Verifying Ontological Commitment in Knowledge-Based Systems*

Andrew Waterson and Alun Preece

University of Aberdeen, Computing Science Department
Aberdeen AB24 3UE, Scotland
Phone: +44 1224 272296; FAX: +44 1224 273422
Email: {awaterso, apreece}@csd.abdn.ac.uk

Abstract

An ontology defines the terminology of a domain of knowledge: the concepts that constitute the domain, and the relationships between those concepts. In order for two or more knowledge-based systems to interoperate — for example, by exchanging knowledge, or collaborating as agents in a co-operative problem-solving process — they must commit to the definitions in a common ontology. Verifying such commitment is therefore a prerequisite for reliable knowledge-based system interoperability. This paper shows how existing knowledge base verification techniques can be applied to verify the commitment of a knowledge-based system to a given ontology. The method takes account of the fact that an ontology will typically be expressed using a different knowledge representation language to the knowledge base, by incorporating translation into the verification procedure. While the representation languages used are specific to a particular project, their features are general and the method has broad applicability.

KNOWLEDGE REUSE AND ONTOLOGIES

The reuse and sharing of knowledge bases is a central theme of knowledge engineering in the 1990s [1]. Whereas, in the 1980s, organisations focused upon the construction of stand-alone knowledge-based systems, a significant amount of current interest lies in integrating existing knowledge bases together into enterprise-wide resources. Such resources play a vital role in modern organisations, relating to the ideas of enterprise modelling and business process reengineering [2].

There are two primary ways in which organisations seek to reuse and integrate existing knowledge bases:

- *Knowledge fusion:* Incorporation of existing knowledge into a new knowledge base, or merging of existing knowledge bases into a combined resource. Data warehousing is an example of this kind of approach [3].
- *Distributed knowledge-based systems:* Interoperation of existing knowledge-based systems (or "agents"), distributed as nodes on a network. An example of this approach is the European ARCHON architecture [4].

Early work on enabling technology for knowledge sharing established that three components are needed to allow knowledge to be shared between two knowledge bases [1]:

- a common protocol in which to communicate knowledge;
- a common language in which to express knowledge;

*This work was supported by the University of Aberdeen Research Committee. This paper is an extended version of the paper "Knowledge Reuse and Knowledge Validation", presented at the *AAAI-97 Workshop on Verification and Validation of Knowledge-Based Systems*, chaired by Grigoris Antoniou (Griffith University, Australia) and Robert Plant (University of Miami, USA).

- a common set of definitions of terminology — an *ontology*.

A great deal of work has been done to define common protocols and languages for the communication and expression of knowledge, the best-known being the KQML protocol [5] and the KIF language [6] produced by the Knowledge Sharing Effort (KSE) project [1]. In many ways, the definition of ontologies is a more difficult problem, because there are many different domains in which terminology must be defined. These include:

- *domain terminology* for the application domain(s) that the knowledge refers to, for example, medicine, aerospace, or commerce;
- *task terminology* for the operational aspects of the knowledge-based systems, for example, diagnosis, scheduling, or design;
- *physical terminology* describing the nature of reality underpinning the knowledge, including time, space, and part-whole relations.

Approaches to building ontologies range from large-scale work in defining highly-reusable ontologies of "commonsense" knowledge [7] to more modest efforts in defining terminology in a specific application area [8].

Ontologies and Ontological Commitment

Although precise definitions of an ontology differ, the most widely held view in the artificial intelligence community is that an ontology is an explicit specification of a *conceptualisation*: "the objects, concepts and entities that are assumed to exist in some area of interest, and the relationships that hold among them" [9]. Ontologies may be expressed using informal or semi-formal specification languages, but for our purposes we are interested only in ontologies defined *formally* in an appropriate knowledge representation language (to permit their manipulation within knowledge-based systems).

As a minimum, an ontology will define taxonomic relationships (informal example: "a student is a person"); more generally, any constraints may be put on terms (informal example: "all students must take at least one course"). It is worth noting that, although the *purpose* of an ontology is to define terminology, the *form* of an ontology can be that of a knowledge base or database conceptual schema; any suitably expressive knowledge representation language or database schema definition language may be used to define an ontology.

The word ontology is borrowed from philosophy where ontology is the study of existence, or a description of what does exist [10]. Any logical theory has its own implicit ontology, this ontology consists of all the things that the theory assumes to exist. A logical theory is said to be *ontologically committed* to the existence of all the entities in its ontology.

While there is a considerable volume of literature about ontologies the subject of ontological commitment has received little attention. Ontological commitment was first discussed from a logical and philosophical point of view by Quine [11]. The work that has been done is generally highly philosophical [11, 12, 13, 14]. Nicola Guarino states that ontological commitment can be seen as "a mapping between a language and something which can be called an ontology" [14]. Quine's ontological commitment [11] required each term in a logical theory to be in that theory's ontology. A sentence dealing with an entity that doesn't exist is meaningless. From Quine's perspective, each logical theory has its own explicit or implicit ontology. However, from the point of view of knowledge engineering, interest lies in ontologies which many knowledge-bases can commit to: *portable ontologies* [9]. It would clearly be impractical to limit a knowledge base to terms which exist in external ontologies, therefore the knowledge engineers' definition of ontological commitment must differ from the philosophers'. The following definition is proposed for ontological commitment: "A formalised mapping between terms in a knowledge-base and identical or equivalent terms in an ontology"

Knowledge Reuse Using Ontologies

There are two requirements to share knowledge between two knowledge bases:

- it must be possible to translate their knowledge representations into a common language;

- it must be possible to map their terminologies into a common ontology.

The first requirement may be accomplished using a set of *translation rules*; the second may be accomplished using a set of *mapping rules*. Note that there does not have to be a single common language and a single common ontology; however, if multiple common languages and ontologies exist, there will need to be multiple sets of translation and mapping rules. The translation problem is not hard if knowledge bases use a syntactically-sugared version of first-order predicate calculus, which is the approach taken by the KSE project [9], and is assumed for the purposes of this paper.

A set of mapping rules between a knowledge base and an ontology defines an *ontological commitment* of the knowledge base. It is highly desirable that this ontological commitment be consistent: *no constraint in the ontology should be in conflict with inferences derivable from the knowledge base, and vice versa.* Checking that an ontological commitment is consistent is a verification issue. It is worth noting that there is no completeness requirement on ontological commitment: it is not necessary for every term in the knowledge base to have an equivalent term in the ontology, but in that case there will be some statements that cannot be shared. Similarly, there is no need to have an equivalent knowledge base term for every term in the ontology.

VERIFYING ONTOLOGICAL COMMITMENT

The previous section highlights the fact that, in order for two or more knowledge-based systems to interoperate — for example, by exchanging knowledge, or collaborating as agents in a co-operative problem-solving process — they must commit to the definitions in a common ontology. Verifying such commitment is therefore a prerequisite for reliable knowledge-based system interoperability. The objective of the work described in this paper is to provide a method for accomplishing such a verification task. In doing so, since there already exist effective verification tools for knowledge-based systems, it is desirable to reuse and adapt such tools rather than "reinvent the wheel". Moreover, any realistic verification procedure must take account of the fact that an ontology will typically be expressed using a different knowledge representation language to the knowledge base.

The solution proposed in this paper has the following features:

- The input to the verification process consists of a given knowledge-based system, an ontology, and a set of translation and mapping rules.
- The goal of the verification process is to identify any ways in which the concepts and relations in the knowledge base of the knowledge-based system conflict with the definitions in the ontology, taking into account the equivalences specified by the mapping rules.
- The verification process incorporates an "off the shelf" knowledge base verification tool — COVER [15] — which has been extended to incorporate the translation and mapping operations into the verification process. The extended tool is called DISCOVER (COVER for DIStributed knowledge bases).
- To use the DISCOVER tool, ontologies must be expressed in the MOVES language (Meta-Ontology for the Verification of Expert Systems). The features of MOVES are very common in ontology representation languages, so it would not be difficult to extend the applicability of DISCOVER.
- As DISCOVER incorporates COVER, it requires knowledge bases to be expressed in CRL (COVER Rule Language); again, this uses generic representation features, to allow broad applicability.

The COVER tool is an *anomaly checker*: it analyses a knowledge base for undesirable properties including conflicting knowledge, redundant knowledge, and deficient knowledge. Many of its features are also provided by other anomaly checking tools — see [16] for a comparative survey of such tools. DISCOVER does not change any of COVER's original functionality; it employs COVER's "verification engine" to check for logical anomalies that occur between three kinds of statement:

- rules in the knowledge base of a knowledge-based system;
- rules in an ontology;

- mapping rules between the knowledge base and ontology.

The logical anomalies that indicate incorrect ontological commitment are similar to those that indicate a faulty knowledge base. However, the location of each rule (knowledge-base, ontology, or mapping) must be taken into account in identifying an anomaly. As in the verification of stand-alone knowledge-based systems, not every anomaly signifies an error (see [17] for a full discussion of this issue), but in the cases where DISCOVER does reveal an error, there are several possible sources:

- faulty statements in the knowledge base;
- faulty definitions in the ontology;
- faulty mapping rules between the knowledge base and ontology.

A special case of this third source is where the knowledge base is not actually intended to commit to the ontology in the particular way defined by the faulty mapping rules. Nevertheless, it is important to identify this problem, to avoid an attempt to "reuse" erroneously-mapped knowledge.

DISCOVER is applicable for use in a variety of software development processes. It may be used in the case where there are several existing knowledge sources for which an organisation needs to develop a common ontology; in this case, DISCOVER is likely to identify errors in the ontology as it is developed to be compatible with each knowledge source (or it may identify errors in specific mapping rules between the ontology and a particular knowledge source) . Alternatively, it may be used in the case where a new knowledge source is being constructed within an existing enterprise knowledge framework; in this case, DISCOVER is likely to identify errors in the knowledge base (or the mapping rules), where it conflicts with the common ontology for the enterprise.

It is worth noting that DISCOVER is intended as a tool for a more general purpose than just verifying ontological commitment: it is really intended as a general-purpose tools for checking the compatibility of distributed knowledge sources (hence the name). Another specific uses of DISCOVER in this context are:

- to use an ontology as a body of *background knowledge* against which a knowledge base can be validated;
- to employ knowledge from other sources to validate a knowledge base; for example, extracting items in a database for use as test cases, provided that the database commits to a common ontology with the knowledge base.

The remainder of this paper is organised as follows:

- description of the MOVES ontology representation language;
- description of the translation procedure and mapping rules;
- description of the DISCOVER verification process;
- examination of what the various kinds of anomalies (conflict, redundancy, and deficiency) mean when they involve ontological commitments.

As a running example, the paper describes how an ontology containing university terms, the "University Ontology" can be used to verify a knowledge base, the "University Knowledge Base", that *commits* to it. These examples are inspired by the knowledge base from [18].

REPRESENTATION LANGUAGES

Knowledge bases are developed in many different languages. For an automatic verification tool to be useful to a wide community of knowledge engineers it must be able to verify knowledge bases implemented in as many languages as possible. Ontologies are also represented in a number of different languages, for example, KIF [6] and CycL [7]. Knowledge base representation languages provide features designed to support problem-solving methods, the most fundamental being *inference rules* for use in forward or backward chaining. Ontology representation languages provide features designed to support the definition of terminology: concepts in the form of *frames* (in the AI sense) or

classes (in the object-oriented sense), simple relationships in the form of *slots* on frames or *attributes* on classes, and complex relationships in the form of general *constraint expressions*.

DISCOVER takes account of this necessary heterogeneity of representation languages by accepting knowledge base and ontology statements in different languages. Because DISCOVER incorporates COVER, the COVER Rule Language (CRL) is used to represent knowledge base rules and facts [15]. CRL was designed to be a generic knowledge base representation language based on first-order logic, and translators exist between CRL and several commercially-used knowledge representation languages (including CLIPS and Expertech). DISCOVER accepts ontological statements (terminology and constraints) expressed in the MOVES language, which has been designed to incorporate the most common features of languages used to represent ontologies. It would not be difficult to translate definitions from ontologies represented in other languages into MOVES, for use by DISCOVER (alternatively, it would be possible to extend DISCOVER to accept other ontology languages).

Internally, DISCOVER translates MOVES statements into CRL "on the fly", in order to exploit the core COVER verification mechanisms. In our running example, the University Ontology is implemented in MOVES, as described below. Later, it is shown how MOVES is translated into CRL for use by DISCOVER. The University Knowledge Base is expressed in CRL, described later in this section.

MOVES

MOVES includes a sub-language for expressing terminology and simple relationships using a frame-based notation, and a sub-language for expressing constraints on the terminology.

The terminological sub-language is essentially a hierarchical frame-based system with multiple inheritance for knowledge representation. The language is loosely based on CycL [7], and also incorporates elements of the P/FDM object database schema definition language employed in the KRAFT knowledge sharing project [19]. Thus, its features are rather general and conservative.

Concepts in the ontology are represented by *frames*, and specific examples of these concepts are represented by *instances*. Frames have slots assigned to them. A slot can hold any simple type (integer, real, string or boolean) or can contain an instance of another frame. All frames except the root frame are subtypes of other frames. The root frame is declared to be a subtype of itself. A frame inherits all slots of its parent frames. Frames are declared using the `frame` statement as shown below in an example from the University Ontology:

```
frame thing of thing.
frame university of thing.
frame department of thing.
frame person of thing.
frame facultyMember of person.
frame student of facultyMember.
frame staff of facultyMember.
frame researchStudent of student.
```

`Figure 1: The University Ontology Frame Hierarchy --- Insert here`

These MOVES declarations create the hierarchy of frames shown in Figure 1. To assign slots to these frames the `hasSlot` function is used.

```
person hasSlot name of boolean.
person hasSlot age of integer.
person hasSlot hasDegree of boolean.
person hasSlot enrolled of boolean.

facultyMember hasSlot uni of university.
facultyMember hasSlot dep of department.

staff hasSlot tenured of boolean.

researchStudent hasSlot supervisor of staff.
```

In the above example from the University Ontology the slots `name`, `age`, `hasDegree`, and `enrolled` are assigned to the frame `person`, and are therefore inherited by the frames `facultyMember`, `staff`, `student` and `researchStudent`. The frame `facultyMember` has the slots `uni` and `dep` assigned to it; these slots are inherited by the frames `staff`, `student`, and `researchStudent`. The frame `staff` has the slot `tenured` assigned to it, and `researchStudent` has the slot `supervisor` assigned to it; neither of these slots are inherited by other frames since neither `staff` nor `researchStudent` have sub-frames.

Once frames and their slots have been declared, instances of these frames can be declared.

```
instance alun of staff.
instance andrew of researchStudent.
```

The declarations above create one instance of `staff` — `alun`; and the instance of `researchStudent` — `andrew`. (The appearance of instances in an ontology may seem surprising, but often it is useful to incorporate *examples* of instantiated concepts in an ontology [20].)

Once instances have been created values can be assigned to their slots using the hasValue function, which operates on an object-attribute-value triple.

```
hasValue(alun, hasDegree, yes).
hasValue(andrew, supervisor, alun).
```

The first of these statements assigns the value `yes` to slot `hasDegree` of `staff` — `alun`. The second statement assigns the instance of `staff` — `alun` to the slot `supervisor` of `student` — `andrew`.

Statements in the MOVES constraint sub-language are based upon the CoLan constrain language used in the KRAFT project [19], and take the following form:

```
constraint <frame_selector>
    [where <instance_selector>]
    tohave <condition>.
```

MOVES constraints have either two or three parts: the `frame selector`, the optional `instance selector`, and the `condition`. The frame selector binds frames to variables. The instance selector selects instances to which the constraint is relevant. The condition is a predicate which must be true for all possible combinations of relevant instances. An example constraint from the University Ontology:

```
constraint
    x isa researchStudent and
    y isa staff
where
    slotValue(x, supervisor, y)
tohave
    slotValue(y, uni, var u) and
    slotValue(x, uni, u).
```

The keyword `where` is used to separate the frame selector from the instance selector, and `tohave` is used to separate the instance selector from the condition. Also note that `var` is used to declare `u` as a variable. A constraint is unsatisfied if an instantiation of variables causes the instance selector to evaluate to *true*, and the condition to evaluate to *false*.

The `slotValue` function takes three arguments: an instance, a slot associated with that instance, and a simple expression. If the value contained in the slot matches the simple expression, the `slotValue` function evaluates to `true`. If `var` is used in a simple expression, a variable is created and the value associated with the selected slot is assigned to the variable (see appendix A for further details of syntax).

The example constraint from the University Ontology says that, for the ontology to be consistent, all research students must be a member of the same university as their supervisors. The frame selector binds the variables `x` and `y` to the `researchStudent` and `staff` frames respectively; the instance selector states that the condition applies only to instances of `x` and `y` such that `y` is the supervisor of `x`; finally, the condition states that where instance `y` of `staff` is a member of university `u`, instance `x` of `researchStudent` must also be a member of university `u`.

It is also possible to declare frames as being disjoint with each other as follows:

```
<frame> disjoint_with <frame>.
```

Two frames are said to be disjoint if there is no frame which is a direct or indirect subclass of both frames. For example:

```
staff disjoint_with student.
```

As will become clear in the next section, constraints in the ontology are of particular importance in the verification process, as they place strong restrictions on the ways a commiting knowledge base can use the equivalent terminology defined in the ontology.

COVER Rule Language

The COVER Rule language (CRL) is fully described in [15]; this section will merely summarise its features to demonstrate that they are common to most knowledge-based system implementation languages. CRL statements take four forms.

Rules CRL rules take the form:

```
rule <id> ::
    <consequent> if
    <antecedent>.
```

For example:

```
rule 120 ::
    researchStudent hasValue yes if
    student hasValue yes and
    takesCourses hasValue no.
```

Rule consequents assign values to data items. These data items may be single or multiple-valued, or boolean.

Goal declarations CRL goal declarations take the form:

```
goal <data_item>.
```

For example:

```
goal getDegree.
```

These declarations specify which data items the knowledge base is attempting to infer.

Constraint declarations CRL constraints take the form:

```
constraint <id> ::
    impermissible if
    <antecedent>.
```

For example:

```
constraint 42 ::
    impermissible if
    professor hasValue yes and
    tenured hasValue no.
```

Constraint declarations specify invalid states of affairs; that is, combinations of data items that should not occur in the domain being modelled.

Askable declarations CRL askable declarations take the form:

```
askable <askable_item> /
    <type> /
    <possible values>.
```

For example:

```
askable tenured / category / [yes, no].
```

Askable declarations specify which data items can be used as inputs to the knowledge base and what values these data items can take.

MAPPING ONTOLOGIES TO KNOWLEDGE BASES

DISCOVER incorporates a syntactic translation program to convert ontologies represented in MOVES into CRL. Once this conversion has taken place, a set of mapping rules is required, to define how to map between the terminologies in the ontology and the knowledge base. The mappings involved can be simple, requiring a single rule to map an ontology term to a knowledge base term, or much more complicated requiring several rules to translate a single term. The translation rules formalise the extent to which the knowledge base *commits* to the ontology. The translation rules are a *statement of ontological commitment*. (It would be possible to check a knowledge base against an ontology that the knowledge base does not commit to, but merely shares a conceptualisation with. In these circumstances the mapping rules would translate between the two *different specifications* of the *same conceptualisation*.)

As an example, the University Ontology contains the following statement:

```
frame researchStudent of student.
```

The translation program converts this to the following CRL rule:

```
rule 130 ::
    uni__student(uni) if
    uni__researchStudent(uni).
```

To avoid clashes in name-space, the translator prepends an identifier for the University Ontology (`uni` — short for university) prepended to the names of the ontology terms. Also, the translator introduces the MOVES variable `uni` to make explicit the first-order definitions from MOVES.

To allow comparison of the translated ontology rules with the knowledge base, DISCOVER requires mapping rules. The mapping rules used to map the example rule are given below:

```
rule 100 ::
    researchStudent hasValue yes if
    uni__researchStudent(uni).

rule 101 ::
    uni__researchStudent(uni) if
    researchStudent hasValue yes.

rule 110 ::
    student hasValue yes if
    uni__student(uni).

rule 111 ::
    uni__student(uni) if
    student hasValue yes.
```

The constant `uni` is a term introduced to represent a person to whom all the knowledge base rules apply. These mapping rules provide us with a chain of inference from the knowledge base terms to the ontology terms using rules 101 and 111 and from the ontology terms to the knowledge base terms using rules 100 and 110.

DISCOVER VERIFICATION PROCEDURE

As stated earlier, the goal of the DISCOVER verification process is to identify any ways in which the concepts and relations in the knowledge base of the knowledge-based system conflict with the definitions in the ontology, taking into account the equivalences specified by the mapping rules. Given the framework described in the previous sections, the COVER verification mechanism can perform this task. As described fully in [15], COVER incorporates three subsystems: the integrity checker; the rule checker; and the environment checker; we briefly summarise their capabilities here, and how these are exploited by DISCOVER.

Integrity Checker

The COVER integrity checker checks the "connectivity" of the knowledge base; that is, it makes sure that data items in the antecedents of all rules are either askable, or are in the consequents of other rules; and that the data items in the consequents of all rules are either in the antecedents of other rules, or are goal items. It also detects circular dependencies among data items in rules. As part of DISCOVER, the integrity checker treats the mapping rules just like any other rule, with the exception that no anomaly can occur as a result of firing two mapping rules in a row that convert in different directions.

In terms of "connectivity" in verifying ontological commitment, there is actually no requirement to map completely between ontology and knowledge base terms. However, where a mapping *is* defined, COVER ensures the basic integrity of the mapping — that is, that the terms used in the mapping do in fact exist in the ontology/knowledge base.

The circularity issue is more important for verifying ontological commitment. Circularity occurs when the original premise of a rule can be inferred from the consequent of the same rule. The mapping rules typically contain circularity by their very nature (for example, rules 100 and 101 in the previous section, which are merely inverse mappings); therefore, DISCOVER does not detect rule cycles that involve only mapping rules, since these rule-cycles are not deemed to be anomalous. On the other hand, the following cycle *is* anomalous:

```
    rule 120 ::
        researchStudent hasValue yes if
        student hasValue yes.

    rule 104 ::
        student hasValue yes if
        uni__student(uni).

rule 1020 ::
        uni__student(uni) if
        uni__researchStudent(uni).

rule 101 ::
        uni__researchStudent(uni) if
        researchStudent hasValue yes.
```

The definition of `researchStudent` is actually circular here (chaining rules 120, 104, 1020, 101, 120), between the ontology and knowledge base. The error here is that rule 120 is too general: not every `student` is a `researchStudent`. Stated more generally, the knowledge base does not mean the same thing by the term `student` as the ontology means by the mapped-equivalent term `uni__student`. The problem can be fixed by modifying rule 120 to be more specific, for example:

```
    rule 120 ::
        researchStudent hasValue yes if
        student hasValue yes and
        takesCourses hasValue no.
```

Rule Checker

The COVER rule checker examines the knowledge base for anomalies which can be detected by comparing *pairs* of rules, looking for duplicate rules, subsumed rules, and conflicting rules. A rule is said to be subsumed if it is merely a specific case of another more general rule. Rules are in conflict with each other if they are able to imply different mutually exclusive inferences from any set of inputs.

Since the ontology and the knowledge base manipulate different terms, prior to invoking the COVER rule check, DISCOVER statically converts the ontology rules to the knowledge base terminology. The mapping rules are used for this conversion, of course. For example, rule 1020 above becomes:

```
rule 1020 ::
    student hasValue yes if
    researchStudent hasValue yes.
```

The three types of rule pair anomalies that DISCOVER can detect are explained below.

Conflict Conflict occurs when two rules are capable of deriving incompatible conclusions from a set of given premises. For example, in the University Knowledge Base the rule

```
rule 25 ::
    staff hasValue yes if
    student hasValue yes and
    tutor hasValue yes.
```

is in conflict with the ontology constraint

```
staff disjoint_with student
```

which is converted to the following, prior to the rule check:

```
rule 125 ::
    staff hasValue no if
    student hasValue yes.

rule 126 ::
    student hasValue no if
    staff hasValue yes.
```

COVER reports the conflict between rules 25 and 125. The error here is that the ontology and knowledge base conceptualise "staff" in different ways. The ontology insists that staff cannot also be student, while the knowledge base allows students performing tutoring work to be considered staff. This is the kind of conceptual mismatching that can cause serious problems in knowledge sharing unless it is detected and eliminated. In this case, there are several ways to eliminate the problem, such as simply not mapping the two different "staff" concepts, or modifying the ontology to allow tutoring staff.

Subsumption A subsumption anomaly is detected if one rule is a more general form of another rule. For example in the University Knowledge Base the rule:

```
rule 14 ::
    facultyMember hasValue yes if
    student hasValue yes and
    studiesHere hasValue yes.
```

is subsumed by the ontology rule

```
frame student of facultyMember.
```

which is converted to the following, prior to the rule check:

```
rule 114 ::
    facultyMember hasValue yes if
    student hasValue yes.
```

When a subsumption anomaly involves two knowledge base rules it is normally a sign of redundancy. When it occurs between an ontology rule and a knowledge base rule, it typically indicates a problem with ontological commitment, either due to faulty mapping rules or a pathological case where the knowledge base and the ontology are based on different incompatible conceptualisations. In the latter case, the correct course of action would be to lower the level of ontological commitment — that is, to map fewer terms — providing of course that greater commitment to the ontology is not required for knowledge sharing. In the example above it is clear that the mismatch between knowledge base and ontology is due to the ontology defining `facultyMember` as someone who is a member of a faculty at a university somewhere, and the knowledge base definition that requires the person to be a member of a particular university. The anomaly can be remedied by adding the line

```
uni ignores studiesHere.
```

to the mapping rules. This statement tells DISCOVER that `studiesHere` is meaningless to the University Ontology, and enables DISCOVER to discount the anomaly as simply the product of the differing granularity of knowledge base and ontology.

Environment Checker

The COVER environment checker performs the most computationally expensive checks on the knowledge base. Firstly, the environment checker generates all combinations of askable items and their values which will infer each goal. These sets of askable items are known as environments. The environment checker then checks each environment to see if it violates a constraint or if a state which would violate a constraint can be inferred from it.

The algorithm of the DISCOVER environment checker consists of the following three steps:

1. *Standard COVER environment check:* The DISCOVER environment checker first checks the knowledge-based system alone without using the mapping rules or the ontology.
2. *Generate selected ontology environments:* Generate environments for those ontology terms that map to KB terms.
3. *Check conflicts:* Check for ambivalence between the environments for ontology terms and their equivalent knowledge base terms.

If an environment can assign different values to a KB term and the equivalent ontology term, conflict is correctly deduced and reported as an anomaly. Suppose the ontology contains the term t_o that maps to the knowledge base term t_k; t_o can be inferred from the knowledge base term p; and t_k can be inferred from the knowledge base term $\neg p$. This state of affairs is anomalous since, t_o and t_k can be inferred from conflicting environments — that is, p and $\neg p$.

In the following example based on the university knowledge base and the university ontology `uni_teachingAssistant` is derivable only from `staff hasValue yes` and `student hasValue yes` which is an impermissible environment according to knowledge base constraint 2304; therefore an anomaly is reported.

```
rule 130 ::
    uni__teachingAssistant(uni) if
    uni__student(uni) and
    uni__staff(uni).

rule 1324 ::
    uni__student(uni)
```

```
    if student hasValue yes.

rule 1325 ::
    uni__staff(uni)
    if staff hasValue yes.

constraint 2304 ::
    impermissible if
    staff hasValue yes and
    student hasValue yes.
```

The error here is the reverse of the earlier example of rule-check conflict: here, the knowledge base (rather than the ontology) contains a constraint stating that `staff` and `student` cannot both have the value `yes` in a legal knowledge base state. This forbids the concept of `teachingAssistant` in the ontology. Now, this may not be a problem, because it indicates the ontology is actually more general than the knowledge base in this particular case. All it really means in practice is that this knowledge base doesn't allow the existence of "teaching assistants". A problem would only arise if the knowledge base tried to share the constraint 2304, which would then contradict the common ontology.

CONCLUSION

This paper has demonstrated how existing verification technology (the COVER tool) can be adapted to check the commitment of a knowledge base to an ontology. The DISCOVER tool can detect anomalies between the ontology and knowledge base, such as conflicting, subsumed and circular definitions. Based on the reports produced by DISCOVER, knowledge engineers can modify the knowledge base to ensure that it commits to the ontology in a satisfactory way. Ontological commitment is a prerequisite for knowledge sharing and reuse.

DISCOVER has extended the integrity and rule checks performed by the COVER tool. DISCOVER has been demonstrated on an example knowledge base and ontology in an academic domain. We intend in the future to apply it to check more complex and realistic examples. We also plan to use DISCOVER to verify the ontological commitment of a database conceptual schema.

This effort is part of a larger research undertaking to address a range of problems associated with knowledge sharing and knowledge verification and validation:

- The general problem of "fusing" knowledge from disparate sources; this is the central objective of the KRAFT project [19].
- The provision of an agent-oriented architecture to provide assistance to knowledge engineers in identifying appropriate tools and sources of meta-knowledge for validation of knowledge-based systems. This is a future goal of the DISCOVER project.
- The need for validation of distributed knowledge-based systems; this is the objective of the COVERAGE project [21].

References

[1] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senatir, and W.R. Swartout. Enabling Technology for Knowledge Sharing. *AI Magazine*, 12(3):36–56, Fall 1991.

[2] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand. Agent-based Business Process Management. *International Journal of Cooperative Information Systems*, 5(2):105–130, 1996.

[3] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, 1992.

[4] D. Cockburn and N.R. Jennings. ARCHON: A Distributed Artificial Intelligence System for Industrial Applications. In *Foundations of Distributed Artificial Intelligence*, New York, 1995. John Wiley & Sons.

[5] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proceedings of Third International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press, 1994.

[6] M.R. Genesereth and R.E. Fikes. Knowledge Interchange Format, Version 3.0, Reference Manual. Technical Report Report Logic-92-1, Logic Group, Computer Science Department, Stanford University, June 1992.

[7] D. B. Lenat and R. V. Guha. *Building Large Knowledge-based Systems.* Addison-Wesley, Reading MA, 1990.

[8] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), 1996.

[9] T.R. Gruber. Towards Principles for The Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, 1995.

[10] Antony Flew. *A Dictionary of Philosophy*, page 238. St Martin's Press, 175 Fifth Avenue, New York, New York 10010, first edition, 1979.

[11] W.V. Quine. *From a Logical Point of View: Nine Logico-Philosophical Essays.* Harvard University Press, Cambridge, MA, 1961.

[12] A. Church. Ontological commitment. *The Journal of Philosophy*, 55:1008–14, 1958.

[13] W.V. Quine. *Word and Object.* The Technology Press of The Massachusetts Institute of Technology and John Wiley & Sons, Inc, 1960.

[14] Nicola Guarino, Massimiliano Carrara, and Pierdaniele Giaretta. Formalizing ontological commitments. In *Proceedings of AAAI 94*, volume 1, pages 560–567, 1994.

[15] A. D. Preece, R. Shinghal, and A. Batarekh. Verifying Expert Systems: a Logical Framework and a Practical Tool. *Expert Systems with Applications*, 5:421–436, 1992.

[16] Alun D. Preece, Rajjan Shinghal, and Aïda Batarekh. Principles and practice in verifying rule-based systems. *Knowledge Engineering Review*, 7(2):115–141, 1992.

[17] Alun D. Preece and Rajjan Shingal. Foundation and application of knowledge base verification. *International Journal of Intelligent Systems*, 9:603–701, 1994.

[18] N. Zlatareva and A. D. Preece. An Effective Logical Framework for Knowledge-Based Systems Verification. *International Journal of Expert Systems: Research and Applications*, 7(3):239–260, 1994.

[19] P. Gray, A. Preece, N Fiddian, W.A. Gray, T. Bench-Capon, M. Shave, N. Azarmi, and M. Wiegand. KRAFT: Knowledge Fusion from Distributed Databases and Knowledge Bases. In *Eighth International Workshop on Database and Expert System Applications (DEXA-97)*, pages 682–691. IEEE Press, 1997.

[20] N. F. Roy and C. D. Hafner. State of the Art in Ontology Design. *AI Magazine*, 18(3):53–74, Fall 1997.

[21] N. Lamb and A. Preece. Verification of Multi-Agent Knowledge-Based Systems. In Marie-Christine Rousset, editor, *ECAI-96 Workshop on Validation of Knowledge Based Systems*, 1996.

APPENDIX: MOVES SYNTAX

```
<ontology>::==          <expressions>
<expressions>::==       <expression> | <expressions> <expression>
<expression>::==        <constraint> | <declaration> | <instantiation>
<constraint>::==        <short_constraint> | <long_constraint> | <disjoint>
<disjoint>::==          <frame> disjoint_with <frame>
<long_constraint>::==   constraint <frame_selector> <instance_selector>
                        tohave <condition>
<short_constraint>::==  constraint <frame_selector> tohave <condition>
<frame_selector>::==    <assignment> | <assignment> and <quantifier>
<assignment>::==        [not] <variable> isa <frame>
<filter>::==            <filter> <filter> | where <selector>
<selector>::==          <assignment> | <specialiser>
<instance_selector>::== slotValue(<variable>,<slot>,<comparison>)
<comparison>::==        <value> | <simple_expr> <value> |
                        <simple_expr> <variable>
<condition>::==         <disjunction>
<disjunction>::==       <conjunction> | <conjunction> or <disjunction>
<conjunction>::==       <slot_test>|<slot_test> and <slot_test>
<slot_test>::==         slotValue(<variable>,<slot>,<comp_comparison >)
<comp_comparison>::==   <value> | <simple_expr> <value> |
                        <comp_exp> <variable>
<comp_exp>::==          <simple_expr> | var
<declaration>::==       <frame_decl> | <slot_decl>
<frame_decl>::==        frame <frame> of <frame_list>
<frame_list>::==        <frame> | <frame> <frame_list>
<slot_decl>::==         <frame> hasSlot <slot> of <type>
<type>::==              integer | real | boolean | <frame>
<instantiation>::==     <frame_inst> | <slot_inst>
<frame_inst>::==        instance <variable> of <frame>
<slot_inst>::==         hasValue(<variable,<slot>,<value>)
<value>::==             <integer> | <real> | <boolean> | <string>| <frame>
<simple_expr>::==       not | lt | gt
<integer>::==           <atom>
<real>::==              <atom>
<boolean>::==           true | false
<string>::==            <atom>
<frame>::==             <atom>
<slot>::==              <atom>
<variable>::==          <atom>
```

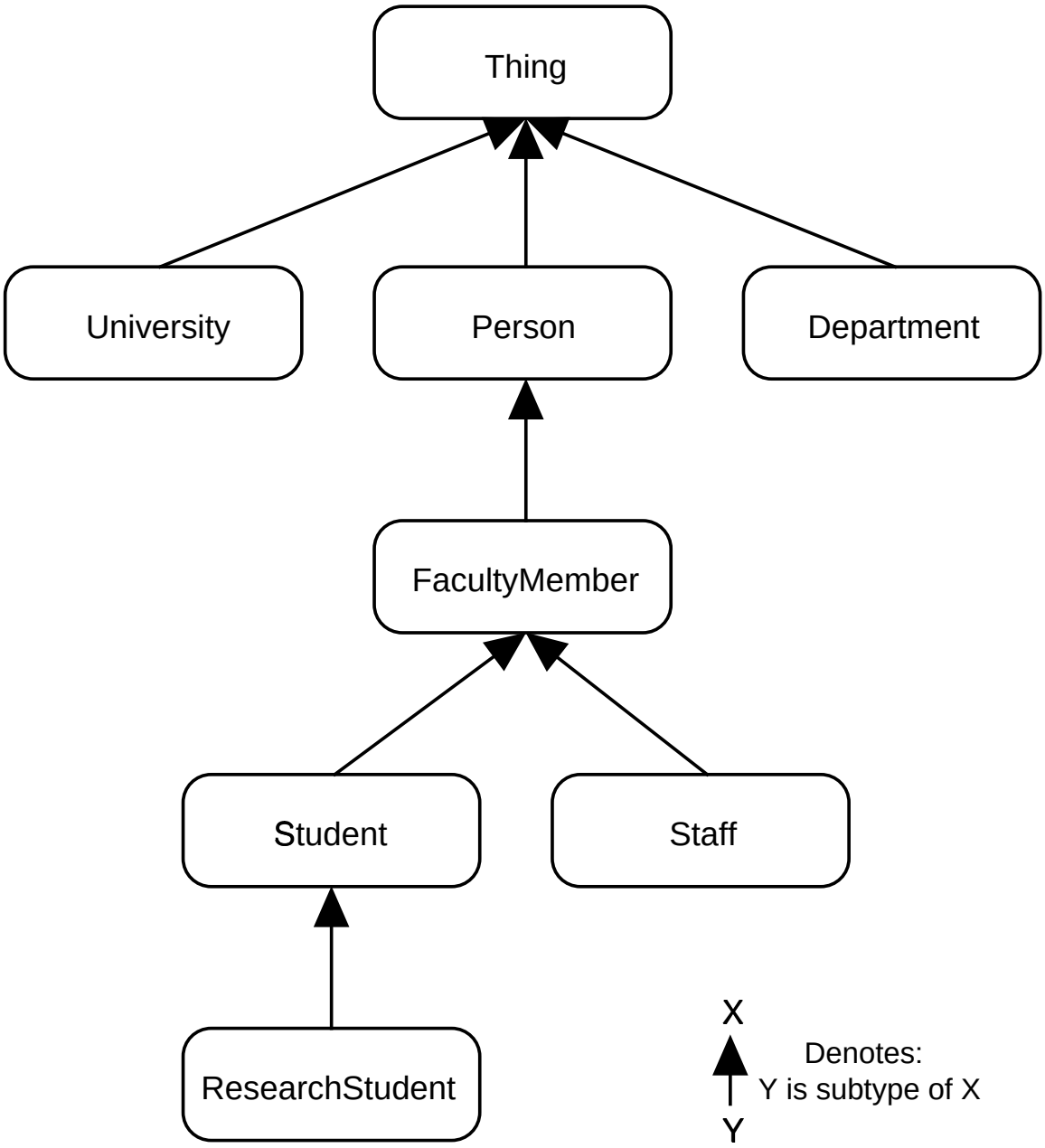

Figure 1: The University Ontology Frame Hierarchy.