Knowledge Reuse and Knowledge Validation

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

Automatic verification tools such as COVER have proven to be valuable aids in the validation process for knowledge-based systems (KBS). COVER checks KBS for logical anomalies. Background domain knowledge can allow COVER to detect errors in knowledge bases that would otherwise go undetected. Ontologies are a necessary component of knowledge sharing: two KBS cannot share knowledge unless they commit to a common ontology. Ontologies also provide a rich source of background domain knowledge for validation. This paper describes a tool, DISCOVER, which verifies KBS against ontologies. DISCOVER verifies heterogeneous sources of knowledge: KBS are represented in CRL (COVER rule language), and ontologies are represented in MOVES (Meta-ontology for the verification of expert systems). The paper describes MOVES and CRL, and discusses a number of anomalies arising between KBS and ontologies. It is shown that DISCOVER can be used to verify that a KBS commits to a given ontology, which is a prerequisite for sharing its knowledge.

Introduction and Motivation

The reuse and sharing of knowledge bases is a central theme of knowledge engineering in the 1990s (Neches *et al.* 1991). Whereas, in the 1980s, organisations focussed upon the construction of standalone 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 the evolution of organisations, relating to the ideas of enterprise modelling and business process reengineering (Jennings *et al.* 1996).

Knowledge Reuse 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 (Wiederhold 1992).
- *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 (Cockburn & Jennings 1995).

Enabling Technology: Ontologies Early work on enabling technology for knowledge sharing established that three components are needed to allow knowledge to be shared between two knowledge bases (Neches *et al.* 1991):

- a common protocol in which to communicate knowledge;
- a common language in which to express knowledge;
- 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 (Finin *et al.* 1994) and the KIF language (Genesereth & Fikes 1992) produced by the Knowledge Sharing Effort (KSE) project (Neches *et al.* 1991). 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 (Lenat & Guha 1990) to more modest efforts in defining terminology in a specific application area (Uschold & Gruninger 1996).

Although precise definitions of an ontology differ, the most widely held view is that an ontology an in explicit specification of a *conceptualisation*: "the objects, concepts and entities that are asumed to exist in some area of interest, and the relationships that hold among them" (Gruber 1995). 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: "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 is that of a knowledge base or database conceptual schema; any knowledge representation language or database schema definition language may be used to define 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 is accomplished using a set of *translation rules*; the second is 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 (Gruber 1995), 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 validation issue. It is worth noting that there is no completeness requirement on ontological committment: 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 unsharable statements. Similarly, there is no need to have an equivalent knowledge base term for every term in the ontology.

Verifying Ontological Commitment To support sharing and reuse of knowledge, we want to provide help in verifying the commitment of a knowledge base to an ontology. Furthermore, we want to reuse existing knowledge base verification tools for this task. This paper reports on an initial investigation of the use of the COVER knowledge base verification tool (Preece, Shinghal, & Batarekh 1992) for verifying ontological commitment. COVER is an *anomaly checker*: it analyses a knowledge base for undesirable properties including conflicting knowledge, redundant knowledge, and deficient knowledge. This 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 (Zlatareva & Preece 1994). The paper:

- introduces a simple ontology description language called MOVES (based on a subset of CycL (Lenat & Guha 1990), but with a Prolog flavour);
- shows how an ontological commitment is defined using mapping rules;
- shows how, with extensions, COVER can be used not only to check the ontological commitment, but also to do the terminological translations between ontology and knowledge base (and, hence, to verify the mapping rules also!);
- examines what anomalies like conflict, redundancy, and deficiency mean when they involve ontological commitments.

The extended version of COVER (which does not change any of COVER's original functionality) is called DISCOVER (COVER for DIStributed knowledge bases).

It is worth observing that DISCOVER can do more than verifying ontological commitment: it can use an ontology as a body of *background knowledge* against which the original knowledge base can be validated. It can also employ knowledge from other sources to validate the knowledge base; for example, it can use items in a database as test cases, provided that the database commits to a common ontology with the knowledge base.

Heterogeneity

Knowledge bases are developed in many different languages. For an automatic verification tool to be useful to a broad corpus of knowledge engineers it must be able to verify knowledge bases implemented in as many languages as possible. Ontologies are also implemented in a number of different languages, e.g. KIF (Genesereth & Fikes 1992), and CycL (Lenat & Guha 1990). DISCOVER verifies knowledge based systems expressed in CRL (COVER Rule Language) against Ontologies also expressed in CRL (Preece, Shinghal, & Batarekh 1992). This does not compromise the usefulness of DISCOVER; since CRL is based

on first order logic, it is possible to derive CRL representations of knowledge expressed in other languages, that are also based on first order logic. The *University Ontology* described in this paper is implemented in MOVES (Meta-Ontology for the Verification of Expert Systems) which is introduced below. The University knowledge base is expressed in CRL, described later in this section.

MOVES

MOVES uses a hierarchical frame based system with multiple inheritance for knowledge representation. 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.
```

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

```
person hasSlot overworked of boolean.
person hasSlot hasdegree of boolean.
person hasSlot enrolled of boolean.
person hasSlot age of integer.

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`, `hasdegree`, `enrolled` and `age` are assigned to the frame `person`, and are therefore inherited by the frames `facultyMember`, `staff`, `student` and `ResearchStudent`. `FacultyMember` has the slots `uni` and `dep` assigned to it; these slots are inherited by the frames `staff`, `student`, and `researchStudent`. `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 subframes.

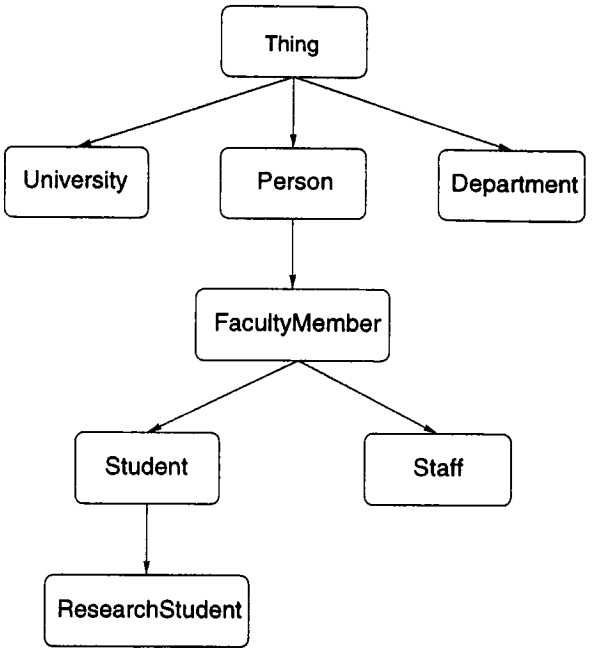

Figure 1: The University Ontology Frame Hierarchy

Constraints It is possible to add constraints to a MOVES ontology. MOVES constraints take the following form:

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

MOVES constraints have either two or three parts: a frame selector, an optional instance selector, and a 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.

```
constraint  x isa researchStudent and
            y isa staff
where
            slotvalue(x,supervior,y)
tohave
            slotvalue(y,uni,var university) and
            slotvalue(x,uni,university).
```

The above 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 `university`, instance `x` of `researchStudent` must also be a member of university `university`.

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

The `slotValue` command 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` command 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.

It is 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.

COVER Rule language

The COVER Rule language (CRL) is used for modelling knowledge bases. CRL was developed for use with the COVER verification tool, upon which DISCOVER is based. CRL statements take four forms.

- **Rules** take the form

```
rule <ruleid> ::
        <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 take a boolean value or single or multi-valued parameters.

- **Constraints** take the form

```
<ruleid> ::
        impermissible if
        <antecedent>.
```

For example:

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

Constraints specify invalid states of affairs i.e. combinations of data items that should not occur in the domain being modelled.

- **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.

- **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

DISCOVER uses a 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 translations 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. It could be said that the translation rules are a *statement of ontological commitment.*[1]

For example the University Ontology contains the following statement:

```
frame researchStudent of student.
```

The automatic translation program converts this to the following CRL rule:

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

Note that the data items have a unique identifier(`uni` — short for university) prepended to their names to avoid clashes in name-space. The original rule contained more information than it's CRL counterpart, i.e. the fact that `researchStudent` frames inherit all the slots that `student` frames contain. However, this information is meta-knowledge and, assuming that the ontology is defined in valid MOVES statements, the meta-knowledge is irrelevant to the verification process.

The translated rule is useless without mapping rules to allow comparison with the knowledge base knowledge. The mapping rules used to map this rule are given below:

[1] 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. We are exploring this issue further.

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

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

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

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

The data item `uni__instance` is a term introduced to represent the 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 termss using rules 101 and 111 and from the ontology terms to the knowledge base terms using rules 100 and 110.

Anomaly Checking

Logical anomalies in a knowledge base are often symptomatic of more serious problems (Preece, Shinghal, & Batarekh 1992). DISCOVER uses a slightly modified version of the COVER anomaly checker. The COVER anomaly checker contains three subsystems: the integrity checker, the rule checker, and the environment checker, we briefly summarise their capabilities here. Full descriptions appear in (Preece, Shinghal, & Batarekh 1992).

The COVER Integrity Checker The COVER integrity checker checks the "connectivity" of the knowledge base i.e. 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.

The COVER Rule Checker The COVER rule checker examines the knowledge base for anomalies which can be detected by comparing pairs of rules, looking for duplicated rules, subsumed rules, and conflict. 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 mutualy exclusive inferences from any set of inputs.

The COVER 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 constaint or if a state which would violate a constraint can be inferred from it.

The DISCOVER Integrity Checker

The DISCOVER 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.

Circularity Circularity occurs when the original premise of a rule can be inferred from the consequent of the same rule. The mapping rules often contain circular rules; therefore, DISCOVER does not detect rule cycles that involve only mapping rules, since these rule-cycles are not deemed to be anomalous. For example the cycle formed by rules 100 and 101 shown earlier is not anomalous since they are inverse mappings. On the other hand, the following cycle is anomalous:

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

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

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

rule 104 ::
        student hasValue yes if
        uni__student(uni__person).
```

The definition of `researchStudent` is circular here (chaining rules 120,104,1020,120).

The DISCOVER Rules Checker

The rule checker examines pairs of rules for anomalies. Since the ontology and the knowledge base manipulate different symbols the ontology must be statically converted to the knowledge base representation. For this check the mapping rules are used to fully convert the ontology rules to the knowledge base representation. For example, rule 1020 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
        demonstrates hasValue yes.
```

is in conflict with the ontology rule:

```
staff disjoint_with student
```

which translates to:

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

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

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 translates as

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

When this 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 probably 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, i.e. to map fewer terms — providing of course that greater commitment to the ontology is not required for knowledge sharing. In the case above it is clear that the mismatch between knowledge base and ontology is due to the ontology defining faculty member 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.

The DISCOVER environment checker

The DISCOVER environment checker is identical to the COVER environment checker, but because of the translation software it can make use of constraints defined in the ontology e.g. the University Ontology constraint:

```
constraint
        s isa staff and r isa researchStudent
where
        slotvalue(r,supervisor,s)
tohave
        slotvalue(r,department,var dept) and
        slotvalue(s,department,dept).
```

which is translated into the folloing CRL statement.

```
rule 100010 ::
    impermissible if
    uni__staff(Us) and
    uni__researchStudent(Ur) and
    uni__hasvalue(Ur,uni__supervisor,Us) and
    not (
        uni__hasvalue(Ur,uni__department,Udept) and
        uni__hasvalue(Ur,uni__department,Udept)
        ).
```

Conclusion

In this paper, we have shown 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.

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

- The general problem of "fusing" knowledge from disparate sources; this is the central objective of the KRAFT project (Home Page).
- The provision of an agent-oriented architecture to provide assistance to knowledge engineers in identifying appropriate tools and sources of metaknowledge 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 (Lamb & Preece 1996).

Acknowledgments

This work is supported by a grant from the University of Aberdeen Research Committee.

References

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

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

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

Gruber, T. 1995. Towards Principles for The Design of Ontologies Used for Knowledge Sharing. In Guarino, N., and Poli, R., eds., *Formal Ontology in Conceptual Analysis and Knowledge Representation.*

Home Page. KRAFT Project (Knowledge Reuse and Fusion/Transformation). URL `http://www.csd.abdn.ac.uk/research/kraft.html`.

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

Lamb, N., and Preece, A. 1996. Verification of Multi-Agent Knowledge-Based Systems. In Rousset, M.-C., ed., *ECAI-96 Workshop on Validation of Knowledge Based Systems.*

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

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

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

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

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

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