Ontology Translation by Ontology Merging and Automated Reasoning

Dejing Dou, Drew McDermott, and Peishen Qi

Yale University, Computer Science Department
New Haven, CT 06520
{dejing.dou, drew.mcdermott, peishen.qi}@yale.edu

Abstract. Ontology translation is one of the most difficult problems that web-based agents must cope with. An *ontology* is a formal specification of a vocabulary, including axioms relating the terms. Ontology translation is best thought of in terms of ontology merging. The merge of two related ontologies is obtained by taking the union of the terms and the axioms defining them. We add *bridging axioms* to one merge of two related ontologies not only as "bridges" between terms in two related ontologies but also make this merge as a new ontology for further merging with other ontologies. Automated reasoning by an inference engine (*OntoEngine*) can be conducted in the merged ontology in either a demand-driven (backward-chaining) or data-driven (forward chaining) way to implement translation. We illustrate our method by describing its application as an online ontology translation system, *OntoMerge*, which translate the *dataset* as a DAML file represented in the source ontology to the *dataset* represented in the target ontology also as a DAML file and it is stable to various datasets and ontologies. As uniform internal representation, *Web-PDDL*, which is developed as Web version of PDDL, is used for representing merged ontologies and datasets for automated reasoning.

1 Introduction

One of the most difficult problems that web-based agents must cope with is ontology translation, because web-based agents often use different ontologies to represent their own knowledge, even when some of them use the same web agent language such as WSDL [1] or DAML [2] to communicate with each other. We define *ontology* as a formal specification of a vocabulary, including axioms relating the terms. A *dataset* is defined as a set of facts expressed using a particular ontology [5]. The ontology translation problem is to translate a dataset represented in one (source) ontology to a dataset represented in another (target) ontology. The differences between two ontologies can include both syntactic and semantic differences, both of which we will have to deal with.

Previous work on ontology translation has made use of two strategies. One is to translate a dataset in any source ontology to the dataset in one big, centralized ontology that serves as an interlingua which can be translated into a dataset in any target ontology. Ontolingua [3] is a typical example for this strategy, but this strategy can't really work well unless a global ontology can cover all existed ontologies, and we can get agreement by all ontology experts to write translators between their own ontologies and this global ontology. Even if in principle such harmony can be attained, in practice keeping all ontologies — including the new ones that come along every day — consistent with the One True Theory is very difficult. If someone creates a simple, lightweight ontology for a particular domain, he may be interested in translating it to neighboring domains, but can't be bothered to think about how it fits into a grand unified theory of knowledge representation. The other strategy is to do ontology translation directly from a dataset in a (source) ontology to a dataset in another (target) ontology, on a dataset-by-dataset basis, without the use of any kind of interlingua. OntoMorph [4] is a typical example of this strategy. For practical purposes this sort of program can be very useful, but it tends to rely on special properties of the datasets to be translated, and doesn't address the question of producing a general-purpose translator that handles any source dataset.

Past work [3, 4] in the area for ontology translation has addressed both syntactic and semantic-issues, but tends to focus more on syntactic translation [4] because it is easier to automate. The "semantic" problem is to replace the vocabulary of the source ontology with the vocabulary of the target, which is much more difficult. The reason it is so difficult is that it often requires subtle judgment about the relationships between meanings of terms in one ontology and the meanings of terms in another ontology. Semantic translation requires ontology experts' intervention and maintenance [5] and can not be fully automated because only humans can understand the meanings of terms and the relationships between these terms.

Being confronted with ontology translation problem especially semantic translation as very difficult, we are developing our online ontology translation system, *OntoMerge*, basing on a new method as implementing ontology translation by ontology merging and automated reasoning. Our goal in this paper is to explain that ontology translation is best thought of in terms of ontology merging. The merge of two related ontologies is obtained by taking the union of the terms and the axioms defining them, using XML namespaces to avoid name clashes. We then add *bridging axioms* that relate the terms in one ontology to the terms in the other through the terms in the merge. We develop one merged ontology not only as a "bridge" between two related ontologies but also as a new ontology for further merging with other ontologies in the ontology community.

Although ontology merging requires ontology experts' intervention and maintenance, automated reasoning by an inference engine (*OntoEngine*) can be conducted in the merged ontology in either a demand-driven (backward-chaining) or data-driven (forward chaining) way to implement ontology translation automatically. We argue this method with its application, *OntoMerge*, which can do syntactic translation and has advantage at semantic translation, is a good idea to implement ontology translation and the system is stable to various datasets and ontologies.

2 Our Approach

2.1 Uniform Internal Representation

As we have pointed out, past work in the area for ontology translation usually mixed up syntactic translation and semantic translation. We think it is more enlightening to separate the two. If all ontologies and datasets can be expressed in terms of some uniform internal representation, we can focus on semantic operations involving this representation, and handle syntax by translating into and out of it. Although the users don't need to know the details of this internal uniform representation, getting them right is important to make our program work. Considering web-based agents, the internal uniform representation should contain the following elements:

1. A set of XML namespaces
2. A set of types related to namespaces.
3. A set of symbols, each with a namespace and a type.
4. A set of axioms involving the symbols.

Our language, called "*Web-PDDL*", is the AI plan-domain definition language PDDL augmented with XML namespaces and more flexible notations for axioms [6]. Like the original PDDL [7], Web-PDDL uses Lisp-like syntax and is a strongly typed first order logic language. Here is an example, part of a bibliography ontology [10] written in Web-PDDL. The DAML version of this ontology is at http://www.daml.org/ontologies/81.

```
(define (domain yale_bib-ont)
        (:extends (uri "http://www.w3.org/2000/01/rdf-schema#"
                   :prefix rdfs))
        (:requirement :existential-preconditions
                      :conditional-effects)
        (:types Publication - Obj
              Article Book Techreport Incollection
              Inproceedings - Publication
              Literal - @rdfs:Literal)
        (:predicates (author p - Publication a - Literal)
                .....
```

Assertions using this ontology are written in the usual Lisp style: (author pub20 "Tom Jefferson"), for instance. Quantifiers and other reserved words use a similar parenthesized syntax.

We write types starting with capital letters. A constant or variable is declared to be of a type T by writing "`x - T`". To make the namespace extensions work, we have broadened the syntax for expressing how one ontology ("domain") extends another. In the original PDDL, there was no specification of how domain names were associated with domain definitions. On the web, the natural way to make the association is to associate domains with URIs, so we replace simple domain names with `uri` expressions, which include a `:prefix` specification similar to XML namespace prefixes. With this new feature, we can add `@prefix:` to each term in Web-PDDL file to declare its namespace (ontology), such as `@rdfs:Literal` means a type from rdfs namespace (ontology). Symbols without a prefix come from the local namespace (ontology).

Types can be thought of as sets, but they are not first-class terms in the language. The sets denoted by two types must be disjoint unless one is a subset ("subtype") of the other. It is sometimes useful to state that an object `x` has type `T`, where `T` is a subtype of the type it was declared with. For this purpose you use the pseudo-predicate `is`, writing `(is T x)`.

If someone wants to use our system with an external representation other than Web-PDDL, there must exist a translator between the two. We have provided such a translator, which we call *PDDAML* [14], for translating between Web-PDDL and DAML. Writing translators for other XML-based web languages would not be difficult. In the following sections, we will use Web-PDDL to describe our work on ontology merging and automated reasoning, and ignore the external representation. For example, PDDAML can translate a dataset in DAML to the dataset in Web-PDDL or vice versa:

In DAML:

```
<rdf:RDF
   xmlns:rdf  = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:daml = "http://www.daml.org/2000/10/daml-ont#"
   xmlns      = "http://cs-www.cs.yale.edu/homes/dvm/daml/ontologies
                 /daml/yale_bib-ont.daml#"
>
<Book rdf:ID="book31">
   <author>Margaret Mitchell<author>
   <year>1996<year>
   <title>Gone With Wind</title>
</Article>
```

When PDDAML translates above dataset to the dataset in Web-PDDL, the class (such as Article) is translated to type and the property (such as author) is translated to predicate in Web-PDDL:

```
(define (dataset example01)
        (:domain (uri "http://cs-www.cs.yale.edu/homes/dvm//daml
                       /ontologies/daml/yale_bib-ont.daml"
                 :prefix yale_bib))
        (:objects
                 book31 - @yale_bib:Article)
        (:facts
                 (@yale_bib:author book31 "Margret Mitchell")
                 (@yale_bib:year book31 "1996")
                 (@yale_bib:title book31 "Gone With Wind")))
```

2.2 Ontology Merging and Bridging Axioms

Once we have cleared away the syntactic differences, the ontology translation problem is just semantic translation from the internal representations of a dataset in the source ontology to the internal representations of a dataset in the target ontology. Semantic translation requires ontology experts' intervention and maintenance because only humans can understand the relationships between meanings of terms in one ontology and terms in another ontology. For the rest of this paper, we will use two alternative bibliography ontologies as a running example. These were developed as contributions to the DAML Ontology Library, and while they are both obviously derived from Bibtex terminology, different design decisions were made by the people who derived them. One was developed at Yale, and we have given it the prefix `yale_bib` [10]; the other was developed at CMU, and it gets the prefix `cmu_bib` [11]. Although neither of them is a realistically complex ontology, the semantic differences that arise among corresponding types, predicates and axioms in these two ontologies serve as good illustrations of what happens with larger examples.

For example, both ontologies have a type called `Article`, but `@cmu_bib:Article` and `@yale_bib:Article` mean two different things. In the `yale_bib` ontology, `Article` is a type which is disjoint with other types such as `Inproceedings` and `Incollection`. Therefore, `@yale_bib:Article` only means those articles which were published in a journal, but `@cmu_bib:Article` includes all articles which were published in a journal, proceedings or collection.

Another example: both of these ontologies have a predicate called `booktitle`. In the `cmu_bib` ontology, `(booktitle ?b - Book ?bs - String)` means `?b` is a `Book` and it has title `?bs` as String. While in the `yale_bib` ontology, `(booktitle ?p - Publication ?pl - Literal)` means `?p` is an `Inproceedings` or an `Incollection`, and `?pl` is the title of the proceedings or collection which contains `?p`.

Here is how these distinctions would come up in the context of an ontology-translation problem. Suppose the source dataset uses the Yale ontology, and includes this fragment:

```
(:objects ... Jefferson76 - Inproceedings)
...
(:facts ... (booktitle Jefferson76 "Proc. Cont. Cong. 1") ...)
```

The translated dataset would then have to include this:

```
(:objects ... Jefferson76 - Article
              proc301 - Proceedings)
...
(facts ... (inProceedings Jefferson76 proc301)
           (booktitle proc301 "Proc. Cont. Cong. 1") ...)
```

Note that we have had to introduce a new constant, `proc301`, to designate the proceedings that `Jefferson76` appears in. Such *skolem terms* [18] are necessary whenever the translation requires talking about an object that can't be identified with any existing object.

It is tempting to think of the translation problem as a problem of rewriting formulas from one vocabulary to another. The simplest version of this approach is to use forward-chaining rewrite rules. A more sophisticated version is to use lifting axioms [17], an axiom of the form `(if p q)`, where `p` is expressed entirely in the source ontology and `q` entirely in the target ontology. Such a rule could be used in either a forward or backward (demand-driven) way, but the key improvement is that the axiom is a true statement, not an ad-hoc rule. This is essentially the idea we will use, except that we do away with any restrictions on the form or the vocabulary of the rules, and we adopt the term "bridging axiom" for them. A *bridging axiom* is then any axiom at all that relates the terms of two or more ontologies. By looking for such axioms, we separate the problem of relating two vocabularies from the problem of translating a particular dataset. This makes dataset translation a slightly more difficult problem, but the flexibility and robustness we gain are well worth it.

Bridging axioms use vocabulary items from both the source and target ontologies, and in fact may make use of new symbols. We have to put these symbols somewhere, so we introduce the concept of the *merged ontology* for a set of *component ontologies*, defined as a new ontology that contains all the symbols of the components, plus whatever new ones are needed. (Namespaces ensure that the symbols don't get mixed up.)

We can now think of dataset translation this way: Take the dataset and treat it as being in the merged ontology covering the source and target. Draw conclusions from it. The bridging axioms make it possible to draw conclusions from premises some of which come from the source and some from the target, or to draw target-vocabulary conclusions from source-language premises, or vice versa. When we get a pure conclusion in either vocabulary from merged-ontology premises we call it a projection into that vocabulary. The approach to translation that we focus on in this paper is to do forward inference from a source dataset in the merged vocabulary, retaining the projections as the result. In some cases, backward chaining would make more sense, as we discuss in the "Future Work" section below. In either case, the idea is to push inferences through the pattern source $\Leftrightarrow$ merge $\Leftrightarrow$ target.

Getting back to our example of merging the `yale_bib` and `cmu_bib` ontologies, we suppose we are using a merged ontology with prefix `cmu_yale_bib_merging`. When one term (type or predicate) in the `yale_bib` ontology has no semantic difference with another term in the `cmu_bib` ontology, we just use a new term with the same meaning as the old ones, then add bridging axioms to express that these three terms are the same. A third term is necessary because the merge is a new ontology that contains all the symbols of the component ontologies. It happens that (as far as we can tell), `@cmu_bib:Book` is the same type as `@yale_bib:Book`, so we can introduce a type Book in the `cmu_yale_bib_merging` ontology, and define it to mean the same as the other two. Because types are not objects, we cannot write an axiom such as `(= Book @cmu_bib:Book)`. (A term with no namespace prefix should be assumed to live in the merged ontology.) So we have to use a pseudo-predicate (or, perhaps, "meta-predicate") `T->` and write rules of the sort shown below. We call these *type-translation rules*:

```
(axioms:
         (T-> @cmu_bib:Book Book)
         (T-> @yale_bib:Book Book)
         ...
```

The general form of a type-translation rule is

```
(T-> type1 type2 [P])
```

which means "`type1` is equivalent to `type2`, except that objects of `type1` satisfy property `P`." We call `P` the distinguisher of `type1` and `type2`. If `P` is omitted, it defaults to true. For instance, suppose ontologies `ont-A` and `ont-B` both use the type `Animal`, but `ont-A` uses it to mean "terrestrial animal". Then the type-translation rule would be

```
(T-> @ont-A:Animal @ont-B:Animal (terrestrial x))
```

Here, `x` is a varible with type `@ont-B:Animal` and `(terrestrial x)` means `x` only represent some animal that lives on earth.

All other axioms can be stated in ordinary first-order logic. Consider the predicate title, declared as `(title ?p - Publication ?t - String)` in both the `yale_bib` and `cmu_bib` ontologies. We just reuse the same symbol in our `cmu_yale_bib_merging` ontology. The corresponding bridging axioms become:

```
(forall (?p - Publication ?ts - String)
        (iff (@cmu_bib:title ?p ?ts) (title ?p ?ts)))


(forall (?p - Publication ?ts - String)
        (iff (@yale\_bib:title ?p ?ts) (title ?p ?ts)))
```

When one term (type or predicate) in the `yale_bib` ontology has a semantic difference from the related term in the `cmu_bib` ontology, the bridging axioms become more substantial. Sometimes the axioms reflect our decision that one of the component ontologies gets a concept closer to "right" than the other. In our example, we might decide that `cmu_bib` has a better notion of Article (type). Then we add bridge axioms for `Article`, `@cmu_bib:Article`, and `@yale_bib:Article`:

```
(T-> @cmu_bib:Article Article)

(forall (?a - Article)
        (iff (exists (?j - Journal ?s - Issue)
             (and (issue ?s ?j)
  (contains ?s ?a)))
     (is @yale_bib:Article ?a)))
```

Above bridging axioms mean: in `cmu_yale_bib_merging` ontology, `Article` is same type as `@cmu_bib:Article`, which we prefer. But if and only if an *Article* is contained in some `Journal`, it is a `@yale_bib:Article`.

We also add bridge axioms for booktitle, for which we decide that `yale_bib`'s predicate makes more sense. The axioms relate `@cmu_bib:booktitle`, `@yale_bib:booktitle`, and our new predicate thus:

```
(forall (?a - Article ?tl - String)
        (iff (@yale_bib:booktitle ?a ?tl)
     (booktitle ?a ?tl)))

(forall (?a - Article ?tl - String)
        (iff (booktitle ?a ?tl)
     (exists (?b - Book)
             (and (contains ?b ?a)
          (@cmu_bib:booktitle ?b ?tl)))))
```

Above bridging axioms mean: we select booktitle in merged ontology as the same predicate as `@yale_bib:booktitle`, so that if and only if `?a` is an `Article` with `booktitle ?tl`, there exists some `Book ?b` such that `?b` contains `?a` and `?b` has `@cmu:booktitle ?tl`. The full version of the merge of the `cmu_bib` and `yale_bib` ontologies can be found at http://cs-www.cs.yale.edu/homes/dvm /daml/ontologies/pddl/cmu_yale_bib_merging.pddl. Given the merged ontology, any term in either component ontology can be mapped into a term or terms in the `cmu_yale_bib_merging ontology`, some terms in `cmu_yale_bib_merging` can be projected back into term(s) in `cmu_bib` and some into term(s) in `yale_bib`. When some datasets represented in the `yale_bib` ontology need to be translated into the datasets represented in the `cmu_bib` ontology, an automated reasoning system, such as a theorem prover, can do inference by using those bridging axioms in either a demand-driven (backward-chaining) or data-driven (forward chaining) way to implement translation, as we explained above. We will explore the forward-chaining option in depth in the next section.

We should mention one additional advantage of merged ontologies compared to direct translation rules [4, 17] between two related ontologies. The merged ontology serves not only as a "bridge" between the two given related ontologies, but also as a new ontology for further merging. If `foo1_bib`, `foo2_bib`, `foo3_bib` ... come out, we can use `cmu_yale_bib_merging` as a starting point for building a merged ontology that covers them all, or we may prefer a more incremental strategy where we merge `foo1_bib` and `foo2_bib`, creating, say "`foo_1_2_merging`", which would be used for translating between these two, then merge it with `cmu_yale_bib_merging` if and when a need arises for translations involving components of both merged ontologies. Exactly how many merged ontologies one needs, and how to select the right one given a new translation problem, are open research questions that we discuss in "Future Work" below. But the point to make here is that the number of merged ontologies one needs is unlikely to be as large as the number of rewrite-rule sets one would need under the direct-translation approach. If there are N ontologies that need to be translated into each other, the direct-translation approach requires $N(N-1)$ rule sets (assuming for simplicity that all pairwise combinations occur), which could be cut to $N(N-1)/2$ if the rules are bidirectional. Our approach requires on the order of N merged ontologies.

2.3 Automated Reasoning by Inference Engine (OntoEngine)

Theorem proving scares people. The desired to avoid it has led researchers to develop web languages, such as RDF and DAML, that look more like description logics [19] and not so much like predicate calculus, putting up with the awkwardness of description logics in hopes of reaping some benefit from their tractable computational properties. We reverse that decision by translating DAML back into predicate calculus and do old-fashioned theorem proving on the resulting internal representation. Our rationale is that we are inclined to think most of the theorem-proving problems that arise during ontology translation are just not that difficult [5]. Our inference engine, called *OntoEngine*, is a special-purpose theorem prover optimized for these sorts of problems. When some dataset in one or several source ontologies are input, OntoEngine can do inference in the merged ontology in either a demand-driven (backward-chaining) or data-driven (forward chaining) way, collecting the resulting projections into one or several target ontologies automatically.

Here is the *onto-structure*, which is defined as data structure of ontologies, in our OntoEngine if it has loaded some dataset and merged ontology in:

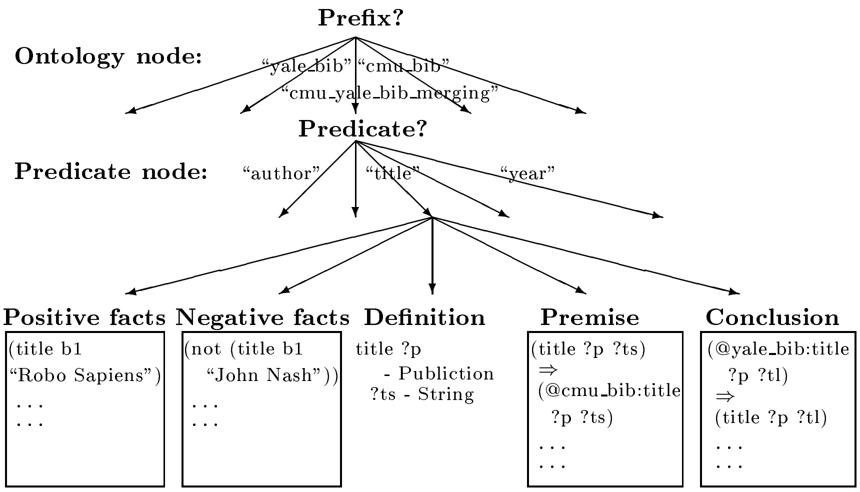

Fig. 1.

Onto-structure is a combination of the Tree-based indexing and the Table-based indexing. It can store all ontologies and dataset (facts) when OntoEngine loads them in and it also will store the new facts generated by inference. `Ontology nodes`s have been stored in a hash table with namespace prefix as key. In each of `Ontology nodes`, all `Predicate nodes` in this ontology have been stored in a hash table with predicate name as key. In each of Predicate nodes, the `Positive facts` and `Negative facts` are vectors of facts, such as `(title b1 "Robo Sapiens")` and `(not (title b1 "John Nash"))`. `Definition` for a predicate is like `(title ?p - Publication ?tl - String)`. When OntoEngine loads the ontology into onto-structure, except the axioms about type-translation rule, the other axioms should be able to be parsed into the INF (Implicative Normal Form) clauses, which is defined as: each clause is an implication with a conjunction of atomic formulas on the left (premise) and a conjunction of atomic formulas on the right (conclusion):

$$P_1 \wedge \cdots \wedge P_j \cdots \wedge P_n \Rightarrow Q_1 \wedge \cdots \wedge Q_k \wedge \cdots \wedge Q_m$$

So the `Premise` and the `Conclusion` are vectors of the INF clauses where atomic formula with this predicate name happens to be one of premises or one of conclusions.

So far, OntoEngine works fine in forward-chaining way to implement ontology translation. For example, if OntoEngine want to translate the following dataset

```
(booktitle McDermott80 - Inproceedings "Proc. AAAI 1" - Literal)
```

in `yale_bib` ontology to the corresponding dataset in `cmu_bib` ontology. This fact means: `McDermott80` is kind of article as `Inproceedings` and it has `booktitle ''Proc. AAAI 1"`, actually it is the title for that proceedings. To translate this fact, OntoEngine will call the forward-chaining reasoner with algorithm as:

In above algorithm, best clause means the clause with maximum weight after using some *bestfirst* method, which gives more weight to the clauses in the order as: least number of atomic formulas on the premise, more number of atomic formulas in the target ontology on the conclusion and more number of atomic formulas on the conclusion.

Getting back to our example for translating that fact. Suppose ontology experts have developed some merged ontologies and put them in their database or put them on online. First, OntoEngine try to find one merged ontology for the `yale_bib` and `cmu_bib` ontologies. If there is at least one ontology, which is supposed to be called `cmu_yale_bib_merging`, is the merge of these two component ontologies, OntoEngine loads it in. If not, OntoEngine has to report that it can't translate that dataset because it can't find appropriate merged ontology for inference. Second, OntoEngine loads that fact into `Positive facts` of the corresponding (booktitle) `Predicate node` in the `yale_bib` `Ontology node`. During forward-chaining reasoner processing that fact, the first (best) bridging axiom (INF clause) for Modus ponens will be:

```
(forall (?a - Article ?tl - String)
```

```
Procedure Process(facts)
    Input: dataset as facts
    Repeat
        oneFact = next fact in facts
        anynew = Forwardchaining(oneFact)
    Until last fact in facts
```

```
Function Forwardchaining(fact) return boolean
    Get best INF clause from corresponding Premises
    newFacts = Modus_Ponens(fact, clause)
    Repeat
        newFact = next fact in newFacts
        anyNew = Forwardchaining(newFact)
    Until last fact in newFacts
    Return anyNew
```

```
Function Modus_Ponens(fact, clause) return facts
    Repeat
        oneAtf = next AtomicFormula from leftside of clause
        substi = Unify(oneAtf, fact)
        Add substi into the whole substitutions and check its consistency
    Until get the consistent whole substitutions
    Get newFacts by substituting rightside of clause with substitutions
    Repeat
        newFact = next fact in newFacts
        If newFact belongs to target ontology Then store it
        Else add it into facts which will be returned for further inference
    Until last fact in newFacts
    Return facts
```

```
Function Unify(oneAtf, fact) return substitutions
    Repeat
        Get one variable from oneAtf and corresponding constant from fact
        Typecheck(variable, constant)
        If succeed Then make single substitution and add it to substitutions
    Until there is no more variable
    Return substitutions
```

```
Function Typecheck(variable, constant) return boolean
    If variable's type is same as constant's type Then match = true
    Else If variable's type is super type of constant's type Then match = true
    Else
        typeCons = type of constant
        fact = (is typeCons constant)
        match = Forwardchaining(fact)
    Return match
```

```
        (if (@yale_bib:booktitle ?a ?tl)
    (booktitle ?a ?tl)))
```

When OntoEngine is doing unification, the type checking system can avoid generating wrong substitutions because Web-PDDL is strongly typed, which means any symbol has a namepace and a type. This is another reason we think Web-PDDL is ideal internal representation for automated reasoning. In this example, the type checking system will find the `McDermott80`'s type is `@yale_bib:Inproceedings` which is not same as `@cmu_yale_bib_merging:Article`, nor it is the subtype of `@cmu_yale_bib_merging:Article`, but there is an bridging axiom about `@yale_bib:Inproceedings`:

```
(forall (?a - Article)
        (iff (exists (?p - Proceedings)(contains ?p ?a))
        (is @yale_bib:Inproceedings ?a)))
```

The type checking system will do recursively inference basing on this bridging axiom and generate skolem term for `?p`, which we suppose is `Proceedings01`, so the substitutions are {`?a/ McDermott80`} and {`?p/Proceeding01`}. One new fact is generated as `(contains Proceedings01 McDermott80)`, it will be stored in the merged ontology. Because it doesn't belong to the target ontology, this new fact will be used for further inference until one new generated fact belongs to the target ontology. Then doing forward chaining for another generated fact or original input fact. This control rule, which is described in *Modus-ponens* function of above algorithm, makes our OntoEngine never fall into infinite loop and it will return if no any new fact is generated. Therefore, we think OntoEngine is a special-purpose theorem prover, which is efficient for ontology translation by its type checking system and control rule.

After more inference are conducted in bridging axioms such as:

```
(forall (?a - Article ?p - Proceedings  ?t - String)
        (iff (and (contains ?p ?a) (title ?p ?t))
     (and (@cmu_bib:inProceedings ?a ?p)
                 (@yale_bib:booktitle ?a ?t))))

(forall (a - Article p - Proceedings)
        (iff (contains p a)
     (@cmu_bib:inProceedings a p)))
```

Finally, OntoEngine can generate new facts in the target(`cmu_bib`) ontology:

```
(title Proceedings01 "Proc. AAAI 1")
(inProceedings McDermott80 Proceedings01)
(booktitle Proceedings01 "Proc. AAAI 1")
```

This exactly is the dataset as the translated result in the `cmu_bib` ontology from that fact in the `yale_bib` ontology.

There are some other Java version theorem provers, such as KSL's JTP [12]. The difference is: JTP is a general-purpose theorem prover and can't be directly used for ontology translation. We finally decided to develop our own theorem prover, OntoEngine, which is not general-purpose but good at ontology translation.

3 Application: OntoMerge

Since our OntoEngine works well with some merged ontologies and datasets, we built an online ontology translation system which is called *OntoMerge* [13]. OntoMerge is developed on top of *PDDAML* [14] and *OntoEngine*. We assume that Web-based agent has specified the content of its inputs and outputs using some kind of Web language, such as DAML, so PDDAML is good at supporting OntoMerge to implement syntactic translation. To implement semantic translation, OntoMerge need support from OntoEngine and merged ontologies which are developed by ontology experts. We think that OntoMerge serves as a semi-automated nexus for agents and humans to find ways of coping with notational differences, both syntactic and semantic, between ontologies. It accepts a dataset as a DAML file in the source ontology, and will respond with the dataset represented in the target ontology also as a DAML file.

Here is an input dataset about two of Drew McDermott's publications as a DAML file in the `yale_bib` ontology:

```
<rdf:RDF
    xmlns:rdf  ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:daml = "http://www.daml.org/2000/10/daml-ont#"
    xmlns:yale_bib = "http://www.cs.yale.edu/homes/dvm/daml/ontologies
                      /daml/yale_bib.daml"
  >
  <rdf:Description rdf:about="">
    <daml:imports  rdf:resource="http://www.cs.yale.edu/homes/dvm/daml
                                 /ontologies/daml/yale_bib.daml"/>
  </rdf:Description>

  <yale_bib:Article rdf:ID="McDermott77">
  <yale_bib:author>Drew McDermott</yale_bib:author>
  <yale_bib:year>1977</yale_bib:year>
  <yale_bib:title>Flexibility and efficiency in a computer program
for designing circuits</yale_bib:title>
  <yale_bib:journal>MIT AI Laboratory Report 402</yale_bib:journal>
  </yale_bib:Article>

  <yale_bib:Inproceedings rdf:ID="McDermott80">
  <yale_bib:author>Drew McDermott</yale_bib:author>
  <yale_bib:year>1980</yale_bib:year>
```

```
  <yale_bib:title>A theory of metric spatial inference</yale_bib:title>
  <yale_bib:booktitle>Proc. AAAI 1</yale_bib:booktitle>
  </yale_bib:Inproceedings>
</rdf:RDF>
```

When receiving the above DAML file, OntoMerge calls PDDAML to translate it into a translation problem as a Web-PDDL file:

```
(define (translation-problem online-transprob)
     (:domain
       (uri "http://www.cs.yale.edu/domains/yale_bib-ont"
        :prefix yale_bib)
     )
     (:target-domain
       (uri "http://www.daml.ri.cmu.edu/domains/atlas-publications"
        :prefix cmu_bib)
     )
     (:objects
        McDermott80 - @yale_bib:Inproceedings
        McDermott77 - @yale_bib:Article
     )
     (:facts-to-translate
        (@yale_bib:author McDermott77 "Drew McDermott")
        (@yale_bib:year McDermott77 "1977")
        (@yale_bib:title McDermott77 "Flexibility and efficiency
in a computer program for designing circuits")
        (@yale_bib:journal McDermott77 "MIT AI Laboratory Report 402")
        (@yale_bib:author McDermott80 "Drew McDermott")
        (@yale_bib:year McDermott80 "1980")
        (@yale_bib:title McDermott80 "A theory of metric spatial
inference")
        (@yale_bib:booktitle McDermott80 "Proc. AAAI 1")
    )
)
```

Then OntoMerge inputs above translation problem to online version of OntoEngine, which is waiting for Web-PDDL file by listening sockets. When getting some translation problem which has specified the source ontology and the target ontology, OntoEngine contacts with some broker agent for finding at least one appropriate merged ontology. If broker agent can find one, it tells the uri to OntoEngine. If not, it reports sorry. After getting a uri, OntoEngine can get merged ontology online. If it is written in Web-PDDL, OntoEngine loads it in directly. If it is written in DAML or other Web language, OntoEngine first calls PDDAML to translate it to Web-PDDL file, then loads it in. Finally, OntoEngine loads the dataset(facts) which need to be translated in and processes those facts one by until no new fact is left nor generated.

The translated datasets is still represented in Web-PDDL as the following translation solution:

```
(define (translation-solution yale_bib_to_cmu_bib_so1)
   (:domain (uri "http://www.daml.ri.cmu.edu/domains/atlas-publications"
             :prefix cmu_bib))
   (:source-domain (uri "http://www.cs.yale.edu/domains/yale_bib-ont"
                    :prefix yale_bib))
   (:objects
       McDermott77 - @yale_bib:Article
       McDermott80 - @yale_bib:Inproceedings
       Proceedings7 - Proceedings
   )
   (:translated-facts
      (title McDermott77 "Flexibility and efficiency in a computer
program for designing circuits")
      (title McDermott80 "A theory of metric spatial inference")
      (title Proceedings7 "Proc. AAAI 1")
      (inProceedings McDermott80 Proceedings7)
      (booktitle Proceedings7 "Proc. AAAI 1")
      (year McDermott77 1977)
      (year McDermott80 1980)
      (author McDermott77 "Drew McDermott")
      (author McDermott80 "Drew McDermott")
      (journal McDermott77 "MIT AI Laboratory Report 402")
   )
)
```

Compare above solution with the original translation problem, it is easy to find that besides the translation due to the semantic difference between `yale_bib`'s `booktitle` and `cmu_bib`'s `booktitle`, OntoEngine translate the object of `yale_bib`'s `year` from string to number as the object of `cmu_bib`'s `year`. To get final output file in DAML, OntoMerge called PDDAML again to translate above dataset in Web-PDDL to the dataset in DAML:

```
<rdf:RDF
    xmlns:rdf  = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:daml = "http://http://www.daml.org/2000/10/daml-ont#"
    xmlns:yale_bib = "http://www.cs.yale.edu/domains/yale_bib-ont#"
    xmlns:cmu_bib = "http://www.daml.ri.cmu.edu/domains/atlas-publications#"
    xmlns = "http://www.daml.ri.cmu.edu/domains/atlas-publications"
>
  <rdf:Description about="">
    <daml:imports rdf:resource="http://www.daml.ri.cmu.edu/domains
                                /atlas-publications"/>
```

```
  </rdf:Description>

  <yale_bib:Article rdf:ID="McDermott77">
    <title>Flexibility and efficiency in a computer program for
designing circuits</title>
    <year>1977</year>
    <author>Drew McDermott</author>
    <journal>MIT AI Laboratory Report 402</journal>
  </yale_bib:Article>

  <yale_bib:Inproceedings rdf:ID="McDermott80">
    <title>A theory of metric spatial inference</title>
    <inProceedings rdf:resource="#Proceedings7"/>
    <year>1980</year>
    <author>Drew McDermott</author>
  </yale_bib:Inproceedings>

  <Proceedings rdf:ID="Proceedings7">
    <title>Proc. AAAI 1</title>
    <booktitle>Proc. AAAI 1</booktitle>
  </Proceedings>
</rdf:RDF>
```

Our OntoMerge works well online as DAML dataset in and DAML dataset out. Users can visit its website at http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html. Currently there are a few merged ontologies online, we set up OntoMerge as an ontology translation server to attract translation problems from other people and get their feed back. If OntoMerge can't handle some problems because no appropriate merged ontology, we will develop the merged ontology soon, but we will keep the OntoEngine as stable to any dataset.

4 Future Work

4.1 Backward Chaining

Although we inclined to think most of the theorem-proving problems that arise during ontology translation are not difficult and inferences can be conducted very well in the merged ontology in data-driven (forward chaining) way, the inference in demand-driven (backward-chaining) way is also necessary. For example, one agent has a query:

```
(and (father Fred ?x) (travel ?x ?y) (desti ?y "SF"))
```

which means "Did Fred's father travel to South Florida?", this query could be answered by another agent with its datasets in another ontology, which may has different meanings for `travel` or `desti`. In this case, ontology translation problem become to answer the query in the target ontology with the datasets in the source ontology. OntoEngine will have to work in the backward-chaining way.

On the other hand, backward-chaining may be necessary in the middle of a process which is working in the forward-chaining way in general. For example, when OntoEngine is unifying a fact as `(P c1)` with the `(P ?x)` in the axiom:

$$(\texttt{P ?x}) \wedge (\textit{member } ?\texttt{x} \; [\texttt{c1}, \texttt{c2}, \texttt{c3}]) \Rightarrow (\texttt{Q ?x})$$

Whether this unification can be successful is determined by whether the query as "Is `c1` a member of list as `[c1,c2,c3]`?" is true, and the only way to implement this unification is doing backward chaining.

After we add backward chaining in OntoEngine, we will keep OntoEngine as stable as possible to various datasets.

4.2 Methodology and Semi-automatic Tools for Ontology Merging

We use ontology merging as a tool to implement ontology translation and we think it is the most important tool for our ontology translation since we want to keep our OntoEngine stable and simple. Our further work for ontology merging will focus on getting better ontology merging by adding more bridging axioms in and developing semi-automatic tools for merging large ontologies.

We defined that the merge of two related ontologies is obtained by taking the union of the terms and the axioms defining them. We then add bridging axioms that relate the terms in one ontology to the terms in the other through the terms in the merge. It is some different with what some other people talk about ontology merging, such as PROMPT [15] and Chimaera [16]. These works would like to treat ontology merging as some kind of ontology-editing for two similar ontologies. They try to do ontology merging semi-automatically but they only provide user some suggestions for merging similar terms usually because the name similarity. The final decision for merging will still be made by user (human). Actually we agree that for now, that method is the only way to develop tools for semi-automatic ontology merging and it can be used to save ontology experts' time when they try to merge large ontologies. The reason is that the most ontologies for now contain more types(classes) and predicates than axioms which relate these types and predicates. Some time there even no any axiom in one ontology, maybe it is because ontology experts usually are lazy if they understand those relationships and don't want to tell others.

Our idea is trying to add more bridging axioms into merged ontologies to describe the relationships between terms of two or more related ontologies. The relationships are not only the relationships between terms in different ontologies but also the relationships between terms in this merged ontology itself. When more axioms are added in ontologies, it is possible to develop real semi-automatic tools to do ontology merging. Because only axioms can really help an ontology expert to understand ontologies developed by other experts, we can develop some semi-automatic tools corresponding to the axioms not only to the name similarity since it is often unreliable such as "cop" is similar as "cap" in the characters but much different in the meanings. We argue those semi-automatic tools based on the axioms are most possible to take a role at repeatedly work for merging large ontologies.

As the development of methodology for ontology merging, we can see a promising future for a new ontology community. Datasets in different ontologies can be translated to each other easily by an online ontology translation server such as OntoMerge and OntoMerge will take no effort to find or develop an appropriate merged ontology for automated reasoning.

4.3 "Ontology covering problem"

Another interesting issue in ontology merging is the "ontology covering problem". Given a set of source and target ontologies, ontology translation system, such as OntoMerge, looks for a merged ontology that was formed by merging exactly the set of ontologies it's now looking at. But in general it will not find such an ontology in its database. Instead it may find one that covers a subset of them, another that covers a superset, another that covers a set that intersects the set we're looking for. Here we introduce the term covers, defined thus: Ontology "covers" other ontologies $O_1, \ldots, O_k$ if it is the merger of a superset of $O_1, \ldots, O_k$. It covers $O_1, \ldots, O_k$ exactly if it is the merger of $O_1, \ldots, O_k$.

Here is an example. Suppose OntoMerge is given two source ontologies Onto_1 and Onto_2, and a target ontology Onto_3. It would prefer to do all reasoning in a merged ontology that covers these three exactly. If it has never seen any of these ontologies before, it must report failure, and ask the knowledge engineers to merge them. But as more and more ontologies are developed by the experts, it's possible that OntoMerge will have in its database Onto_1_3_5, the merger of Onto_1, Onto_3, and Onto_5, and Onto_2_4_5, the merger of Onto_2, Onto_4, and Onto_5. Now it has a choice of approaches:

- Run inferences in the combination of Onoto_1_3_5 and Onto_2.
- Run inferences in the combination of Onto_1_3_5 and Onto_2_4_5.
- Stop and ask the experts to merge Onto_1_3_5 and Onto_2_4_5.
- Stop and ask the experts to merge Onto_1, Onto_2, and Onto_3.

Here, combination means only take union of two ontologies as long as they don't contradict each other and no bridging axioms are added in. It is not obvious which is the correct choice. We would at least like it to be the case that the worst consequence of an incorrect choice is that some inferences may be missed, or that the deduction process runs with less than optimal efficiency. But we don't know how to guarantee these properties.

Or suppose the ontology DB contains one ontology covering ontologies 1, 2, 3, and 4, and another covering 1, 2, 3, and 5. On paper, either will work. But will we get the same inferences (projecting the vocabulary of Onto_4 and Onto_5 away)? Is one more efficient?

5 Conclusions

We have described a new approach to implement ontology translation by ontology merging and automated reasoning. Here are the main points we tried to make:

1. Ontology translation is best thought of in terms of ontology merging, the merge of two related ontologies is obtained by taking the union of the terms and the axioms defining them. We then add bridging axioms that relate the terms in one ontology to the terms in the other through the terms in the merge. Ontology merging should be more focus on adding axioms and make merged ontology as a new, better ontology for further merging, which will bring a new ontology community.
2. It is more enlightening to separate syntactic translation and semantic translation. If all ontologies and datasets can be expressed in terms of some uniform internal representation, semantic translation can be implemented by automatic reasoning. Web-PDDL is an ideal uniform internal representation and let some automatic translator (such as PDDAML) between Web-PDDL and other Web agent language (such as DAML) to do syntactic translation.
3. Automated reasoning can be conducted by our inference engine, OntoEngine, very well in the merged ontology in data-driven (forward-chaining) way for translating some dataset (facts) in a source ontology to the dataset in another target ontology. Demand-driven (backward-chaining) way will be added in soon since it is necessary. We think our OntoEngine as some kind of special-purpose theorem prover which will play a key role on web. OntoEngine has good control rule because it is specially used for ontology translation and it knows which is the target ontology. OntoEngine also has good type checking system for inference because Web-PDDL is a strongly typed first order logic language. We will keep our OntoEngine stable and simple to various datasets, and we will focus on improving methodology of ontology merging.
4. We set up an ontology translation server, OntoMerge, to describe our method. We hope OntoMerge can attract more ontology translation problem from other people and get their feed back, which will help our future work.

Acknowledgements: This work was supported by DARPA as DAML program.

References

1. http://www.w3c.org/TR/wsdl
2. http://www.daml.org/
3. T.R. Gruber. *A translation approach to portable ontology specifications.* Knowledge Acquisition, 5(2):199–220, 1993.
4. H. Chalupsky. *Ontomorph: A translation system for symbolic logic.* In Proc. Int'l. Con. On Principles of Knowledge Representation and Reasoning, pages 471–482, 2000. San Francisco: Morgan Kaufmann.
5. D. McDermott, M. Burstein, and D. Smith. *Overcoming Ontology Mismatches in Transactions with Self-Describing Agents.* In Proceedings of Semantic Web Working Symposium, pages 285–302, 2001.
6. Drew McDermott and Dejing Dou. *Representing Disjunction and Quantifiers in RDF.* To appear, Semantic Web Conference 2002.
7. Drew McDermott. *The Planning Domain Definition Language Manual.* Technical Report 1165, Yale Computer Science, 1998. (CVC Report 98-003)
8. http://www.cs.yale.edu/homes/dvm/daml/ontologies/daml/yale_bib.daml
9. http://www.daml.ri.cmu.edu/ont/homework/atlas-publications.daml
10. http://cs-www.cs.yale.edu/homes/ddj/ontologies/yale_bib_ont.pddl
11. http://cs-www.cs.yale.edu/homes/ddj/ontologies/cmu_atlas_publications.pddl
12. http://www.ksl.stanford.edu/software/JTP/
13. http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html
14. http://www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator.html
15. N. F. Noy and M. A. Musen. *PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment.* Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), Austin, TX.
16. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. textslAn Environment for Merging and Testing Large Ontologies. Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), Breckenridge, Colorado, April, 2000.
17. Sasa Buvac and Richard Fikes. *A declarative formalization of knowledge translation.* Proceedings of the ACM CIKM: The 4th International Conference on Information and Knowledge Management, 1995.
18. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approch.* Prentice-Hall, Inc. 1995.
19. F. Baader, D. McGuinness, D. Nardi, and P. P. Schneider 2002 *The Description Logic Handbook.* Cambridge University Press