ELSEVIER

Knowledge-Based Systems 15 (2002) 147–167

Knowledge-Based SYSTEMS

www.elsevier.com/locate/knosys

On the insufficiency of ontologies: problems in knowledge sharing and alternative solutions

Flavio S. Correa da Silva[a,1], Wamberto W. Vasconcelos[b,*], David S. Robertson[b], Virginia Brilhante[b], Ana C.V. de Melo[a,1], Marcelo Finger[a,1], Jaume Agustí[c,2]

[a]*Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão, 1010, 05508-900 São Paulo, Brazil*
[b]*Division of Informatics, University of Edinburgh, 80 South Bridge, EH1 1HN Scotland, UK*
[c]*Institut d'Investigació en Intel.ligencia Artificial, Consejo Superior de Investigaciones Científicas (IIIA/CSIC) UAB, E-08193 Bellaterra, Catalonia, Spain*

Received 10 October 2000; accepted 2 February 2001

Abstract

One of the benefits of formally represented knowledge lies in its potential to be shared. Ontologies have been proposed as the ultimate solution to problems in knowledge sharing. However even when an agreed correspondence between ontologies is reached that is not the end of the problems in knowledge sharing. In this paper we explore a number of realistic knowledge-sharing situations and their related problems for which ontologies fall short in providing a solution. For each situation we propose and analyse alternative solutions. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Ontologies; Knowledge sharing; Knowledge representation and inference

1. Introduction

One of the benefits of formally represented knowledge lies in its potential to be shared. The opportunity presented by computer interconnection technologies such as the Internet is for locally produced knowledge bases and inference mechanisms to interact in solving problems which are more complicated than each individual system could tackle on its own. However, in many such situations we have to face problems concerning how we can ensure, or at least assess, the reliability of different forms of cooperation, given that each system may have been developed by different people and are likely to be based on different languages and formalisms.

Ontologies [24,33,35,52] have been proposed as the ultimate solution to problems in knowledge sharing. Ontologies provide explicit mappings between shared concepts from formalisms of different systems [52]. However, as pointed out in Ref. [16], even when an agreed correspondence between ontologies is reached that is not the end of the problems. In this paper we explore a number of realistic knowledge-sharing situations and their related problems for which ontologies fall short in providing a solution. For each of such problems we propose and analyse alternative solutions.

The structure of this paper is as follows. In the rest of this section we further discuss our aimed class of knowledge-based systems (KBSs, for short) and their properties; we also present a parallel between knowledge sharing and object-oriented technologies (Section 1.2) and a brief discussion on the existing proposals for the communication among KBSs (Section 1.3). We present the organisation of the rest of this paper in the ensuing paragraphs.

Even when a single language can be employed to represent the messages exchanged among the systems, this by no means implies that a single *theory* can be constructed for the shared ontologies among those systems. This simplified situation is obviously desirable and should be pursued whenever possible. In Ref. [24] we find the proposal of an 'ontology server', to act as a tool to construct large collections of shared ontologies. In Section 2 we address the difficulties inherent to this approach.

In Ref. [28] we find the differentiation between KBSs and

* Corresponding author. Tel.: +44-131-650-2718; fax: +44-131-650-6513.

E-mail addresses: fcs@ime.usp.br (F.S. Correa da Silva), wvasconcelos@acm.org (W.W. Vasconcelos), dr@dai.ed.ac.uk (D.S. Robertson), virginia@dai.ed.ac.uk (V. Brilhante), acvm@ime.usp.br (A.C.V. de Melo), mfinger@ime.usp.br (M. Finger), agusti@iiia.csic.es (J. Agustí).

[1] Fax: +55-11-3818-6134.

[2] Tel.: +34-9-3-5809570; fax: +34-9-3-5809661.

PII: S0950-7051(01)00152-6

knowledge base repositories, being the latter used to store *declarative* knowledge and information, akin and subject to the same care requested by a database to maintain integrity and accessibility. Sharing declarative knowledge (e.g. contained in knowledge base repositories) can be quite diverse from sharing *reasoning* (i.e. deductive) knowledge, as shown in Section 3.

Modelling the content of the messages to be exchanged between pairs of KBSs is a semantic problem. Since the construction of the corresponding logical theory of the interlingua is carried through at the syntactical level, it can happen that some important semantic issues become invisible in the implementation, nevertheless they may be still requirements to maintain consistency of the interpretation of messages. This issue is further analysed in Section 4.

The knowledge sharing strategy for the construction and application of KBSs may lead to more and more distributed and fragmented networks of systems. Individual KBSs may thus become smaller, and it may happen that a request not be solvable by a single system, but still be solvable by an appropriate combination of operations from many systems. In this case, a *knowledge broker* is demanded to split a single request into a collection of requests, gather the corresponding operation execution results and assemble them to satisfy the demanding client. This situation is explored in Section 5.

Finally, in Section 6 we draw some conclusions and discuss the results presented.

1.1. Knowledge-based systems: issues and features

Knowledge sharing is closely connected to knowledge-based systems, which are, on their turn, founded on *knowledge*. The most commonly used definition of knowledge is *justified true belief*, as found in, for instance, the work of Delgrande and Mylopoulos [18]. Hence, KBSs are computer software systems that work with justified true beliefs. We can expand this a bit, and propose that a KBS has as its content collections of information structures — representations of *beliefs* — for which we define formal interpretations that grant them the status of *true beliefs*. Moreover, this status can be checked by means of (most often deductive) proofs, that provide them a formal *justification*.

A large number of formal languages exist for KBSs, each being chosen according to its adequacy for the representation of the knowledge related to specific domains, ease-of-use, popularity, and existing supporting tools. For instance, bayesian and probabilistic languages [43] are convenient for uncertain knowledge; linear logics [30] are convenient for representing knowledge whose inferences may be resource-bounded, and so on. Attempts to provide heterogeneous sources of knowledge with the ability to cooperate have arisen in different contexts. Some projects related to this idea can be found [32,42,50]. A more formal counterpart to these proposals, oriented to formal specification of software, can be found in Refs. [9,31].

KBSs are best presented in terms of logical systems, although this does not necessarily mean that they have to be implemented as such. If we concentrate on deductive logics we can finally achieve the conceptualisation of a KBS that we intend to adopt throughout this work. We shall regard a KBS as a deductive theory, written in a particular logic language with well-defined formal semantics, which includes the expected soundness and completeness results for the deductive theory with respect to the corresponding semantic theory. A *query* is a logical conjecture posed to the deductive theory, whose truth evaluation is justified by proof-theoretical means.

KBSs are built for 'practical' problem-solving, hence the semantics of their corresponding deductive theories must be convincing enough as models for the problems being solved. Furthermore, the proof generation procedures must be appropriately implemented as efficient pieces of software, and they must be appropriately wrapped as usable pieces of software, with accordingly well-engineered interfaces.

Building a full fledged KBS can be a highly costly endeavour, as it requires a deep analysis of the system's problem domain and problem solving procedures, the engineering of an appropriate logical language for the system, the modelling and formal reconstruction of the problem and its solving procedures in terms of the semantics of the corresponding logical language. It is also required that the actual construction of the deductive theory which models the specific problem and procedures are taken into account, and, of course, the whole software engineering business related to implementing the system, including testing, validation, code optimisation and interface development.

A historical account[3] of the development of KBSs shows that the first systems constructed benefited from the existence of each other only with respect to the developed skills and generic methods in use, that could be passed away and exploited in the construction of the following systems. Little of the accumulated domain knowledge, deductive theories and underlying languages could be inherited from one system to another.

1.2. Knowledge sharing and object-oriented systems

Recent developments in software construction technology have suggested that a more efficient approach to the development of KBSs could be taken, that would lead to cost-effective construction of higher-quality systems. Namely, techniques related to distributed object oriented systems development and implementation have influenced the development of KBSs towards knowledge sharing [24,27–29,41].

A well accepted standardised architecture for implementing distributed object oriented systems is the Common Object Request Broker Architecture (CORBA) proposed

[3] This can be inferred from the reported experiences of development of early KBSs, as found in e.g. Ref. [37].

by the Object Management Group. According to the CORBA specification [45]:

- '(…) an object system is a collection of objects that isolates the requestors of services (clients) from the providers of services by a well-defined encapsulating interface.'
- '(…) An object system provides services to clients. A *client* is any entity capable of requesting the service.'
- '(…) An *object* is an identifiable, encapsulated entity that provides one or more services that can be requested by a client.'
- '(…) A *request* is an event, i.e. something that occurs at a particular time. The information associated with a request consists of an operation, a target object, zero or more (actual) parameters, and an optional request context.'
- '(…) An *interface* is a description of a set of possible operations that a client may request of an object.'
- '(…) An *operation* is an identifiable entity that denotes a service that can be requested.'

Clients can be external users or objects themselves. Abiding by this architecture, we can implement objects and clients in a distributed fashion. One advantage of adopting this discipline of software system construction is the opportunity for object reuse that can improve software production efficiency and reliability. Object reuse requires that requests, interfaces and operations are in accordance with a standard communication protocol (e.g. the OMG Interface Definition Language in CORBA).

If we identify objects with KBSs, it becomes clear that these concepts can be borrowed to describe what knowledge sharing is about: requests are queries, presented in a standardised format to a knowledge broker, *mediator* or *matchmaker* that, based on available interfaces, will trigger the appropriate KBSs and send back the results of corresponding operations:

> *Knowledge sharing is the process of conveying knowledge embedded into one KBS to another.*

The key advantage of knowledge sharing lies on the reuse of (partial) capabilities embedded into KBSs [41]. These capabilities include declarative knowledge, representation and (inferential, deductive) reasoning capabilities.

The implementation of knowledge sharing services asks for the provision of adequate communication infrastructure, to handle requests, advertisement of capabilities and exchanging of messages that can result from the execution of operations within KBSs. There exist standard solutions for the lower level message-passing services, as well as for the required distributed object management services (e.g. the already mentioned CORBA standards). Moreover, given that the objects and messages being passed are of a specialised kind — namely, they are all queries and responses to queries directed to and/or generated by KBSs — there is room for further standardisation, that can improve reliability and efficiency in system production.

1.3. Communication among KBSs

The Knowledge Query and Manipulation Language (KQML) [28,39] has been proposed as the standard for the communication layer among KBSs. It provides for the encapsulation and passing of messages containing queries and responses to queries. The actual content of the messages has to be expressed using a different language,[4] commonly called an *Interlingua* [27,42] because it must be capable of expressing concepts that are common to systems that exchange messages.

Strictly speaking, there is no reason to assume that a generic interlingua can be constructed which is sufficiently expressive to discriminate contents of messages to be exchanged between every pair of KBSs. It has been argued that a language with the expressive power of first-order logic can be sufficient for the representation of most of such messages. An example of interlingua based on this point of view that has been employed with relative success is the Knowledge Interchange Format (KIF) [29]. The interlingua is thus used to represent concepts belonging to a System that have to be passed over to another one. These concepts must therefore 'make sense' in both systems, i.e. they must belong to the *shared ontologies* of both systems. Following Gruber [33], "an ontology is an explicit specification of a conceptualisation". A shared ontology is the explicit specification of shared conceptualisations, i.e. a common representation for concepts that belong to more than one system.

Building a theory of shared ontologies for a pair of KBSs is in many senses similar to building a KBS of its own, since it requires a deep analysis of the systems' problem domains *and* problem solving procedures, the engineering of an appropriate logical language to act as interlingua (or the analysis whether the existing ones, e.g. KIF, are suited to act as interlingua), and the formal modelling of the content of the messages to be exchanged between the systems in terms of the semantics of the interlingua. It is also necessary that the actual construction and implementation of the messages be taken care of as queries and operation execution results and their correspondences with expressions internal to the systems exchanging messages, in the form of a logical theory of the interlingua.

2. On sharing and reusing knowledge of multiple ontologies

In this section we describe an experiment in multiple ontologies reuse: given the task of engineering a new

[4] KQML itself could be used to represent the content of messages, but it has not been designed for this purpose and other languages are better equipped for this task.

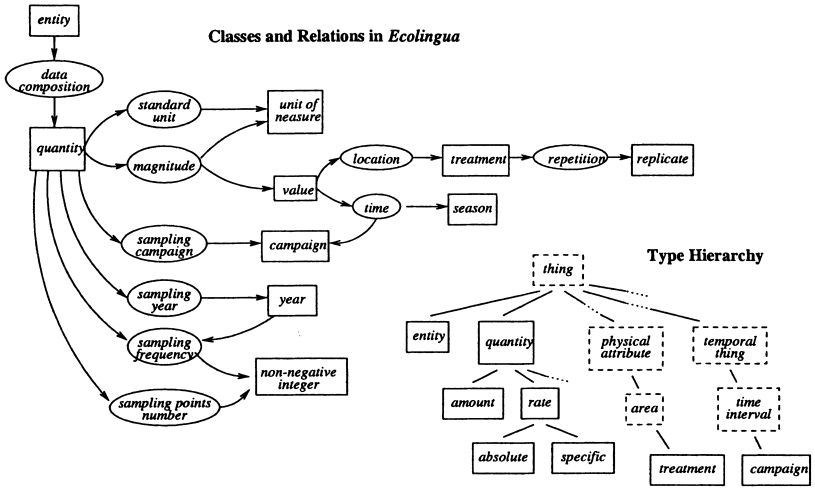


Fig. 1. Part of Ecolingua ontology.

ontology, the easiest way is to reuse existing ontologies whose concepts overlap with those of the new ontology. However, some difficulties have been detected in this approach and we have explored alternative solutions for them.

The knowledge sharing community has started to feel uneasy about the dearth of convincing successes on ontologies reuse and application [53]. Few detailed experiment reports can be found in the relevant literature: the research work of Swartout et al. [51] used SENSUS, a broad natural language based ontology for machine translation, to construct a domain specific ontology for military air campaign planning. Domain specific concepts were linked to the SENSUS hierarchy of concepts. For narrowing down the concepts included, they pruned the hierarchy like in Section 2.2, followed by reintroduction of manually selected relevant concepts. The work of Uschold et al. [54] reuses the EngMath Ontology [34], from the Ontolingua Server library, for deployment of a layout design application, having gone through, like us, the process of translation into a target specification language, Slang.

We wanted to engineer a formal ontology for description of ecological metadata, Ecolingua [5,6]. Its construction has been motivated by the development of an automated ecological modelling system [7], where partial models are generated based on high-level properties of data, e.g. quantities associated with ecological entities, physical dimensions of quantities, sampling campaigns and frequencies, spatial distribution of sampling points, etc. Upon descriptions of these properties through the ontology, automated modelling mechanisms draw partial models that are consistent with the data.

Given the multidisciplinary nature of concepts related to ecological data, we were interested in exploring knowledge reuse for constructing Ecolingua. The extensive library of shareable ontologies made available by the Ontolingua Server [24] led us to choose it as an ontology construction tool. Ontolingua, the server's underlying language, was created as an attempt to solve the portability problem for ontologies. It adopts a translation approach in which ontologies are specified in a standard, system-independent form and translated into specific representation languages [33].

The deployment of an ontology through the server in a form that can be used by some reasoning system is thus a two-fold process. First, the conceptual ontology is created by means of specifying its definitions through the *class*, *slot*, *relation*, *function*, *individual* and *axiom* constructs, and then it is translated into some target implementation language that is adequate to the intended usage of the ontology.

The extra challenge of the Ecolingua experiment, driven by the diversity of the domain, was the reuse of concepts from a number of different ontologies, even though all under the Ontolingua unifying framework. In the light of the effort made in our experiment, where we had all the ontologies in a unique representational framework, we believe that automated tools to support reuse and integration of different ontologies under different representation systems is not yet a reality.

2.1. Conceptual ontology

The Ecolingua [5,6] vocabulary embodies concepts that one would evoke in order to convey descriptions of ecological data properties, functioning as a framework that is instantiated to particular data sets. The collection of concept definitions is structured as a hierarchy of interrelated classes, with axioms restricting interpretations of the definitions. Fig. 1 shows some of the Ecolingua classes and relations together with an excerpt of Ecolingua's type hierarchy. The classes constrain the type of objects involved in the relations. The directed arcs show the direction of the relation, from domain classes to range classes; e.g. the relation sampling frequency gives a non-negative integer (range class) as the number of occasions in which a quantity (domain class) has been sampled along a year (domain class). The classes in dashed boxes in the type hierarchy

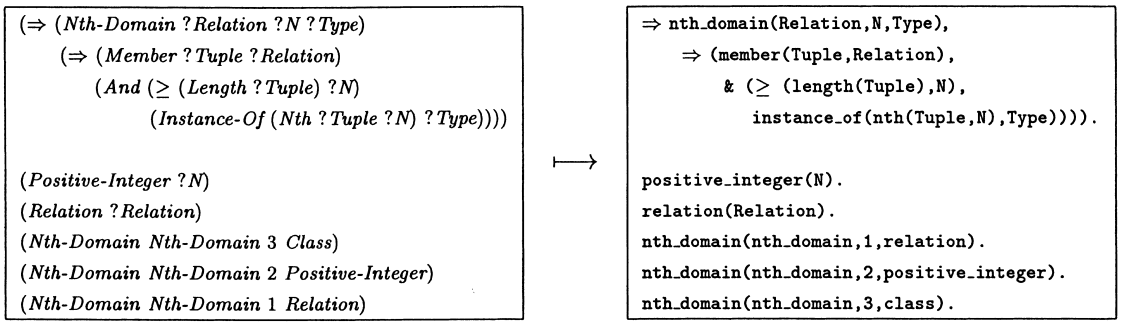


Fig. 2. KIF axioms (left) and their translation into ontolingua Prolog syntax (right).

exemplify reuse of other ontologies' classes as super-classes of the end-ontology classes. *Thing*, *physical attribute*, *area*, *temporal thing* and *time interval* all belong to the same ontology, Hpkb-Upper-Level, from which we reuse definitions.

2.2. Translating conceptual into operational

The conceptual ontology at the time of translation had 37 classes and about the same number of relations/functions. We now describe in detail the steps we have taken in order to have Ecolingua translated into Horn Clauses of a Prolog program [3,40], our choice of target language.

2.2.1. Step 1: automatic translation by the Ontolingua Server into Prolog Syntax

The Ontolingua Server provides a translation service into different target languages, including the so called Prolog syntax. However, it does not produce Prolog runnable code but logical sentences in a KIF-like Prolog readable syntax. This is so because KIF is the formal language used to represent the definitions comprising the Ontolingua server's ontologies. It is a prefix monotonic first-order logic with set theory that subsumes Horn clause logic [29]. Thus, only a subset of the definitions is translatable into standard Prolog. A note on design decisions for translating Ontolingua to Prolog Syntax can be found in Ref. [23].

In Ecolingua we refer to selected definitions in five other ontologies: Hpkb-Upper-Level, Kif-Numbers, Kif-Extensions, Simple-Time and Physical-Quantities. The Ontolingua Server translator provides the option of translating an ontology in isolation, in which case the referenced definitions to other ontologies are not included, or translating the ontology along with all the complete referenced ontologies. The translation of Ecolingua into Prolog Syntax through the former option produces an 85 Kb file, whereas the latter produces a 5.3 Mb file. Obviously, as far as knowledge sharing is concerned, we are interested in translation mechanisms that provide for inclusion of referred definitions. One defines a class, say, *plot* as a subclass of the Hpkb-Upper-Level ontology class *area* for the sake of reusing area definitions, and therefore those should also be part of the translation.

In the following step we show how we re-engineered the 5.3 Mb file (which we shall call hereafter simply workfile) in order to turn it into a better structured, smaller, and, hopefully, more manageable, set of axioms. Initially, minor editing was necessary in order to make the file produced by Ontolingua's translator readable by our adopted Prolog interpreter, SICStus [38]. For instance, literals such as `listof ()`, without any arguments, have appeared — these have been replaced by `listof (_)` with a fake argument to comply with Prolog's syntax. It should be pointed out that Ontolingua's automatic translator cannot cope with some constructs in which case these appear as commented lines in the output file.

2.2.2. Step 2: cleaning up extraneous clauses

Extraneous clauses are over-general facts, definitions of self subclasses and duplicated classes. These clauses should be identified and removed from the work file.

- *Over general facts* — A fact is over general if all its arguments are uninstantiated variables: for instance,
  $overgeneral(F) \leftarrow$
  $is\_ontolingua\_predicate(F) \wedge$
  $F \wedge arguments(F, Args) \wedge$
  $\forall X.var(X) \leftarrow member(X, Args)$.

There were 59 occurrences of over general facts such as the one above in the workfile. Examples are *list*(*X*), *class*(*X*), *relation*(*X*), *set*(*X*), *positive_integer*(*X*) and *natural*(*X*). To illustrate this issue, let us consider in detail a particular clause defining relation *positive_integer*(*X*). It comes from the axioms defining the *Nth-Domain* relation in the Frame ontology, which is part of the Ontolingua Server's library of ontologies. In the server the relation is documented as: 'Domain restrictions generalised to n-ary relations. The sentence (*Nth-domain Rel 3 Type*) says that the 3rd element of each tuple in the relation *Rel* is an instance of the *type* class.'

We show in Fig. 2 both the original KIF axioms and their translations into Ontolingua Prolog syntax. The axioms

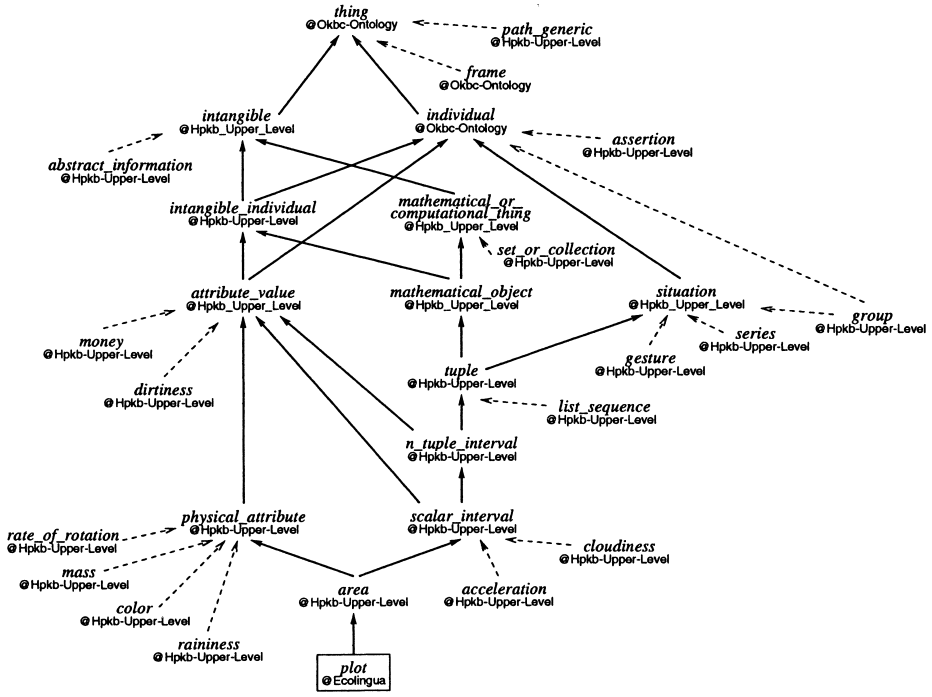


Fig. 3. Extract of the class hierarchy (directed arcs are instances of *subclass_of*(*Class*, *Parent_class*) relation).

(*Positive-Integer* ?*N*) and (*Relation* ?*Relation*) define the classes of objects that instantiate variables ?*N* and ?*Relation*. There is an implicit notion of variable scope here. The variables named ?*N* and ?*Relation* share the same instantiation across the axioms. This is what causes the over general facts in the Prolog version. In Prolog, the scope of a variable does not go beyond the clause where it appears. Thus the variable `N` in the implication axiom and the other variable `N` in the `positive_integer(N)` axiom do not share. Any term can match `N` in the Prolog axiom `positive_integer(N)`, leading to unwanted, yet logically sound instantiations.

- *Self Subclasses* — The Ontolingua server does not allow users to define circular subclasses/superclasses: in such cases an error message 'Cannot have a circular superclass/subclass graph' is displayed. However, the query `subclass_of(X,X)` over the knowledge base in the workfile succeeds for 12 instances of `X`, all of them classes in the `Hpkb-Upper-Level` ontology (e.g. *subclass_of*(*locomotion_process*,*locomotion_process*)).

- *Duplicated Clauses* — Duplicated clauses amounted to 57 Kb of the workfile.

2.2.3. Step 3: pruning the class hierarchy

Every time a class is created in the server it must be defined as a subclass of some other existent class, through the *Subclass-Of* relation. The *Subclass-Of* relation is defined in the Okbc-Ontology as: 'class C is a subclass of parent class P if and only if every instance of C is also an instance of P', with correspondent KIF axiom:

```
( ⇔ (Subclass-Of ?Child-Class ?Parent-Class)
    (Forall (?Instance)
            ( ⇒ (Instance-Of ?Instance ?Child-Class)
                (Instance-Of ?Instance ?Parent-Class))))
```

The parent class may be chosen among the classes of the end-ontology being built or from the classes of any of the ontologies in the server's library (henceforth called alien classes). The chosen parent class in turn could also have been defined as a subclass of some other class. Ultimately, the hierarchy of classes is completed, with the end-ontology classes as leaf nodes and as root nodes, classes from other ontologies which are not defined as being subclasses of some other class. An end-ontology class will not appear as a root node, since every class when created must be defined as a subclass of some other class. There are 18 classes in Ecolingua whose parent classes are defined elsewhere. For most of them we were able to find classes that are conceptually appropriate as parent classes, e.g. the Ecolingua class *plot* is a subclass of *area* defined in the `Hpkb-Upper-Level`. For the remaining classes we defined as parent the catch-all class *thing* in the Okbc-Ontology, since *thing* is the class of everything in the universe of discourse that can be in a class. This includes all the relations and objects defined in the KIF specification, plus all other objects defined in user ontologies.

The resulting class hierarchy is an acyclic graph (or

forest) structure. The graph's nodes are classes and it's directed arcs are of the *subclass_of* relation. It contains 1758 classes altogether, 1721 of which are alien classes. All classes are found as the set of all nodes that take part in the *subclass_of* relation:

$$S_{Nodes} = \{Node | \exists Child, Parent\ .\ subclass\_of(Child, Node) \vee subclass\_of(Node, Parent)\}$$

The roots in the forest are the nodes that have no parent:

$$S_{Roots} = \{Node | \exists Child\ .\ subclass\_of(Child, Node) \wedge not(\exists Parent\ .\ subclass\_of(Node, Parent))\}$$

S_{Roots} contains eight elements: *thing* — Okbc-Ontology; *term*, *expression* and *operator* — KIF-Meta; *set* and *bounded* — KIF-Sets; *list* — KIF-Lists; and *number* — KIF-Numbers. The largest tree by far is the tree under *thing*, containing 1681 classes. The *thing*, *set*, *bounded* and *number* trees are connected encompassing all but 47 of the classes, which are nodes in the four other disjoint trees with roots *term*, *expression*, *operator* and *list*. Altogether, the classes hierarchy topology is a forest of five disjoint connected trees, one of them containing approximately 90% of the classes.

Fig. 3 shows an extract of the class hierarchy. In the figure we adopt the server's notation *Class*@Ontology, denoting class and correspondent ontology. It illustrates the myriad of classes included in the hierarchy, most of them of unlikely practical relevance. For instance, the establishment of the relation *subclass_of*(*plot*, *area*), being *plot* an end-ontology class and *area* defined in the Hpkb-Upper-Level ontology causes all the classes in Hpkb-Upper-Level to be included in the hierarchy.

We prune the hierarchy by traversing it bottom-up: from the leaf nodes, which are end-ontology classes, up to the root nodes. S_P is the set of pruned classes, the ones we keep, if S_{Eco} is the set of the end-ontology classes and S_P is connected to S_{Eco}:

$$pruned\_classes(S_P) \leftarrow S_{Eco} = \{Class | ecolingua\_class(Class)\} \wedge connected(S_P, S_{Eco})$$

S_P is connected to S_{Eco} if it is the set of classes that are in the paths from the end-ontology classes to the forest roots:

$$connected(S_P, S_{Eco}) \leftarrow S_P = \{Class | \exists Eco\_class\ .\ Eco\_class \in S_{Eco} \wedge in\_path(Class, Eco\_class)\}$$

The current class is part of the path:

$$in\_path(Class, Class)$$

The parent class of the current class is also in the path:

$$in\_path(P\_class, Class) \leftarrow subclass\_of(Class, P\_class)$$

And so is any ancestor of the current class:

$$in\_path(A\_class, Class) \leftarrow subclass\_of(Class, P\_class) \wedge in\_path(A\_class, P\_class)$$

This pruning method reduces the class hierarchy from 1758 classes to only 76.

The bold arrows in Fig. 3 represent the paths from *plot*, an end-ontology class, to *thing*, a root class. All classes in these paths are kept. The classes with dashed arcs to their parent classes exemplify classes that are pruned off. They are not in the paths from *plot* to *thing* (and are not in the paths from any of the end-ontology classes to any of the root classes).

Despite reducing the hierarchy size dramatically, the method cannot guarantee that all and only classes of interest, conceptually speaking, are kept.

It is reasonable to expect some specifications (slots and their values and axioms) in the definitions of classes to be useful for a reasoning system using the ontology axioms. For instance, the slot *absolute_value_fn*, the unary mathematical function that returns the absolute value of its argument (Hpkb-Upper-Level ontology), with value type *scalar_interval* that *plot* inherits through the path *plot* → *area* → *scalar_interval*, can be used to ensure that the area of a plot is given as a positive value. Nevertheless, as we move higher along the paths, away from *plot*, the classes become more and more conceptually remotely related to it, rendering unlikely practical reuse of their specifications.

On the other hand, there are classes of interest that end up being pruned off. This is the case, for instance, of some subclasses of *physical_attribute*, such as *raininess*, *color* and *mass* which are desirable as part of Ecolingua's universe of discourse. Only an inspection of the class hierarchy by the ontology designer could guarantee that all and only classes of interest are maintained, but this is obviously impractical when the design involves reuse of multiple and large-scale ontologies.

2.2.4. Step 4: pruning clauses

The objective of this step is to prune our workfile, keeping only the clauses deemed relevant. The relevance criteria are empirical, based on our familiarity with Ontolingua relations. The pruning method presented reduces the workfile down to 1.4 Mb. We start by asserting an initial set of relevant classes, relations and functions. The pruning process consists of selecting clauses that are related to this initial set. We take as **initial relevant classes**:

- The end-ontology classes and all their ancestor classes, i.e. the classes in the set S_P of pruned classes (Section 2.2).

- The classes *class* (Okbc-Ontology), *relation*, *binary_relation* and *function* (Kif-Relations), since these are the meta-ontology classes of which the ontological constructs are instances.

The **initial relevant relations** and **functions** are all relations and functions of the end-ontology.

Every clause in the workfile is checked for relevance, with clauses of certain forms treated specifically. With these specific checks we first select the clauses that specify relevant instances of the *class*, *relation*, *individual* and *axiom* constructs (checks (1)–(5)), and then the clauses that specify *subclass* and *instance* relations involving them (checks (6) and (7)). Clauses of other forms are relevant if they involve relevant classes or relations (check (8)). Below we explain the relevance checks for each of these specific forms of clauses as well as the general case.

(1) Clauses *class*(C). A clause *class*(C) is relevant if C is one of the initial relevant classes:

$$\begin{aligned} &\mathit{relevant}(\mathit{class}(C)) \leftarrow \\ &\quad C \in S_P \vee \\ &\quad C \in \{\mathit{class}, \mathit{relation}, \mathit{binary\_relation}, \mathit{function}\} \end{aligned}$$

Or if there is a relevant class, relation or function T which is an instance of class C:

$$\begin{aligned} &\mathit{relevant}(\mathit{class}(C)) \leftarrow \\ &\quad \mathit{instance\_of}(T,C) \wedge \\ &\quad (\mathit{relevant}(\mathit{class}(T)) \vee \mathit{relevant}(\mathit{relation}(T)) \vee \\ &\quad \mathit{relevant}(\mathit{function}(T))) \end{aligned}$$

instance_of(T,C), *T is an instance of class C*, is an Ontolingua relation defined in the Okbc-Ontology that holds for every ontological term with an associated definition. Every class, relation or function is an instance of some class.[5] For example, the relation *adjacent_to* (Hpkb-Upper-Level ontology) is an instance of the classes *relation* and *spatial_predicate*, the latter being a class of relations. The aim of this check is to include as relevant those classes whose instances (other classes, relations or functions) are relevant.

(2) Clauses *relation*(R) and *function*(R). A clause *relation*(R) or *function*(R) is relevant if R is one of the initial relevant relations or functions, respectively, i.e. R is an end-ontology relation or function:

$$\begin{aligned} &\mathit{relevant}(\mathscr{P}(R)) \leftarrow \\ &\quad \mathscr{P} \in \{\mathit{function}, \mathit{relation}\} \wedge \\ &\quad \mathit{ecolingua\_relation}(R) \end{aligned}$$

Besides the end-ontology ones we should keep relations and functions (relations hereafter, since in Ontolingua, a function is a special kind of relation) that are related to alien classes. *Relations* in Ontolingua are used to describe relationships between n terms, with $n \geq 2$. *domain*(R,C), *the domain of relation R is class C*, also a relation in the Okbc-Ontology, specifies the classes to which each of the $n-1$ terms belong. Each of the $n-1$ terms is an *instance_of* its domain class. For instance, *domain*(*number_of_replicates*, *treatment*) specifies that the function *number_of replicates* applies to objects of the Ecolingua class *treatment*.

The counter-relation to *domain*(R,C) is *range*(R,C), which specifies the class of the nth term in the relation.

As a pruning strategy we keep the relations that have relevant domain classes only, not including relations with relevant range classes. The intuition behind this strategy is that a relation with a relevant domain class has a stronger bond with our relevant classes hierarchy. The strategy stretches the boundaries of the ontology from inside out, absorbing relations that are driven by, or have domains which are, relevant classes. Keeping relations that have relevant range classes would mean we would have to include outsider relations that rather end in relevant classes. Thus:

$$\begin{aligned} &\mathit{relevant}(\mathscr{P}(R)) \leftarrow \\ &\quad \mathscr{P} \in \{\mathit{function}, \mathit{relation}\} \wedge \\ &\quad \mathit{domain}(R, C) \wedge \mathit{relevant}(\mathit{class}(C)) \end{aligned}$$

(3) Clauses *inverse*(X,Y). *inverse*($R1$,$R2$) is a Kif-Relations ontology function that applies over binary relations. Its meaning is that one binary relation is the inverse of another if they are equivalent when their arguments are swapped. For instance, the Simple-Time ontology relations *after* (T_1,T_2) and *before* (T_2,T_1), where T_1 and T_2 are time ranges, are equivalent. Therefore *inverse_of* (*after*,*before*) holds.

These clauses are kept for the sake of retaining in the ontology the knowledge of the inverse relations of relevant relations. Thus, clause *inverse*(R_1,R_2) is relevant if any of R_1 or R_2 is a relevant relation:

$$\begin{aligned} &\mathit{relevant}(\mathit{inverse}(R_1, R_2)) \leftarrow \\ &\quad \mathit{relevant}(\mathit{relation}(R_1)) \vee \mathit{relevant}(\mathit{relation}(R_2)) \end{aligned}$$

(4) *Clauses specifying individuals*. Clauses specifying individuals appear as ground unary predicates $\mathscr{C}(I)$, where I is an individual and $\mathscr{C}$ is the class of which I is a member. For instance, *forest_logging_disturbance_class*(*clearing_edge*) denotes that *clearing_edge* is an individual of the class *forest_logging_disturbance_class*.

We want to keep the clauses that specify individuals of relevant classes, excluding the meta-ontology classes *class*, *relation*, *binary_relation* and *function*. The exclusion is necessary because there exist clauses of the forms *class*(T), *relation*(T), *binary_relation*(T) and *function*(T) for every

[5] An *instance* should not be confused with the Ontolingua construct *individual*. An instance can be a class but an *individual* cannot.

class, relation or function T. Thus:

$$relevant(\mathscr{C}(I)) \leftarrow \\ \quad \mathscr{C} \notin \{class, relation, binary\_relation, function\}\mathscr{C} \wedge \\ \quad relevant(class(\mathscr{C}))$$

(5) *Clauses specifying logical sentences*. Ontologies' axioms get translated by the Ontolingua Server into logical sentences appearing as clauses with the predicates: *exists*, representing the existential quantifier, *forall*, the universal quantifier, $\Rightarrow$, implication, $\Leftarrow$, reverse implication, $\Leftrightarrow$, equivalence, and $-$, negation.

All logical sentences that involve relevant classes, relations or functions should be kept, since they represent the axioms that constraint interpretations of the relevant ontological definitions. Names of classes, relations and functions appear in the logical sentences clauses as functors in the arguments of the logical operators predicates, for instance $\Leftrightarrow (individual(A), - (set(A)))$. Thus, a clause representing a logical sentence is relevant if any of its arguments contains a functor which is a relevant class, relation or function:

$$relevant(Lclause) \leftarrow \\ \quad predicate\_name(Lclause, P) \wedge \\ \quad P \in \{exists, forall, \Rightarrow, \Leftarrow, \Leftrightarrow, -\} \wedge \\ \quad arguments(Lclause, Args) \wedge \\ \quad contains\_functor(Args, F) \wedge \\ \quad (relevant(class(F)) \vee relevant(relation(F)) \vee \\ \quad relevant(function(F)))$$

(6) Clauses *subclass_of*(C_1,C_2). A clause *subclass_of*(C_1,C_2) is relevant if any of C_1 or C_2 is a relevant class:

$$relevant(subclass\_of(C_1, C_2)) \leftarrow \\ \quad relevant(class(C_1)) \vee relevant(class(C_2))$$

The *subclass_of* (C_1,C_2) clauses give shape to the class hierarchy. It is crucial to keep them as part of the pruned ontology to allow for the inheritance mechanism.

(7) Clauses *instance_of*(T,C). A clause *instance_of*(T,C) is relevant if the term T is a relevant class, relation or function:

$$relevant(instance\_of(T, C)) \leftarrow \\ \quad relevant(class(T)) \vee \\ \quad relevant(relation(T)) \vee \\ \quad relevant(function(T))$$

Note that checking relevance of these clauses on the grounds that C is a relevant class would have no pruning effect. The meta-ontology classes *class, relation, binary_relation* and *function* are relevant classes, and to every term T which is a class exists a clause *instance_of*(T,*class*), to every term T which is a relation exists a clause *instance_of*(T, *relation*), and so on.

(8) *Other clauses*. Here we check the relevance of any clause that do not fall into the cases above, i.e. clauses with predicates outside the set:

$$S_{Pred} = \{class, relation, function, \\ \quad subclass\_of, instance\_of, inverse, \\ \quad exists, forall, \Rightarrow, \Leftarrow, \Leftrightarrow, -\}$$

Check (2) above guarantees that the domain classes of relevant relations are part of the pruned ontology, and subsequent checks guarantee that clauses that involve these domain classes are kept. Complementary to that, clauses that contain as a subterm the range class of a relevant relation must be kept. Thus, a clause is relevant if its predicate is not in S_{Pred} and it has a subterm which is the range class of a relevant relation or function:

$$relevant(Clause) \leftarrow \\ \quad predicate\_name(Clause, P) \wedge P \notin S_{Pred} \wedge \\ \quad (relevant(relation(R)) \vee relevant(function(R))) \wedge \\ \quad range(R, Class) \wedge subterm(Clause, Class)$$

And finally the most general case: a clause is relevant if its predicate is not in S_{Pred} and it has a subterm which is a relevant class, relation or function.

$$relevant(Clause) \leftarrow \\ \quad predicate\_name(Clause, P) \wedge \\ \quad P \notin S_{Pred} \wedge subterm(Clause, T) \wedge \\ \quad (relevant(class(T)) \vee relevant(relation(T)) \vee \\ \quad relevant(function(T)))$$

2.2.5. Step 5: transforming logical sentences into Horn clauses

At this stage we have a much smaller collection of KIF expressions but these still are not in the form required for the computations we wish to perform in Prolog. The good news is that, since KIF and Prolog share a common logical parentage, standard truth-preserving transformations from first order predicate calculus to Horn clauses (see for example Ref. [40]) can be used to convert many of the expressions to exactly the form required. For instance, the KIF expression $\Leftrightarrow$ (*a*) (*b and c*) translates to the equivalent set of Horn clauses $\{a \leftarrow b \wedge c, b \leftarrow a, c \leftarrow a\}$. The bad news is that not all KIF expressions are translated into an elegant computational form by this means.

An example is the expression $\Leftrightarrow$ (*a* ?*X*) (*or*(*b* ?*X*)(*c* ?*X*)). As a first stage translation we can obtain the equivalent set of expressions $\{a(X) \leftarrow (b(X) \vee c(X)), a(X) \rightarrow (b(X) \vee c(X))\}$. The first of these is acceptable as a Horn clause but the second is not. There is then an issue about the form into which we should translate the second expression. We observe that in our domain of application (and we suspect in many others) circumstances like this one occur when the KIF expression's most likely computational use is in testing the consistency of the precondition of the implication. In our example we want to show that $b(X)$ or $c(X)$ is true when $a(X)$ is true. Since we are testing X, rather than generating an instance of it, we can use the equivalence

between $P \rightarrow Q$ and $\neg (P \wedge \neg Q)$ to translate to a Prolog goal. For our example this is the consistency constraint $\neg (a(X) \wedge \neg (b(X) \vee c(X)))$.

By being selective in this way we translate all our KIF axioms to Prolog, either as program definitions or as consistency constraints.

2.3. Discussion

The Ontolingua server has been helpful in conceptualising Ecolingua, promoting structured formal specifications to a degree. Although KIF axioms can be written, it is curious to observe their scarcity in most ontologies in the server's library. Ontologies and concepts are abundant, however, selecting relevant ones for reuse can be tedious. The only way to do this is to browse the ontologies and to search for concept names by matching words. No meaning-based search facility is available.

The problems with the translation begin with getting as outcome from the server the union of the full content of all referred ontologies, including the infrastructural definitions of the implicitly imported Frame ontology. The pruning methods we apply are heuristics. We cannot guarantee that the pruned ontology is a self-contained, consistent logical theory. When translating Ontolingua ontologies into any implementation language, one has to compromise on expressiveness and reasoning. Inference engines that support the full expressiveness of KIF, a first-order logic, cannot be built. The common logical parentage between KIF and Prolog allowed our compromises to be relatively modest.

We do not have a metric for cost-effectiveness of this experiment. Neither could we use an off-the-shelf one for metrics on knowledge reuse are yet to be formulated. Ref. [10] is the pioneering work on that. We do have a subjective evaluation, though. If we were to start building Ecolingua, knowing what we have learned, we would still use the Ontolingua server for browsing existing, potentially useful, ontologies. However, we would prefer to manually write down the actual specification of the ontology from scratch and to rewrite selected definitions for inclusion from other ontologies, being in total control of every ontological commitment made.

Applying Ecolingua in the development of a system that synthesises ecological models based on metadata described through it, has been reassuring that constructing the ontology in the way described in this paper was not cost-effective. Even after having significantly pruned the ontology we only actually use a small subset of it. Furthermore, we often have to fine-tune the axioms in this subset to the needs of the application.

3. On sharing declarative × inferential knowledge

There are interesting contexts in which it would be desirable, if not essential, to share not the declarative knowledge but the inferences performed by KBSs. The usual process of translating the knowledge base of KBS_1, the donor, into the formalism of KBS_2, the recipient, and making it all available for local use by KBS_2, is not always adequate — if a knowledge base contains sensitive information or if queries to it can be charged, it is not advisable to supply it to another system.

We regard inferential knowledge as the knowledge implicit in the knowledge base of a KBS and made explicit by its inference engine. If we are to share this kind of knowledge then we must ensure that inferences be appropriately represented and passed on from one KBS to another. There ought to be, beyond the usual ontological correspondence between the communicating systems, a correspondence between the inference engines, in terms of their operators and deduction rules. If the logical formalisms of the KBSs are similar or if entailment relationships between their inferential powers can be detected (e.g. modal logic inferences and classic logic inferences), the correspondence can be easily obtained.

However, an interesting problem arises when not all kinds of inferences provided by one KBS are acceptable by another system, that is, a situation of *partial* knowledge sharing. This can be the case, for instance, of a KBS which would not accept inferences longer than a certain limit or, if the KBS providing the inferences charged for them, beyond a certain price. More sophisticated situations arise when certain operators or rules of inference are not accepted. Since inferences can be very large, the problem of examining them has to be given a computationally feasible solution.

We have investigated in Ref. [26] the problem of inferential knowledge sharing among *resource-sensitive* systems (also called substructural logic systems [19]). The class of substructural logics, encompassing, for instance, intuitionistic logics [17], relevance logics [1] and linear logics [30], employ in their inferences *structural rules* which take into account the structure of premises in a deduction [19]. Substructural logics differ from each other by virtue of the structural rules allowed in their proofs: the set of structural rules permitted in one logic may be extended or reduced thus giving rise to other logics. A number of real-life problems can be naturally represented and elegantly solved via resource-sensitive inferencing. Useful and computationally efficient reasoning systems have been developed employing substructural logics, for instance Refs. [36,56].

We want to achieve knowledge sharing in an opportunistic form: rather than assuming we have a formal description of each system, our scenario will be a more realistic one if we assume that very little is known of all participating systems. A knowledge-based system KBS_1 may want to *try* to use another system KBS_2 by posing it queries and analysing its answers. In this section we describe a computationally efficient approach to perform the analysis of answers. We shall assume that our systems all share the same vocabulary, that is, their knowledge bases are all subsets of a language $\mathscr{P}$. If this assumption does not hold

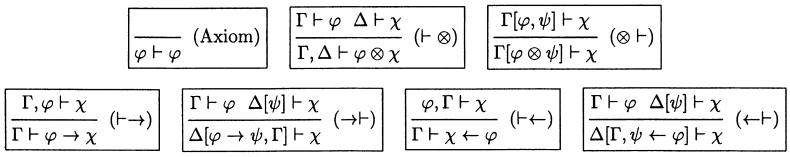


Fig. 4. Connective rules as Gentzen sequent rules.

then a translation function among the languages of each system should be provided [9] or, alternatively, the correspondence among specific constructs to be shared [15] ought to be given.

To enable a KBS to reject an answer from a remote system, the latter must provide not only the answer for a given query, but also describe how that inference was achieved. This means that the donor must provide, together with its answer, a description of the inference steps that led to that conclusion. The receiver will inspect that inference and decide whether to accept or reject it. This poses an extra overhead which may lead to unacceptable inefficiencies. Proofs are normally large objects, usually of orders of magnitude larger than the answer they generate. Procedures to examine a proof and check for properties in it will naturally reflect the size and complexity of the objects involved.

In the following sections we show how this problem can be avoided by sending a much more compact representation of the 'important aspects' of the inferences. We show that this can be done efficiently for the class of resource-sensitive logics known as *substructural logics*.

3.1. Substructural logics

Substructural logics are a family of logics which differ from each other by the set of *structural rules* that each logic in the family accepts. These structural rules determine how resources are dealt with by each logic, and therefore the whole family is also known as *resource sensitive logics*.

For the purpose of this presentation, we will be working with a fragment of the logic defined by the connectives $\otimes$ (multiplicative conjunction), $\rightarrow$ (right implication) and $\leftarrow$ (left implication) (in the absence of commutativity, $\rightarrow$ and $\leftarrow$ are not equivalent). Each logic in the family will obey the connective rules shown in Fig. 4, depicted as Gentzen sequent rules.

As we are not assuming a priori structural rules, the antecedent of a sequent is a binary tree, with formulae at its leaves and ',' at the internal nodes. Antecedents are by default left-associative, so Φ_1, Φ_2, Φ_3 actually represents $((\Phi_1,\Phi_2),\Phi_3)$. The consequent of a sequent is always a single formula. By $\Gamma[\varphi]$ we mean a specific occurrence of φ in the structure Γ, and a corresponding $\Gamma[\Psi]$ in the lower part of a rule means the substitution of that occurrence of φ by the structure Ψ in Γ. What distinguishes one substructural logic from another are the structural rules that are allowed in its inferences.

For example, the Lambek calculus is the logic that accepts only the associativity rules (while the pure Lambek calculus accepts none), which, in terms of resources, means that all formulae must be used in a given order; linear logic accepts associativity and commutativity, so formulae must all be used, and only once, but in any order; relevance logics further accepts contraction, which allows it to reuse formulae in a deduction. Finally, intuitionistic logic accepts all structural rules, and therefore accepts all *constructible* theorems.

3.2. Combinator logics and structurally-free theorem proving

Dunn and Meyer [21] noted that the structural rules can be represented by *combinators*. Combinators are λ-terms with no free variables [4]. We shall represent combinators by capital letters; the choice of letters is historical. Some examples of combinators are

$$\mathsf{B} \equiv \lambda xyz{\cdot}x(yz) \quad \mathsf{B}xyz \twoheadrightarrow x(yz)$$

$$\mathsf{C} \equiv \lambda xyz{\cdot}xzy \quad \mathsf{C}xyz \twoheadrightarrow xzy$$

$$\mathsf{W} \equiv \lambda xy{\cdot}xyy \quad \mathsf{W}xy \twoheadrightarrow xyy$$

The symbol $\twoheadrightarrow$ means 'reduces to' and in the traditional λ-calculus it is replaced by $=$. The right hand-side formula in each box shows that combinators can be defined without λ-abstraction and, in this sense, they become *proper* combinators dissociated from the λ-calculus, as in their original formulation [47].

In Ref. [21] it is proposed a *structurally-free logic* (SFL) where the system is free from any structural presupposition (whence its name). All structural operations have to be accounted for via *combinator rules*, and hence another name for such a logic is *combinator logic*. In the language of such a logic, the combinators are considered as special atomic formulae. Hence, $p \rightarrow (\mathsf{B} \times q)$ is a formula of such a language. The connective rules for such a language are exactly those presented before. However, there are no structural rules in SFL. Instead, we have *connective rules*; a generic combinator rule is

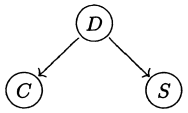
$$\frac{\Gamma[\sigma(\Phi_1,\ldots,\Phi_k)] \vdash \chi}{\Gamma[\mathsf{X},\Phi_1,\ldots,\Phi_k] \vdash \chi}(\mathsf{X} \vdash)$$

(where $Xx_1,\ldots,x_k \twoheadrightarrow \sigma(x_1,\ldots,x_k)$) for a combinator that, when applied to the lower sequent as shown, generates the upper sequent. Some instantiations of the generic

D : reduced plane fares;

S : high-season;

C : chartered planes available.



$p(D) = 0.4$

$p(S|D) = 0.2$

$p(S|\neg D) = 0.7$

$p(C|D) = 0.8$

$p(C|\neg D) = 0.3$

Disconnected variables are (conditionally) independent. So, every conditional probability involving these variables can be inferred from the above. For example:

- $p(S) = p(S|D)p(D) + p(S|\neg D)p(\neg D) = 0.2 \times 0.4 + 0.7 \times 0.6 = 0.5$
- $p(D|S) = \frac{p(D)p(S|D)}{p(S)} = \frac{0.4 \times 0.2}{0.5} = 0.16$
- $p(D, C|S) = \frac{p(D)p(C|D)p(S|D)}{p(S)} = \frac{0.4 \times 0.8 \times 0.2}{0.5} = 0.13$

Fig. 5. Discount belief network (discount-BN).

combinator rule for the combinators presented above are:

Left-associativity *Commutativity* *Contraction*

$$\frac{\Gamma[\Phi,(\Psi,\Xi)] \vdash \chi}{\Gamma[\mathsf{B},\Phi,\Psi,\Xi] \vdash \chi}(\mathsf{B}\vdash) \qquad \frac{\Gamma[\Phi,\Xi,\Psi] \vdash \chi}{\Gamma[\mathsf{C},\Phi,\Psi,\Xi] \vdash \chi}(\mathsf{C}\vdash) \qquad \frac{\Gamma[\Phi,\Psi,\Psi] \vdash \chi}{\Gamma[\mathsf{W},\Phi,\Psi] \vdash \chi}(\mathsf{W}\vdash)$$

Note that above we also show the structural rule associated with the combinator rule.

Combinator rules leave a 'trail' of combinators in a proof, and such combinators are evidence of the structural rules needed for the deduction. Finger [25] proposed the notion of *Structurally-Free Theorem Proving* (SFTP), which can be defined as follows. Given an antecedent Γ and a consequent χ, find a combinator X such that $\mathsf{X}, \Gamma \vdash \chi$ is deducible in combinator logic. Such an activity is a generalisation of traditional theorem proving, because by inspecting the combinators that compose the answer X, it allows us to answer the question: *in which substructural logics is a given sequent deducible.*

3.3. An example: knowledge sharing between relevance and linear logics

Let us suppose the following formulae:

Zurich ← *London*

Barcelona ← *London*

London

Comprise the knowledge-base of a donor system. These propositional formulae entail simple information about routes of flights (first and second formulae) and the origin of the journey (third formula). Let us assume further that the inference engine of such system incorporates a relevant logics [1]. When posed a query ← *Zurich* ⊗ *Barcelona* (that is, *is it possible to go both to Zurich and Barcelona, leaving from London*?) then the donor system prepares the following sequent proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Zurich \vdash Zurich \quad London \vdash London}{Zurich \leftarrow London, London \vdash Zurich}(\leftarrow\vdash) \quad \dfrac{Barcelona \vdash Barcelona \quad London \vdash London}{Barcelona \leftarrow London, London \vdash Barcelona}(\leftarrow\vdash)}{Zurich \leftarrow London, London, (Barcelona \leftarrow London, London) \vdash Zurich \otimes Barcelona}(\vdash\otimes)}{\langle \mathsf{B}_2\rangle, Zurich \leftarrow London, London, Barcelona \leftarrow London, London \vdash Zurich \otimes Barcelona}(\mathsf{B}\vdash)}{\langle \mathsf{C}_2, \mathsf{B}_2\rangle, Zurich \leftarrow London, Barcelona \leftarrow London, London, London \vdash Zurich \otimes Barcelona}(\mathsf{C}\vdash)}{\langle \mathsf{W}_3, \mathsf{C}_2, \mathsf{B}_2\rangle, Zurich \leftarrow London, Barcelona \leftarrow London, London \vdash Zurich \otimes Barcelona}(\mathsf{W}\vdash)$$

The subscript in the combinators indicates the position in the sequent where that combinator was applied: it can be defined purely in terms of combinators as $\mathsf{X}_1 = \mathsf{X}$ and $\mathsf{X}_{i+1} = \mathsf{B}\mathsf{X}_i$. The output of the query to the KBS above consists of the answer that ← *Zurich* ⊗ *Barcelona* holds and the list $\langle \mathsf{W}_3, \mathsf{C}_2, \mathsf{B}_2\rangle$. Recipient systems are able to efficiently evaluate the answer provided: rather than examining the proof, a linear scan of the list of combinators is enough to check for properties of the donor system. In our example, since W is found in the output string of combinators, we conclude that the deduction is not linear. Recipient systems which required that proofs be linear could reject the deduction above just by examining the simplified list of combinators.

Such a method for quickly checking the kind of structural rule, i.e. the control of resources, used by a remote inference system easily generalises to any substructural system. All it is required to do is to send, together with the answer to a query, a combinator list associated with the deduction of the answer to the query.

4. On sharing syntactic × semantic knowledge

As noted in Refs. [15,16], besides having restrictions on the use of deductive systems (Section 3), a semantic mapping on the systems domains is required for certain knowledge sharing. A nice example of this situation can be found in systems for reasoning under uncertainty, where even if we do have a shared ontology for the problem being solved, we must still establish semantic links between the inferences performed within each system to actually have knowledge being shared and reused.

This section presents the issues on having a simple logical system for interval-based probabilistic reasoning being able to consult bayesian belief networks to complete its own inferences. Despite sharing a lot of semantic information, the differences on the internal procedures to construct answers create some difficulties to semantically connect answers from both systems.

Sections 4.1–4.4 introduce the basic definitions of the probabilistic logics and the bayesian belief networks used in our work and how they can be semantically related. A more comprehensive review on these subjects is shown in Ref. [12]. In order to keep the discussion more objective, we provide concrete examples.

4.1. A system for probabilistic reasoning

The sample system for probabilistic reasoning used here[6] employs a resolution-style SLDNF deductive system for clausal theories and can be implemented as a pure Prolog meta-interpreter [2,3,49]. Such a system allows deductions of degrees of belief apportioned by a rational agent to statements represented as normal clauses and queries. These degrees of belief are represented as probabilities on possible worlds, based on the foundations for well-known formalisms like incidence calculus [8,13], probabilistic logic [44] and the Dempster–Shafer theory of evidence [22,48].

This defines a rich subset of first-order logic with a computationally efficient inference procedure and a formally specified declarative semantics. A program II is a theory consisting of a collection of normal clauses $H \leftarrow Q_1,\ldots,Q_n$ and unit clauses $H' \leftarrow$, and proofs will be triggered by a query $\leftarrow Q'_1,\ldots,Q'_m$, where H, H' are atomic predicates and Q_j, Q'_k are atomic predicates or negations of atomic predicates.

A set of possible worlds is a collection of *worlds* (or *states*, or *interpretations*), each of them assigning different truth-values to the formulae in our language. Intuitively, a possible world should be viewed as a conceivable hypothetical scenario upon which we can construct our reasoning. The degree of belief attached to a query is the probability of selecting a possible world at random and the query being true in that world. Following Ref. [43], one way of evaluating the degree of belief is based on solutions obtained for the inner and outer approximations for the probability of a query $\vec{Q} = Q'_1,\ldots,Q'_m$, denoted as $\mathscr{P}_*(\vec{Q})$ and $\mathscr{P}^*(\vec{Q})$ respectively, and calculated as below (complementary definitions are presented in Appendix A):

$$
\begin{aligned}
H \leftarrow Q \quad &\Rightarrow \quad \mathscr{P}_*(H) = \mathscr{P}_*(Q)\\
&\qquad\quad \mathscr{P}^*(H) = \mathscr{P}^*(Q)\\
H \leftarrow \vec{Q}_1, \vec{Q}_2 \quad &\Rightarrow \quad \mathscr{P}_*(H) = max\{0, \mathscr{P}_*(\vec{Q}_1) + \mathscr{P}_*(\vec{Q}_2) - 1\}\\
&\qquad\quad \mathscr{P}^*(H) = min\{\mathscr{P}^*(\vec{Q}_1), \mathscr{P}^*(\vec{Q}_2)\}\\
H \leftarrow \vec{Q}_1 \quad &\Rightarrow \quad \mathscr{P}_*(H) = max\{\mathscr{P}_*(\vec{Q}_1), \mathscr{P}_*(\vec{Q}_2)\}\\
H \leftarrow \vec{Q}_2 \quad &\qquad\quad \mathscr{P}^*(H) = min\{1, \mathscr{P}^*(\vec{Q}_1) + \mathscr{P}^*(\vec{Q}_2)\}\\
H \leftarrow\neg\, Q \quad &\Rightarrow \quad \mathscr{P}_*(H) = 1 - \mathscr{P}^*(Q)\\
&\qquad\quad \mathscr{P}^*(H) = 1 - \mathscr{P}_*(Q)
\end{aligned}
$$

Example 4.1. Let us consider the following KBS_3 for flight reservations within the United Kingdom:

```
          available(From,To)←   exists(X), route(X,From,To)
          available(From,To)←   exists(X), route(X,To,From)
        cheap(From,To,Month)←   discount(From,To,Month), available(From,To)
goodflight(From,To,Month)←      reliableorcheap(From,To,Month), available(From,To)
bestflight(From,To,Month)←      cheap(From,To,Month), reliable(From,To)

   route(1,london,manchester)   =   [1.,1.]          exists(1)   =   [.6,.9]
    route(2,london,edinburgh)   =   [1.,1.]          exists(2)   =   [.5,.8]
     route(3,london,aberdeen)   =   [1.,1.]          exists(3)   =   [.5,.7]
route(4,manchester,aberdeen)    =   [1.,1.]          exists(4)   =   [.5,.7]
 route(5,edinburgh,aberdeen)    =   [1.,1.]          exists(5)   =   [.4,.7]
```

The probability of having flights from a place `X` to `Y` is given by `available(X,Y)`. The probability related to the query ← `available(london,manchester)` is calculated according to the rules above, that is

$$[max\{.6,.0\}, min\{1.,.9\}] = [.6,.9]$$

The probability of having cheap flights, say from London

[6] This is a short version of the system presented in Ref. [14] and implemented in Ref. [11].

Table 1
Structural similarities between logical system and belief network

	Logical system	Belief network
Atomic information	Predicates and ground terms	Boolean variables
relations	Normal clauses	Directed graph edges
Degrees of belief	Probability intervals	Conditional probabilities
A priori information	Intervals for unit clauses	Selected set of marginal probabilities
Output	Intervals for queries	Probabilities given evidential assumptions

Table 2
Operational differences between logical system and belief network

	Logical system	Belief network
Basis for propagation	Logical implication	Probabilistic conditioning
Inference system	Resolution	Bayesian rule
Representation of degrees of belief	Interval based	Single valued

to Manchester in August, i.e. ← `cheap (london,manchester,august)` cannot be calculated in this theory through explicit means since `discount(london, manchester,august)` has no values. This task is deferred to Section 4.4.

4.2. Bayesian belief networks

A Bayesian belief network is a directed acyclic graph in which nodes stand for propositional variables, and edges for the probabilistic relation between them [55]. For instance, an edge from node v_i to node v_j denotes how v_j depends on v_i. On the other hand, the probability of v_i may change as v_j is given as an evidence. A more formal definition of Bayesian belief network is given in Appendix A.

Belief networks are mainly applied to the representation and manipulation of knowledge concerning how evidences support hypotheses. This is denoted by the probability propagation along the network via probability values assigned to nodes and edges, conditional probability (Appendix A).

Example 4.2. The belief network shown in Fig. 5 represents the statistical knowledge on how plane fares are reduced (D) depending on the season (S), and the availability of chartered planes (C).

4.3. Mapping formalisms

Comparing the formalisms for uncertain reasoning presented in the previous sections, we realise their structural similarities, as presented in Table 1. Both systems were created to propagate their (probability-based) representations of degrees of belief along pieces of knowledge. As a result, they contain operators to perform conceptually equivalent tasks, as indicated by their structural similarities (Table 1). In contrast with the above, the procedures employed within each system to effectively propagate degrees of belief are very dissimilar, as presented in Table 2.

To have both systems sharing knowledge, their corresponding capabilities at the **conceptual level** must be defined, namely by mapping their semantics (Fig. 6). The relation must be established as a functional transformation from logical atomic queries to predefined conditional probabilities of the belief network, and then back from the (single valued) probability estimations from the belief network to the (interval based) probability values in the logical system. The interpretation of queries and answers in each system must be compatible with each other, hence the connection must be established at the semantic level.

To link a predicate `p(a)` from the logic system to a pair of evidences from the belief network $(v_1,\ldots,v_m|e_1,\ldots,e_r)$ and $(v_1,\ldots, v_m, v_{m+1},\ldots,v_n|e_1,\ldots,e_r)$, the set of possible worlds selected by the evidences $e_1,\ldots,e_r$ must be disjoint with the set of possible worlds initially used to interpret the program. Also, the algebra and probability measure based on this set of possible worlds must be independent from the other algebras and measures used in the initial interpretation of the program. Under such restrictions, the set of possible worlds, algebras and probability measures considered initially for the program can be extended. Results from the bayesian belief network can then be included as part of the logical proofs.

Regarding the Example 4.1, a semantic relation between the KBS_3 and the discount belief network, presented in Example 4.2, can be established. If the notion of reduced fares are appropriately related to reduced plane fares in KBS_3, the predicate `discount(From,To,Month)` can be inferred from the discount belief network. As the `Month` is given, we can associate high and low-season time. For the query ← `discount(london,manchester,august)`, for example, we may use high season time (S) as evidence. This way, the inner approximation for the query ← `discount(london,manchester,august)` may be given by having discount and chartered planes as high-season is given as evidence. On the other hand, the outer approximation may be given by having reduced fares as

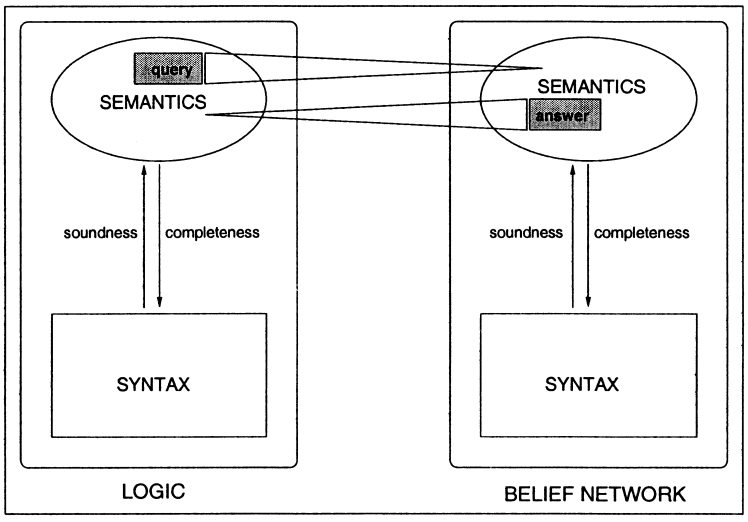


Fig. 6. Connection between two KBSs: conceptual level.

high-season is given as evidence. For the query above, these systems are related as follows:

logic formula	associated interval	result
`discount(london,manchester,august)`	$[p(D,C\|S), p(D\|S)]$	[0.13, 0.16]

4.4. Logical system consulting belief networks

This section presents a range of examples in which the logic system presented above consults one or two belief networks to complete its queries.

Example 4.3. Once the semantic relation between the systems is established, the KBS_3 can use the discount-BN knowledge to complete its inferences. Using the relations shown in Fig. 7 representing the statistical knowledge on reliable flights, based on the size of the air-companies and the weather conditions for the area.

Besides having the notions of cheap fares, a new notion on reliable flights must be established to answer the query ← `bestflight(london,manchester,august)` for example. To do so, the semantic relation must first be established between the query ← `reliable(london, manchester)` and the reliability-BN. The weather conditions is given as evidence since the place to go is well-known. The relation is then defined as follows:

logic formula	associated interval	result
`reliable(london,manchester)`	$[p(R,B\|W_e), p(R\|W_e)]$	[0.38, 0.8]

defined in the previous section, we may now answer the query ← `cheap(london,manchester,august)`:

`available(london,manchester)` = [0.6, 0.9]

`discount(london,manchester,august)`

= [0.13, 0.16]

`cheap(london,manchester,august)`

= [max{0, (0.13 + 0.6 − 1)}, min{0.16, 0.9}] = [0.0, 0.16]

Example 4.4. Now, let us consider another belief network

The query ← `bestflight(london,manchester, august)` can now be answered:

`reliable(london,manchester` = [0.38, 0.8]

`cheap(london,manchester,august)` = [0.0, 0.16]

`bestflight(london,manchester,august)`

= [max{0, (0.38 + 0.0 − 1)}, min{0.8, 0.16}]

= [0.0, 0.16]

Example 4.5. In KBS_3, good flights are measured by either the flight is cheap or the flight is reliable. The query ← `goodflight(london,manchester,august)` can

then be answered by sharing knowledge either from the reliability-BN or from the discount-BN. If the following semantic relations are established:

logic formula	associated interval	network	result
`reliableorcheap(london,manchester,august)`	$[p(R,B\|W_e), p(R\|W_e)]$	*reliability-BN*	[0.38, 0.8]
`reliableorcheap(london,manchester,august)`	$[p(D,C\|S), p(D\|S)]$	*discount-BN*	[0.13, 0.16]

Query ← `goodflight(london,manchester,august)` may have two different results, depending on which BN is being used. If we use reliability-BN, then we have

$$\begin{aligned}
\texttt{available(london,manchester)} &= [0.6, 0.9] \\
\texttt{reliableorcheap(london,manchester,august)} &= [0.38, 0.8] \\
\texttt{goodflight(london,manchester,august)} &= [max\{0, (0.38 + 0.6 - 1)\}, min\{0.9, 0.8\}] = [0.0, 0.8]
\end{aligned}$$

If, on the other hand, we use reliability-BN, then we have:

$$\begin{aligned}
\texttt{available(london,manchester)} &= [0.6, 0.9] \\
\texttt{reliableorcheap(london,manchester,august)} &= [0.13, 0.16] \\
\texttt{goodflight(london,manchester,august)} &= [max\{0, (0.13 + 0.6 - 1)\}, min\{0.16, 0.9\}] = [0.0, 0.16]
\end{aligned}$$

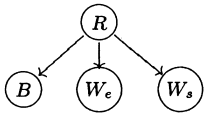
The results presented here take into account only a simple situation in which information provided by the networks is completely missing in the logical system. Additionally, we have not considered how the connection between systems would have to be made so that logical theories could also provide useful information to belief networks. As with the example above, different KBSs could be used to answer the same query. A broker containing the descriptions of the systems capabilities must then be provided. The broker is concerned with accepting queries, formatting them accordingly and providing the means to obtain answers based on the capabilities relation (details on this subject are presented in Section 5).

5. On sharing group knowledge

In previous sections we have shown how knowledge can be shared between pairs of systems under different assumptions about the reasoning mechanisms employed by those systems. The ultimate goal of knowledge sharing, however, requires more than this. It also is necessary that systems which hitherto have not interacted, and which have been designed by separate teams of engineers, should be able to share information. A prerequisite for doing this on a large scale is some mechanism for guessing which of the many systems available are capable of supplying different kinds of information, then acquiring actual information based on these capability descriptions and finally assembling the results to supply the information originally requested. This task is sometimes described as information 'brokering'. In Ref. [46] we describe a simple, compact mechanism based on Horn clause logic for performing this task. The diagram of Fig. 8 (adapted from that paper) describes the general brokering method.

Initially, each KBS describes its capabilities in our capability language (explained later by example). In the diagram we depict these capabilities as the boxed C1 and C2 attached to each of the two KBSs in the example. Capabilities are advertised by sending these to a broker, which records the capabilities and the KBSs who claim to be able to supply them. In Fig. 8 this is shown as a single broker but we could, if we wished, have a distributed system

R : plane is reliable;
B : big air-companies do the service;
W_s : weather conditions in Scotland;.
W_e : weather conditions in England;.



$p(R) = 0.6$

$p(B|R) = 0.6$
$p(B|\neg R) = 0.5$

$p(W_s|R) = 0.8$
$p(W_s|\neg R) = 0.6$

$p(W_e|R) = 0.8$
$p(W_e|\neg R) = 0.7$

Other probability information can be inferred from the above:

- $p(W_e) = p(W_e|R)p(R) + p(W_e|\neg R)p(\neg R) = 0.76$
- $p(R|W_e) = \frac{p(R)p(W_e|R)}{p(W_e)} = \frac{0.6\times0.8}{0.76} = 0.8$
- $p(R,B|W_e) = \frac{p(R)p(B|R)p(W_e|R)}{p(W_e)} = \frac{0.6\times0.6\times0.8}{0.76} = 0.38$

Fig. 7. Reliability belief network (reliability-BN).

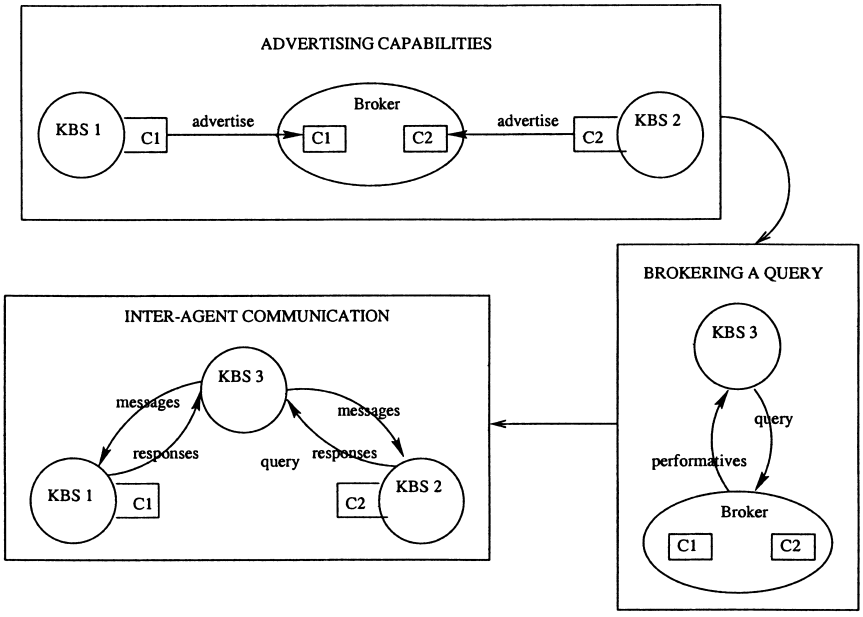


Fig. 8. A capability brokering method.

of brokers — in the most extreme case of a fully distributed agent system there could be a separate broker for each agent. In the next stage, another KBS (in our diagram, KBS 3) sends a query to the broker. The broker constructs from its capability descriptions its internal description, which we call a 'brokerage structure', of how the query might be answered based on those capabilities. It then translates its brokerage structure into a sequence of performative statements describing the messages which it thinks should enable the query to be satisfied by requesting appropriate KBSs to discharge their capabilities. In the final stage this performative information is used by the KBS which sent the query to select which KBSs to contact, to send appropriate messages to them, and await appropriate responses.

The details of the brokering mechanism are given in Ref. [46]. To demonstrate its interaction with the pairwise knowledge sharing of previous sections we return to our earlier examples to demonstrate how we may supply capability descriptions of each KBS and then use these descriptions to broker information.

Our example will involve three KBSs: the flight reservation system from Example 4.1 of Section 4.1; the discount belief network from Fig. 5 of Section 4.2; and the reliability belief network from Fig. 7 of Section 4.4. We first describe a set of capabilities for each of these three systems.

We describe capabilities using Horn clauses of the form $C \leftarrow P_1 \wedge \cdots \wedge P_n$, where C is a query which the system may be capable of answering and each P_i is a query which must be made of another system and the answer supplied in order for an answer to C to be attempted. Where the query for C may be attempted without interacting with other systems we omit the conditions and simply state C. Notice that these capability descriptions are not part of the machinery of inference of the KBS. They only specify which queries the KBS is competent to attempt, without prescribing how (or even if) they will be answered. Capability sets for each of the three example KBSs, accompanied by informal descriptions in English, are given below:

Capabilities for flight reservation KBS. Assessing the chance of flight availability; guessing the chance of a cheap fare if it is supplied information about the likelihood of a discount; guessing the chance of a good flight if it is supplied with the likelihood of a reliable or cheap flight; or suggesting the best flight if given information about the chances of discount and reliability.

$$\left\{\begin{array}{rcl} available(From,To) = P & & \\ cheap(From,To,Month) = P_1 & \leftarrow & discount(From,To) = P_2 \\ goodflight(From,To,Month) = P_1 & \leftarrow & reliableorcheap(From,To,Month) = P_2 \\ bestflight(From,To,Month) = P_1 & \leftarrow & discount(From,To) = P_2 \wedge reliable(From,To) = P_3 \end{array}\right\}$$

Capabilities for discount belief network. Estimating the probability that it is high season; estimating the probability of a reduced plane fare given that it is high season; estimating the probability of a reduced plane fare and chartered planes being available given that it is high season.

$$\left\{\begin{array}{c} p(S) = P \\ p(D|S) = P \\ p(D, C|S) = P \end{array}\right\}$$

Capabilities for reliability belief network. Estimating the probability that weather conditions are good in England; estimating the probability that a plane is reliable given that weather conditions are good in England; estimating the probability that a plane is reliable and that a big air company does the service given that weather conditions are good in England.

$$\left\{\begin{array}{c} p(W_e) = P \\ p(R|W_e) = P \\ p(R, B|W_e) = P \end{array}\right\}$$

Since the ontologies used in the three systems are different we also need to supply definitions of translations between terms in appropriate ontologies - allowing a term used in one system to be understood in the ontology of another. We use $T_1 \approx T_2$ to denote that the term T_1 in one ontology is considered equivalent to the term T_2 in another.[7] The definitions are then:

$$reliableorcheap(From, To, Month) = [P_l, P_u] \approx p(D, C|S) = P_l \wedge p(D|S) = P_u$$

$$reliableorcheap(From, To) = [P_l, P_u] \approx p(R, B|W_e) = P_l \wedge p(R, W_e) = P_u$$

If our broker is supplied with this information then it can construct brokerage structures for queries posed to it that involve the coordinated efforts of all three KBSs. The formal details of this are given in Ref. [46] but, for our current purposes, it suffices to show informally how this is assembled for the query $goodflight(london,manchester,august) = P$. One nested brokerage structure for this is as shown below:

- The capability $goodflight(london,manchester,august) = P_1 \leftarrow reliableorcheap(london,manchester,august)$: P_2 suggests that the query might be answered by the flight reservation KBS provided that it is supplied with $reliableorcheap(london, manchester, august) = P_2$.
- $reliableorcheap(london, manchester, august) = [P_l, P_u]$ can be translated to $p(D, C|S) = P_l \wedge p(D|S) = P_u$.
- $p(D, C|S) = P_l$ is a capability of the discount belief network.
- $p(D|S) = P_u$ is a capability of the discount belief network.

This structure is then easily translated into a sequence of messages which can be sent to the appropriate KBSs. Given the actual inference mechanisms which we saw in previous sections, the behaviour we would expect is as follows:

- The broker asks the discount belief network if it can answer the queries $p(D, C|S) = P_l$ and $p(D|S) = P_u$.
- The answers returned are $p(DC|S) = 0.13$ and $p(D|S) = 0.16$ (see Fig. 1).
- The broker then applies the translation: $reliableorcheap(london,manchester,august) = [0.13,0.16] \approx p(D,C|S) = 0.13 \wedge p(D|S) = 0.16$.
- It now informs the flight reservation KBS of its belief that $reliableorcheap(london, manchester, august) = [0.13, 0.16]$ and asks it to supply an answer for $goodflight(london, manchester, august) = P_1$ given this information.
- The answer is $goodflight(london, manchester, august) = [0.13, 0.16]$.

6. Conclusions and discussion

Ontologies are useful in aiding knowledge sharing. However, as described in our investigation, many other important issues arise which cannot be dealt with simply with ontologies. Other techniques have to be investigated and combined in order to solve practical problems inherent in knowledge sharing. In this paper we have discussed a number of problems related with employing ontologies to foster knowledge sharing and suggested alternative solutions to them. We have exploited the following aspects:

1. *Reusing ontologies to engineer new ontologies* — Ontologies enthusiasts claim that reuse is a useful feature of ontologies. However, we have shown via an actual experiment in which we tried to build a new ontology reusing a library of existing ontologies, that reuse is not straightforward and many practical issues arise. We have presented and assessed techniques to cope with such issues.
2. *Sharing inferences*. — When we want to share not the declarative knowledge of a KBS but its inferences, then ontologies fall short in providing adequate solutions. If not all kinds of inferences provided by one KBS are accepted by another system, that is, a situation of *partial knowledge* sharing arises, then it is required that the inferences be examined by the receiving system. Since inferences may be very large, the problem of examining them has to be given a computationally feasible solution. We have proposed an efficient way to build and economically represent proofs by means of *combinator logics* so as to enable their efficient interchange and analysis by the KBSs sharing their knowledge.
3. *Sharing semantic knowledge*. — In systems for reasoning under uncertainty, even when we have a shared ontology,

[7] Strictly we should name the ontologies in these definitions but we leave this detail out of our example.

we must still establish *semantic* links between the inferences performed within each system in order to appropriately share and reuse knowledge. We have studied a significant instance of this problem, in which a logical system for interval-based probabilistic reasoning is made to consult a bayesian belief network to complete its inferences. The difficulties arising are pointed out and we show how these can be remedied.

4. *Sharing group knowledge* — When knowledge sharing is to take place in a large scale, mechanisms are required for guessing which of the many systems available are capable of supplying different kinds of information. We have investigated a simple alternative for a mechanism that acts as a knowledge broker in which the capabilities of systems are advertised (as Horn clauses) and consulted by the requesting systems. We illustrate the workings and the usefulness of our proposal with a simple motivating example.

This paper does not claim to be exhaustive in pointing out issues in knowledge sharing which are not accounted for by ontologies. We have compiled a list of our own experiences to illustrate our point. It is our hope that more case studies appear in the literature to guide practitioners in the task of (re-)engineering KBSs for knowledge sharing.

Acknowledgements

This article presents results of research project *Distributed Environment for Cooperation Among Formalisms for Knowledge-based Systems* (DECaFf-KB), sponsored by the Brazilian Research Sponsoring Agency CAPES and the British Council. Flavio S. Correa da Silva was partially sponsored by the Brazilian Research sponsoring Agency FAPESP, Grant no.93/0603-01. Wamberto W. Vasconcelos was on a Post-Doctoral leave of absence from Departamento de Estatística e Computação, Universidade Estadual do Ceará, Ceará, Brazil, sponsored by the Brazilian Research Council CNPq, grant no. 201340/91-7. Virginia Brilhante was on leave of absence from Departamento de Ciência da Computação, Universidade do Amazonas, Brazil, sponsored by CAPES, Grant no. BEX 1498/96-7. Ana C.V. de Melo was partially sponsored by the Brazilian Research Sponsoring Agency FAPESP, Grant no. 93/0603-01. Marcelo Finger was partially sponsored by CNPq, Grant no. 300597/95-9. Flavio S. Corea da Silva was partially sponsored by the Brazilian Research Sponsoring Agency FAPESP, grant no. 93/0603-01, and by an ACI grant of the Generalitat de Catalunya. Jaume Agustí was partially sponsored by an ACI grant of the Generalitat de Catalunya.

Appendix A. Basic facts on probability theory and bayesian belief networks

Given a finite set D, an algebra χ_D on D is a set of subsets of D such that

- $D \in \chi_D$;
- $A \in \chi_D \Rightarrow \neg A \in \chi_D$;
- $A, B \in \chi_D \Rightarrow A \cup B \in \chi_D$.

A subset of D is called an *event* on D. Events belonging to χ_D are called *measurable events*. The basis χ'_D of an algebra χ_D is the subset of χ_D such that

- $\{\} \notin \chi'_D$;
- $A, B \in \chi'_D \Rightarrow A \cap B = \{\}$;
- $K \in \chi_D \Rightarrow \exists A_1, \ldots, A_n \in \chi'_D : K = \bigcup_1^n A_i$.

A *probability measure* on χ_D is a function $\mathscr{P}{:}\chi_D \rightarrow [0,1]$ such that

- $P(D) = 1$ (total probability);
- $A \cap B = \{\} \Rightarrow \mathscr{P}(A \cup B) = \mathscr{P}(A) + \mathscr{P}(B)$ (finite additivity).

Given two measurable events $A, B \in \chi_D$, *the conditional probability* $\mathscr{P}(A|B)$ is defined as:

$$\mathscr{P}(A|B) = \begin{cases} \dfrac{\mathscr{P}(A \cap B)}{\mathscr{P}(B)}, & \mathscr{P}(B) \neq 0 \\ 0, & \mathscr{P}(B) = 0 \end{cases}$$

Two measurable events A, B are called *independent iff* $\mathscr{P}(A|B) = \mathscr{P}(A)$ which, as a corollary, gives that $\mathscr{P}(A \cap B) = \mathscr{P}(A) \times \mathscr{P}(B)$. Probability measures can be extended to *non-measurable events*, i.e. sets $A_j \in 2^D \backslash \chi_D$. Given D, χ_D and P, we define the *inner and outer extensions* to $\mathscr{P}$ ($\mathscr{P}_*$ and $\mathscr{P}^*$, respectively) as [20]:

- $\mathscr{P}_*, \mathscr{P}^* : 2^D \rightarrow [0,1]$
- $\mathscr{P}_*(A) = sup\{\mathscr{P}(X) : X \subseteq A, X \in \chi_D\} = \mathscr{P}(\cup X : X \subseteq A, X \in \chi'_D)$
- $\mathscr{P}^*(A) = inf\{\mathscr{P}(X) : A \subseteq X, X \in \chi_D\} = \mathscr{P}(\cup X : X \cap A \neq \{\}, X \in \chi'_D)$

A bayesian belief network [55] is a tuple $B = (G, P)$ where:

- $G = (VG, AG)$ is an acyclic digraph with nodes $VG = \{V_1, \ldots, V_n\}$, $n \geq 1$, and edges AG;
- $P = \{pV_i | V_i \in VG\}$ is a set of real-valued non-negative functions

  $pV_i : \{C_{V_i}\} \times \{C_{\rho G(V_i)}\} \rightarrow [0,1]$

  called (conditional) probability assessment functions, such that for each configuration $c_{\rho G}(V_i)$ of the set $\rho_G(V_i)$ of (immediate) predecessors of node V_i in G, we have that $p_{V_i}(\neg v_i | c_{\rho G(V_i)}) = 1 - p_{V_i}(v_i | c_{\rho G(V_i)})$.

References

[1] A.R. Anderson, N.D. Belnap Jr., Entailment: The Logic of Relevance and Necessity, vol. 1, Princeton University Press, Princeton, 1975.
[2] K.R. Apt, Logic programming, in: J. van Leeuwen (Ed.), Handbook of

Theoretical Computer, Elsevier — MIT Press, Amsterdam — Cambridge, 1994, pp. 493–574.
[3] K.R. Apt, From Logic Programming to Prolog, Prentice-Hall, UK, 1997.
[4] H.P. Barendregt, The Lambda Calculus: Its Syntax and Semantics, Number 103 in North-Holland Studies in Logic and the Foundation of Mathematics, Elsevier, Amsterdam, 1981.
[5] V. Brilhante, Using formal data descriptions for ecological modelling guidance, Discussion Paper 197, Divison of Informatics, University of Edinburgh, 1998.
[6] V. Brilhante, Using formal meta-data descriptions for automated ecological modeling, Papers from the AAAI-99 Workshop on Environmental Decision Support Systems and Artificial Intelligence WS-99-07, AAAI, AAAI Press, Menlo Park, CA, 1999.
[7] V. Brilhante, D. Robertson, in: C. Rautenstrauch (Ed.), Environmental Information Systems in Industry and Public Administration, Idea Group Publishing, Hershey, PA, 2001 chapter Metadata-supported Automated Ecological Modelling, in press.
[8] A. Bundy, Incidence calculus: a mechanism for probabilistic reasoning, Journal of Automated Reasoning 1 (3) (1985) 263–284.
[9] M. Cerioli, J. Meseguer, May I borrow your logic? (transporting logical structures along maps), Theoretical Computer Science 173 (1997) 311–347.
[10] P. Cohen, V. Chaudhri, A. Pease, R. Schrag, Does prior knowledge facilitate the development of knowledge-based systems? Proceedings of the 16th International Conference on Artificial Intelligence — AAAI-99, AAAI Press, 1999 pp. 221–226.
[11] F.S. Correa da Silva, Automated reasoning with uncertainties, PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.
[12] F.S. Correa da Silva, R.C. Araujo, J. Agustí, A.C.V. Melo, Knowledge sharing between a probabilistic logic and bayesian belief networks, Proceedings of the International Conference on Processing and Management of Uncertainty, 2000.
[13] F.S. Correa da Silva, A. Bundy, A rational reconstruction of incidence calculus, Technical Report 517, Department of Artificial Intelligence, University of Edinburgh, 1991.
[14] F.S. Correa da Silva, D.S. Robertson, J. Hesketh, Automated Reasoning with Uncertainties, Knowledge Representation and Uncertainty, Volume 808 of Lecture Notes in Artificial Intelligence, Springer, Berlin, 1994 Chapter 5.
[15] F.S. Correa da Silva, W.W. Vasconcelos, D.S. Robertson, Cooperation between knowledge-based systems, Proceedings of the IV World Congress on Expert Systems, Mexico City, Mexico, 1998, pp. 819–825.
[16] F.S. Correa da Silva, W.W. Vasconcelos, D.S. Robertson, J. Agustí, A.C.V. Melo, Why Ontologies are not Enough for Knowledge Sharing, Lecture Notes in Artificial Intelligence, vol. 1611, Springer, Berlin, 1999 pp. 520–529.
[17] D. Van Dalen, Intuitionistic logic, in: D. Gabbay, F. Guenthner (Eds.), Handbook of Philosoph. Log., vol. III: Alternatives to Classical Logic, Kluwer Academic Publishers, Dordrecht, The Netherlands. 1984.
[18] J.P. Delgrande, J. Mylopoulos, Knowledge representation: features of knowledge, in: W. Bibel, P. Jorrand (Eds.), Fundamentals of Artificial Intelligence: An Advanced Course, LNCS 232 Springer, Berlin, 1986.
[19] K. Došen, A historical introduction to substructural logics, in: P.S. Heister, K. Došen (Eds.), Substructural Logics, Oxford University Press, Oxford, 1993, pp. 1–31.
[20] R.M. Dudley, Real Analysis and Probability, Wadsworth and Brooks/Cole, Belmont, CA, 1989.
[21] J.M. Dunn, R.K. Meyer, Combinators and structurally free logic, Logic Journal of the IGPL 5 (4) (1997) 505–538.
[22] R. Fagin, J.Y. Halpern, Uncertainty, belief, and probability, Technical Report RJ-6191, IBM Research Report, 1989.
[23] A. Farquhar, Ontolingua to Prolog syntax translation, http://www.ksl.Stanford.EDU/people/axf/ol-to-prolog.txt, 1995.
[24] A. Farquhar, R.E. Fikes, J. Rice, The Ontolingua Server: a tool for collaborative ontology construction, Technical Report KSL-96-26.w6, Stanford University, 1996, Web-page available at http://www-ksl-svc.stanford.edu.
[25] M. Finger, Towards structurally-free theorem proving, Logic Journal of the IGPL 6 (3) (1998) 425–449.
[26] M. Finger, W.W. Vasconcelos, Sharing resource-sensitive knowledge using combinator logics, Proceedings of SBIA/IBERAMIA'2000, LNAI, 2000.
[27] T. Finin, M. Cutkosky, T. Gruber, J. van Baalen, Knowledge sharing technology project overview, Technical Report KSL-91-71, Stanford University, November 1991.
[28] T. Finin, D. McKay, R. Fritzon, An overview of KQML: a knowledge query and manipulation language, Technical report, University of Maryland Baltimore County, March 1992.
[29] M.R. Genesereth, R.E. Fikes (Eds.), Knowledge Interchange Format, version 3.0 Reference Manual Computer Science Department, Stanford University, 1992 Report Logic 92-1.
[30] J.Y. Girard, Linear logic, Theoretical Computer Science 50 (1987) 1–102.
[31] J.A. Goguen, R.M. Burstall, Institutions: abstract model theory for specification and programming, Journal of the ACM 39 (1992) 95–146.
[32] P. Gray et al., KRAFT — Knowledge reuse and fusion/transformation, http://www.csd.abdn.ac.uk/apreece/Research/KRAFT/KRAFTinfo.html, 1998.
[33] T.R. Gruber, A. Translation, Approach to portable ontology specifications, Knowledge Acquisition 5 (2) (1993) 199–220.
[34] T.R. Gruber, G.R. Olsen, An ontology for engineering mathematics, Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning, Bonn, Germany, Morgan Kaufman, Los Altos, CA, 1994.
[35] N. Guarino (Ed.), Formal Ontology in Information Systems, Volume 46 of series Frontiers in Artificial Intellligence and Appliances. IOS Press, Amsterdam, 1998.
[36] J.S. Hodas, D. Miller, Logic programming in a fragment of intuitionistic linear logic, Information and Computation 110 (2) (1994) 327–365.
[37] P. Jackson, Introduction to Expert Systems, Volume 46 of series Frontiers in Artificial Intelligence and Applications, Addison-Wesley, Reading, MA, 1999.
[38] Intelligent Systems Laboratory, SICStus Prolog User's Manual, Swedish Institute of Computer Science, available at http://www.sics.se/isl/sicstus2.html#Manuals, February 2000.
[39] Y. Labrou, T. Finin, A proposal for a new KQML specification, Technical report, University of Maryland Baltimore County, February 1997.
[40] J.W. Lloyd, Foundations of Logic Programming, Springer, Berlin, 1993 second, extended edition.
[41] E. Motta, Reusable Components for Knowledge Modelling, IOS Press, 1999.
[42] R. Neches, D. Gunning, The knowledge sharing effort, http://www-ksl.stanford.edu/knowledge-sharing/papers/kse-overview.html, 1999.
[43] R. Ng, V.S. Subrahmanian, Probabilistic Logic Programming, Information and Computation (1992).
[44] N.J. Nilsson, Probabilistic logic, Artificial Intelligence 28 (1986) 71–87.
[45] OMG and X/Open, The common object request broker: architecture and specification, Technical Report Revision 2.0, July 1995 — updated July 1996, The Object Management Group, Inc. and X/Open, Co Ltd., 1996.
[46] D. Robertson, F. Correa da Silva, J. Agusti, W. Vasconcelos, A lightweight capability communication mechanism, Proceedings of the 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, New Orleans, Louisiana, 2000.
[47] A. Schönfinkel, Über die Bausteine der Mathematischen Logik, in: J. van Heijenoort (Ed.), From Frege to Gödel, Harvard University Press, Cambridge, MA, 1924 Reprinted.

[48] G. Shafer, A Mathematical Theory of Evidence, Princeton University Press, Princeton, 1976.
[49] L. Sterling, E. Shapiro, The Art of Prolog, MIT Press, Cambridge, MA, 1986.
[50] V.S. Subrahmanian, HERMES — a heterogeneous reasoning and mediator system, http://www.cs.umd.edu/projects/hermes/index.html, 1999.
[51] B. Swartout, R. Patil, K. Knight, T. Russ, Toward distributed use of large-scale ontologies, Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop — KAW-96, Banff, Canada, 1996.
[52] M. Uschold, M. Gruninger, Ontologies: principles, methods and applications, Knowledge Engineering Review 11 (2) (1996) 93–136.
[53] M. Uschold, Where are the killer apps? Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods — ECAI-98, Brighton, UK, 1998.
[54] M. Uschold, M. Healy, K. Williamson, P. Clark, S. Woods, Ontology reuse and application, Proceedings of the First International Conference on Formal Ontology in Information Systems — FOIS'98, Trento, Italy, IOS Press, 1998 pp. 179–192.
[55] L. van der Gaag, Bayesian belief networks: odds and ends, The Computer 39 (1996) 97–113.
[56] M. Winikoff, J. Harland, Implementation and development issues for the linear logic programming language Lygon, Proceedings of the VIII Australasian Computer Science Conference, Adelaide, Australia, February 1995, pp. 562–573.