieval engine E, build a (locally) $\Pi(\Omega)$-sound and complete engine $\Pi(E)$.

We want to solve this problem for various classes of brokers Π; more precisely, we are interested in two kinds of brokers, "restriction" brokers and "induction" brokers.

3.2.1 Restriction brokers

A *restriction* broker over domain $\mathcal{D}$ is associated to a subset Δ of $\mathcal{D}$. The restriction broker for Δ is defined by:

$$\Pi_\Delta(\Omega) \quad = \quad \Omega \cap \Delta$$

An obvious way to build the engine $\Pi_\Delta(E)$ for an engine E is by

$$\begin{array}{lcl} \lfloor \Pi_\Delta(E) \rfloor(c) & = & \Pi_\Delta(\lfloor E \rfloor(c)) \\ \lceil \Pi_\Delta(E) \rceil(c) & = & \lceil E \rceil(c) \end{array}$$

This engine behaves exactly as E except that whenever it generates plain answers, it eliminates those which do not belong to Δ. In other words, it applies a "generate and test" strategy, which is the only possible one in absence of further information. However, when Δ has some good properties with respect to the constraints, a better strategy can be applied. Let's assume for instance that there exists a mapping ϕ over constraints such that for any constraint c:

$$\Delta \cap \|c\| \quad = \quad \bigcup_{c' \in \phi(c)} \|c'\|$$

This means that it is possible to characterize the elements of Δ which satisfy a constraint c in terms of the more specific constraints in $\phi(c)$. For example, if Δ itself is of the form $\bigcup_i \|d_i\|$ where d_i are constraints, it is then possible to choose

$$\phi(c) \quad = \quad \{c \sqcap d_i\}_i$$

Given the mapping ϕ, we can now give a more efficient definition of the retrieval engine $\Pi_\Delta(E)$ by

$$\begin{array}{lcl} \lfloor \Pi_\Delta(E) \rfloor(c) & = & \bigcup_{c' \in \phi(c)} \lfloor E \rfloor(c') \\ \lceil \Pi_\Delta(E) \rceil(c) & = & \bigcup_{c' \in \phi(c)} \lceil E \rceil(c') \end{array}$$

In other words, this engine interleaves retrieval steps (using E) and restriction steps (using ϕ).

Proposition 3 *Let Ω be a subset of the domain and E be a retrieval engine. If E is locally sound and complete w.r.t. Ω then $\Pi_\Delta(E)$ is locally sound and complete w.r.t. $\Pi_\Delta(\Omega)$.*

This property has only been proved for the local versions of soundness and completeness. Extension to the full case may require extra assumptions on E.

3.2.2 Induction brokers

An *induction* broker over domain $\mathcal{D}$ is associated to a set of generators. Each generator is a mapping $B : \mathcal{D}^n \mapsto \wp(\mathcal{D})$, i.e. it maps each n-tuple of elements of the domain into zero, one or more new "compound" elements. The induction broker for $\mathcal{B}$ is defined by:

$$\Pi_\mathcal{B}(\Omega) \quad = \quad \bigcap_{\Omega \subset \Omega' \ \wedge \ \mathcal{B} \vdash \Omega'} \Omega'$$

where the relation $\vdash$ is defined for any set $\mathcal{B}$ of generators and subset Ω of the domain as follows:

$$\mathcal{B} \vdash \Omega \quad \textbf{if and only if} \quad \forall B \in \mathcal{B} \ \ \forall x_1, \ldots, x_n \in \Omega \ \ B(x_1, \ldots, x_n) \subset \Omega$$

It is easy to show that $\Omega \subset \Pi_\mathcal{B}(\Omega)$ and $\mathcal{B} \vdash \Pi_\mathcal{B}(\Omega)$. In fact, $\Pi_\mathcal{B}(\Omega)$ is the smallest $\mathcal{B}$-stable subset of the domain that contains Ω.
A treatment of induction brokers has already been described and illustrated in [2] in a model without virtual answers. Dealing with virtual answers requires an extension of the constraint language $\mathcal{C}$. We first introduce *constrained terms* over $\mathcal{C}$ as follows:

1. Any element of the domain is a constrained term.
2. The symbol $\Box$ is a constrained term.
3. If B is a generator of arity n and $t_1, \ldots, t_n$ are constrained terms,
   then so is $B[t_1 : l_1, \ldots, t_n : l_n] : c$ where each l_i is a list of base constraints and c is a base constraint.

We define the interpretation $\|l\|$ of a list l of base constraints as the intersection of the interpretations of its elements. The interpretation $\|t\|$ of a constrained term t is given by:

$$\begin{array}{l} \|a\| \;=\; \{a\} \;\; \textbf{if } a \text{ is an element of the domain} \\ \|\Box\| \;=\; \mathcal{D} \\ \|B[t_1 : l_1, \ldots, t_n : l_n] : c\| \;=\; \|c\| \cap \bigcup_{x_i \in \|l_i\| \cap \|t_i\|} B(x_1, \ldots, x_n) \end{array}$$

The extended constraint language $\Pi_{\mathcal{B}}(\mathcal{C})$ consists of the base constraints and the proper constrained terms t, i.e. those containing at least one generator. Thus, the extended constraints allow to specify the same subsets of the domain as the base constraints, plus subsets which are obtained by recursive application of the generators, fixing some of their input arguments and constraining their output (and possibly input arguments) with base constraints. Given a retrieval engine E on $\mathcal{C}$, we can now define the retrieval engine $\Pi_{\mathcal{B}}(E)$ on the extended constraint language as follows (for any base constraint c and proper constrained term t):

$$\begin{array}{lcl} \lfloor \Pi_{\mathcal{B}}(E) \rfloor(c) & = & \lfloor E \rfloor(c) \\ \lfloor \Pi_{\mathcal{B}}(E) \rfloor(t) & = & \textbf{if } t \text{ is ground } \textbf{then } \|t\| \textbf{ else } \emptyset \\ \lceil \Pi_{\mathcal{B}}(E) \rceil(c) & = & \lceil E \rceil(c) \cup \{B[\Box : \epsilon, \ldots, \Box : \epsilon] : c \;\; \textbf{where } \; B \in \mathcal{B}\} \\ \lceil \Pi_{\mathcal{B}}(E) \rceil(t) & = & \textit{see below} \end{array}$$

Thus, for a base constraint c, the plain answers given by $\Pi_{\mathcal{B}}(E)$ are those given by E and the virtual answers are those given by E on c, plus the constrained terms of the form $B[\Box : \epsilon, \ldots, \Box : \epsilon] : c$ for $B \in \mathcal{B}$, which specifies the elements satisfying c and obtained by the generator B applied to any elements. Plain answers to a proper constrained term t are obtained only when t is ground (i.e. contains no occurrence of $\Box$), i.e. when all the generators in t have all their arguments assigned and can actually be computed.
To define the virtual answers to a proper constrained term, we need a "strategy" which, given a proper constrained term t, returns an occurence of $\Box$ in t together with a constraint on the possible values at that occurrence. Typically, a strategy will perform local constraint propagations on the constraints stated on the input and output of each generator contained in t, and taking into account the arguments that are assigned. Once all the propagations are done, it chooses one occurrence of $\Box$ and yields the constraints propagated at that occurrence. The constraint propagation part may be a systematic mechanism using knowledge on the generators. It may fail if the constraints become inconsistent. The selection of an occurrence of $\Box$ is more of a heuristic kind (e.g. take the "left-most" or the "most constrained" occurrence of $\Box$). Finally we have, given a proper constrained term t:

- If t is ground, or if the strategy fails on t, then $\lceil \Pi_{\mathcal{B}}(E) \rceil(t) = \emptyset$
- Otherwise, let u, c be, respectively, the occurrence of $\Box$ in t and the inferred constraint at that occurrence computed by the strategy.

$$\begin{array}{rcl} \lceil \Pi_{\mathcal{B}}(E) \rceil(t) & = & \{\textbf{Assert}_{u:c'}(t) \textbf{ where } c' \in \lceil E \rceil(c)\} \;\cup \\ & & \{\textbf{Assert}_{u:a}(t) \textbf{ where } a \in \lfloor E \rfloor(c)\} \;\cup \\ & & \{\textbf{Assert}_{u:t'}(t) \textbf{ where } t' = B[\Box : \epsilon, \ldots, \Box : \epsilon] : c \textbf{ for some } B \in \mathcal{B}\} \end{array}$$

Intuitively, the proper constrained term $\textbf{Assert}_{u:\alpha}(t)$ refines t by asserting a piece of information α at an occurrence u of $\Box$ in t. The information α may be an element of the domain or a proper constrained term (last two cases above), in which case $\textbf{Assert}_{u:\alpha}(t)$ is simply obtained by substituting α at occurrence u in t. But α may also be a base constraint c' (as in the first case above). In that case, $\textbf{Assert}_{u:\alpha}(t)$ is obtained by considering the subterm $B[t_1 : l_1, \ldots, t_n : l_n] : c$ of t in which the occurrence u of $\Box$ is one of the t_i, and appending c' to l_i.
We make some minimal assumptions concerning the strategy. Given some proper constrained terms t, let u, c be the occurrence of $\Box$ in t and the constraint computed by the strategy.

- For any element a of the domain, if $\|\textbf{Assert}_{u:a}(t)\|$ is not empty, then $a \in \|c\|$. This states the correctness of the constraint propagation performed by the strategy.
- The inferred constraint c entails at least all the constraints in the (possibly empty) list binding the occurrence u of $\Box$ in t. This means that the strategy propagates at least the constraints that are explicit in t.
- Finally, if t' is obtained from t by assertions at other occurrences than u (so that u is still an occurrence of $\Box$ in t'), and if the strategy returns u, c' for t', then c' entails c. This means that with more information, the strategy propagates more constraints.

Conjecture 4 *Let Ω be a subset of the domain and E be a retrieval engine. If E is sound and complete w.r.t. Ω, then $\Pi_{\mathcal{B}}(E)$ is sound and complete w.r.t. $\Pi_{\mathcal{B}}(\Omega)$ on $\mathcal{C}$.*

It is essential to restrict to the base constraints $\mathcal{C}$, otherwise soundness could obviously be violated: consider for example a ground proper constrained term t containing occurrences of elements of the domain which are not in Ω. The proof of this result has not yet been entirely checked; we keep on the safe side by stating it as a conjecture.

4 Conclusion

In this paper, we have introduced the notion of "virtual answers" to capture the wide diversity of interaction protocols with information sources over a large network such as the Web. Therefore, we have developed a general constraint-based interaction model capable of modelling a wide variety of interaction protocols. We have studied properties of soundness and completeness in this model, as well as generic brokering mechanisms. Currently, our model is under evaluation for a potential integration into the Constraint Based Knowledge Broker system, a system aiming at heterogeneous information retrieval, schema integration, and knowledge fusion. In fact, the main issue is not so much in the brokering mechanism itself: for example, the algorithm described in [2] can be extended to account for virtual answers without major hitch and using the same underlying principles. More challenging issues arise from the adaptation of interfaces (both user interfaces and information source wrappers) to the new interaction model. In particular, one challenge is to provide assistance in the design of "virtual answer"-aware wrappers, extending the current wrapper generation tools described in [8] based on semi-automatic parsing of the returned pages.

References

[1] H. Aït-Kaci, A. Podelski, and G. Smolka. A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science*, 122:263–283, 1994.

[2] J-M. Andreoli, U. Borghoff, and R. Pareschi. The constraint-based knowledge broker model: Semantics, implementation and analysis. *Journal of Symbolic Computation*, 21(4):635–667, 1996.

[3] J-M. Andreoli, U. Borghoff, and R. Pareschi. Signed feature constraint solving. In *Proc. of the 3rd Int'l Conf. on the Practical Application of Constraint Technology (PACT'97)*, pages 35–46, London, U.K., 1997.

[4] J-M. Andreoli, U. Borghoff, R. Pareschi, and J. Schlichter. Constraint agents for the information age. *Journal of Universal Computer Science*, 1(12):762–789, 1995.

[5] R. Brachman, D. McGuinness, P. Patel Schneider, L. Resnick, and A. Borgida. Living with CLASSIC: When and How to Use a KL-ONE-Like Language. In J. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, San Mateo, Ca., U.S.A., 1991.

[6] R.J. Brachman and J.G. Schmolze. An overview of the kl-one knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[7] B. Chidlovskii and U. Borghoff. Query translation for distributed information gathering on the web. In *Proc. of 2nd IEEE Int. Database Engineering and Application Symp. (IDEAS'98)*, pages 214–223, Cardiff, U.K., 1998.

[8] B. Chidlovskii, U. Borghoff, and P-Y. Chevalier. Towards sophisticated wrapping of web-based information repositories. In *Proc. of 5th Int'l. RIAO Conf. on Computer-Assisted Information Searching on the Internet*, pages 123–135, Montreal, Canada, 1997.

[9] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml: A language and protocol for knowledge and information exchange. In K. Fuchi and T. Yokoi, editors, *Knowledge Building and Knowledge Sharing*. Ohmsha and IOS Press, 1994.

[10] W. Litwin and T. Risch. Main memory oriented optimization of oo queries using typed datalog with foreign predicates. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):517–528, 1992.

A Proofs of the Propositions

Given a retrieval engine E and a constraint c, we denote $\lceil E \rceil^\star(c)$ the set of paths of origin c in the binary relation defined by $\lceil E \rceil$. In other words:

$$\lceil E \rceil^\star(c) \quad = \quad \{c_0, \ldots, c_N \,/\, c_0 = c \ \wedge \ \forall n < N \ c_{n+1} \in \lceil E \rceil(c_n)\}$$

Proposition

Let E be a retrieval engine and c a constraint. Let's show that

$$\lceil E \rceil_\star(c) \quad = \quad \{c_N \ / \ N \in \aleph \wedge c_0, \ldots, c_N \in \lceil E \rceil^\star(c)\} \tag{1}$$

Let C be the above set.

- By considering the path of length 0 at c, we have that C contains c.
- C is E-stable. Indeed, let's consider an element of C. By definition, it is of the form c_N for some path $c_0, \ldots, c_N$ in $\lceil E \rceil^\star(c)$. Let d be in $\lceil E \rceil(c_N)$. We have to show that $d \in C$, which is straightforward since the sequence $c_0, \ldots, c_N, d$ belongs to $\lceil E \rceil^\star(c)$, hence its end element is in C.
- Any E-stable set containing c contains C. Indeed, let C' be an E-stable set containing c. It is easy to prove by induction on N that $\forall c_0, \ldots, c_N \in \lceil E \rceil^\star(c)\ c_N \in C'$, and hence $C \subset C'$.

Hence C is the smallest E-stable set containing c, which is, by definition, $\lceil E \rceil_\star(c)$.

Proposition

Let E be a retrieval engine and c a constraint. Let's show that

$$\lfloor E \rfloor_\star(c) \quad = \quad \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)} \lfloor E \rfloor_\star(d) \tag{2}$$

By definition of $\lfloor E \rfloor_\star$ and application of result (1), we have

$$\lfloor E \rfloor_\star(c) = \bigcup_{c_0, \ldots, c_N \in \lceil E \rceil^\star(c)} \lfloor E \rfloor(c_N)$$

If we distinguish the case $N = 0$, we obtain

- There is only one path of origin c and length 0, reduced to c.
- Paths of origin c and length $N > 0$ are exactly obtained from paths of origin $d \in \lceil E \rceil(c)$ and length $N - 1$.

Hence $\lfloor E \rfloor_\star(c) \ = \ \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)\ c_1, \ldots, c_N \in \lceil E \rceil^\star(d)} \lfloor E \rfloor(c_N) \ = \ \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)} \lfloor E \rfloor_\star(d)$ by application of result (1) and definition of $\lfloor E \rfloor_\star$.

Proposition

Let E be a retrieval engine.

- If E is Ω-sound then E is locally Ω-sound. Indeed, soundness means $\lfloor E \rfloor_\star(c) \ \subset \ \|c\| \cap \Omega$. Since $\lfloor E \rfloor(c) \ \subset \ \lfloor E \rfloor_\star(c)$, we have $\lfloor E \rfloor(c) \ \subset \ \|c\| \cap \Omega$, i.e. local soundness.
- If E is locally Ω-sound then E is Ω-sound. Indeed, local soundness means that $\lfloor E \rfloor(d) \ \subset \ \|d\| \cap \Omega$ for all d, hence for all c

  $$\lfloor E \rfloor_\star(c) \ = \ \bigcup_{d \in \lceil E \rceil_\star(c)} \lfloor E \rfloor(d) \ \subset \ \bigcup_{d \in \lceil E \rceil_\star(c)} \|d\| \cap \Omega$$

  Now, the set $\{d \ / \ \|d\| \subset \|c\|\}$ is E-stable and contains c, hence it contains $\lceil E \rceil_\star(c)$. Hence $\|d\| \subset \|c\|$ for all $d \in \lceil E \rceil_\star(c)$. Hence the result.
- If E is Ω-complete then E is locally Ω-complete. Indeed, completeness means $\|c\| \cap \Omega \ \subset \ \lfloor E \rfloor_\star(c)$. Hence, by application of result (2), we have

  $$\|c\| \cap \Omega \ \subset \ \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)} \lfloor E \rfloor_\star(d) \ \subset \ \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)} \|d\|$$

  i.e. local completeness.

Proposition

Let E be a locally Ω-complete retrieval engine such that the relation $d \in \lceil E \rceil(c)$ be a true refinement. Let's show that E is Ω-complete.

- We first show by induction on N that

$$\forall c \in \mathcal{C} \quad \|c\| \cap \Omega \quad \subset \bigcup_{c_0,\ldots,c_N \in \lceil E \rceil^\star(c)} \left(\bigcup_{n<N} \lfloor E \rfloor(c_n) \cup (\|c_N\| \cap \Omega) \right)$$

  The initial case is a tautology and the induction is a bare application of the local completeness property to each $\|c_N\| \cap \Omega$ above. Now, by combining the above property, the definition of $\lfloor E \rfloor_\star$ and result (1), we have, for all number N and constraint c,

$$\|c\| \cap \Omega \quad \subset \quad \lfloor E \rfloor_\star(c) \cup \bigcup_{c_0,\ldots,c_N \in \lceil E \rceil^\star(c)} \|c_N\| \cap \Omega$$

  and hence, for all constraint c,

$$\|c\| \cap \Omega \quad \subset \quad \lfloor E \rfloor_\star(c) \cup \bigcap_N \bigcup_{c_0,\ldots,c_N \in \lceil E \rceil^\star(c)} \|c_N\| \cap \Omega$$

- We then show, using the true refinement hypothesis, that

$$\bigcap_N \bigcup_{c_0,\ldots,c_N \in \lceil E \rceil^\star(c)} \|c_N\| \cap \Omega \quad \subset \quad \lfloor E \rfloor_\star(c)$$

  We reason by contradiction, assuming that there exists an element a in the set

$$\left[\bigcap_N \bigcup_{c_0,\ldots,c_N \in \lceil E \rceil^\star(c)} \|c_N\| \cap \Omega\right] \quad \setminus \quad \lfloor E \rfloor_\star(c)$$

  and showing that this leads to a contradiction. Indeed, let γ be the sequence of constraints defined as follows:

  - $\gamma_0 = c$. Note that it is easy to prove that $a \in \|\gamma_0\|$ (case $N = 0$).
  - Let γ_n be such that $a \in \|\gamma_n\|$. By local completeness, we have two possibilities:
    * $a \in \|d\|$ for some $d \in \lceil E \rceil(\gamma_n)$. We then take γ_{n+1} to be one of these d, and we therefore have $a \in \|\gamma_{n+1}\|$. Note also that $\gamma_{n+1} \in \lceil E \rceil(\gamma_n)$.
    * $a \in \lfloor E \rfloor(\gamma_n)$. But then, $\gamma_0, \ldots, \gamma_n$ is a path of origin c with $\gamma_{i+1} \in \lceil E \rceil(\gamma_i)$, hence it an element of $\lceil E \rceil^\star(c)$. Hence, using result (1), we have $\gamma_n \in \lceil E \rceil_\star(c)$, and, by definition of $\lfloor E \rfloor_\star$, we conclude that $\lfloor E \rfloor(\gamma_n) \subset \lfloor E \rfloor_\star(c)$. Therefore $a \in \lfloor E \rfloor_\star(c)$, which is a contradiction.

  Thus we have built an infinite sequence γ_n such that $\gamma_{n+1} \in \lceil E \rceil(\gamma_n)$ and $a \in \|\gamma_n\|$ for all n, which contradicts the hypothesis that E is a true refinement.

- By combining the two inclusions above, we obtain $\|c\| \cap \Omega \subset \lfloor E \rfloor_\star(c)$ i.e. completeness.

Proposition

Let Δ be a subset of the domain, and let E be a locally Ω-sound and complete retrieval engine. Let's show that $\Pi_\Delta(E)$ is locally $\Pi_\Delta(\Omega)$-sound and complete.

- Let c be a constraint. Local soundness of E means $\lfloor E \rfloor(c) \subset \|c\| \cap \Omega$. Hence,

$$\begin{array}{l} \lfloor \Pi_\Delta(E) \rfloor(c) \quad = \quad \bigcup_{c' \in \phi(c)} \lfloor E \rfloor(c') \quad \subset \\ \bigcup_{c' \in \phi(c)} \|c'\| \cap \Omega \quad = \quad \|c\| \cap \Delta \cap \Omega \quad = \quad \|c\| \cap \Pi_\Delta(\Omega) \end{array}$$

  Hence $\Pi_\Delta(E)$ is locally $\Pi_\Delta(\Omega)$-sound.

- Let c be a constraint. Local completeness of E means $\|c\| \cap \Omega \subset \lfloor E \rfloor(c) \cup \bigcup_{d \in \lceil E \rceil(c)} \|d\|$. Hence,

$$\begin{array}{l} \|c\| \cap \Pi_\Delta(\Omega) \quad = \quad \|c\| \cap \Delta \cap \Omega \quad = \quad \bigcup_{c' \in \phi(c)} \|c'\| \cap \Omega \quad \subset \\ \bigcup_{c' \in \phi(c)} (\lfloor E \rfloor(c') \cup \bigcup_{d \in \lceil E \rceil(c')} \|d\|) \quad = \quad \bigcup_{c' \in \phi(c)} (\lfloor E \rfloor(c')) \cup \bigcup_{d \in \bigcup_{c' \in \phi(c)} \lceil E \rceil(c')} \|d\| \quad = \\ \lfloor \Pi_\Delta(E) \rfloor(c) \cup \bigcup_{d \in \lceil \Pi_\Delta(E) \rceil(c)} \|d\| \end{array}$$

  Hence $\Pi_\Delta(E)$ is locally $\Pi_\Delta(\Omega)$-complete.