A Formal Dialogue Model for Ontology Authoring

Richard Power [1]

Abstract. Several research teams have proposed controlled fragments of English suitable for building ontologies for the Semantic Web [9, 12, 7]. These Controlled Languages are typically employed in applications that allow users to compose texts through guided authoring; the sentences in the text are parsed and interpreted to obtain axioms in OWL. We investigate here a variant of this approach in which the user enters sentences one at a time through an instant messaging interface, and receives an immediate response in English from the authoring system before typing in the next sentence. Such an application can support a variety of dialogue moves by the user—not only assertions, but also retractions, questions, requests for explanation, and so forth—and for each, we need to define an appropriate set of responses for the authoring system. To illustrate the interest and complexity of this task, we develop a formal model for just one case: responses to an assertion.

1 INTRODUCTION

This paper proposes a formal model for responding to assertions in an ontology authoring dialogue. The context is an application in which an Ontology Management System (henceforth 'the program') supports an ontology developer (henceforth 'the user') in building an ontology, through a dialogue in a controlled fragment of English. The dialogue is conducted using an instant messaging interface, and is made up of two-utterance exchanges, each exchange comprising a sentence from the user followed by a response from the program. Such an application might support a variety of dialogue acts by the user, including questions, retractions, and requests for explanation, but we focus here only on exchanges where the user asserts an axiom; the point at issue is how the program ought to respond.[2]

Before describing the background in more detail, it is worth looking at a short example of such a dialogue. Let us suppose that the user is aiming to encode some facts from the 'People+Pets' domain proposed as an exercise by Horrocks [8], and that starting from an empty ontology, the dialogue opens with the following exchanges, with user assertions numbered for reference, and program responses shown in italics:

1. Rover is a dog.

Okay.

2. Every dog lives in a kennel.

Okay, so Rover lives in a kennel.

3. No dog is a cat.

Okay.

4. Butch is a dog that lives in a kennel.

Okay, but I can infer that Butch lives in a kennel.

5. Butch is a cat.

Inconsistent, since no dog is a cat.

6. Every kennel is a kennel that is located near a house.

Okay, but I have simplified as follows: Every kennel is located near a house.

7. Every dog that lives in a kennel is a pet.

Okay, but since every dog lives in a kennel, you could simplify as follows: Every dog is a pet.

The controlled language in this example is a variant of OWL Simplified English [11] (henceforth OSE). Each sentence can be interpreted deterministically as representing an expression in OWL [4]. As can be seen, the program does not merely accept any assertion that the user provides: sometimes an assertion is reformulated, or at least a possible reformulation is suggested; sometimes the assertion is rejected as contradictory. These are not responses to the English formulation of the axiom, but to the axiom itself; thus they would apply equally to any OWL encoding. This is shown by the following list, which gives interpretations of each assertion in description logic notation, along with comments on the program's response.

(1) $Rover \in dog$
No problem with the assertion, and nothing to report.

(2) $dog \sqsubseteq \exists livesIn.kennel$
The program can give useful feedback by reporting new entailments.

(3) $dog \sqcap cat \sqsubseteq \bot$
Here the program does not report the entailment that Rover is a non-cat. In principle an unlimited number of entailments can be drawn, so a policy is needed over which ones to report.

(4) $Butch \in dog \sqcap \exists livesIn.kennel$
Can be refactored to two simpler statements, $Butch \in dog$ and $Butch \in \exists livesIn.kennel$, of which the first is informative, but the second can be inferred using axiom 2.

(5) $Butch \in cat$
Because of axioms 3 and 4, this entails that Butch belongs to the unsatisfiable class $dog \sqcap cat$, and would therefore make the ontology inconsistent if accepted. Accordingly the program refuses to add it to the ontology.

(6) $kennel \sqsubseteq kennel \sqcap \exists locatedNear.house$
This axiom would be partly redundant in any context, since it can be refactored to two statements of which the first is the tautology $kennel \sqsubseteq kennel$.

(7) $dog \sqcap \exists livesIn.kennel \sqsubseteq pet$
In context this could be simplified to $dog \sqsubseteq pet$ while leaving the entailments of the ontology unchanged. However, there is an argument for allowing this redundancy in case the user later decides to retract axiom 2.

As well as introducing the application, the example shows that even starting with no domain knowledge at all, the program can exploit its reasoning abilities in order to detect redundancies or contradictions in the information provided, and also to report implications. Where flaws are detected, there is a choice between allowing them to remain, at least temporarily, or rejecting them (i.e., refusing to add the axiom

[1] Open University, UK, email: FirstName.SecondName@open.ac.uk

[2] An analysis of common moves in ontology authoring dialogues has been performed in the WhatIf project [10].

to the developing ontology). Our aim is to lay out these response options systematically, and suggest principles for determining which response the program should choose in any given case.

2 LOGICAL PRELIMINARIES

We have already introduced some notation from description logic, and before proceeding it will be convenient to outline the relevant concepts systematically (readers familiar with OWL may therefore skip the section). Description logic is not a single formal language but a set of resources; by selecting from these resources one can construct *fragments* with the desired trade-off between expressiveness and tractability. We opt here for a fragment known as $\mathcal{EL}$++, as defined for example by Baader [2], which achieves the virtues of simplicity and tractability (entailments computed in polynomial time), while retaining enough expressivity to encode some important ontologies such as SNOMED [3] and the Gene Ontology [1].

[h]

Notation	Logical type	English example
C	Named Class	dog
$\{I\}$	Named Individual	Rover
$C_1 \sqcap C_2$	Intersection	dog that is a pet
$\exists P.C$	Existential Restriction	lives in a kennel
$\perp$	Empty Class	non-existent entity

Table 1. Class patterns in $\mathcal{EL}$++

Domains are described in $\mathcal{EL}$++ using three kinds of atomic term: individuals, classes, and properties.[3] Individuals are entities such as *Rover* or *London*. Classes are sets of entities such as *dogs* or *cities*. Properties are relations between entities, such as a person or animal *living in* a place. By combining these terms with logical symbols, we can construct class expressions of various kinds, illustrated in table 1. They include *enumerated sets* containing one entity (allowing us to reformulate *Rover* as a class), intersections (or more informally, conjunctions), and existential restrictions, which denote entities having a specified relation to at least one entity belonging to a specified class.

Notation	English example
$\{I\} \sqsubseteq C$	Rover is a dog
$\{I\} \sqsubseteq \exists P.C$	Rover lives in a kennel
$C_1 \sqsubseteq C_2$	Every dog is an animal
$C_1 \sqsubseteq \exists P.C_2$	Every dog lives in a kennel
$C_1 \sqcap C_2 \sqsubseteq \perp$	No dog is a cat

Table 2. Statement patterns in $\mathcal{EL}$++

The patterns in table 1 allow us to express all statements as subclass relationships between two classes, which for convenience we will call the 'subject' and 'predicate' classes.[4] The meaning of such statements is always that the subject is a subclass of the predicate, where 'subclass' is defined so that every class is also a subclass of itself. Table 2 illustrates some common statement patterns with their OSE verbalisations. The last illustration shows that a disjointness relationship can be reformulated as a subclass relationship, which would be verbalised more literally by the sentence 'Every dog that is a cat is a non-existent entity'.

[3] We simplify here by considering only object properties, not data properties, for which we would also need two other kinds of term, datatypes and literals.

[4] This is not a standard terminology, but it is intuitive in applications using controlled natural languages, in which a statement $C_S \sqsubseteq C_P$ is usually verbalised with C_S as subject and C_P as predicate.

3 ASSUMPTIONS

An advantage in investigating ontology authoring dialogues is that we can define precisely the purpose of the interaction between user and program, and the range of permissible moves by both parties. We know that the purpose is to create an efficient and consistent description of a domain, encoded in OWL or some well-defined fragment thereof. We know that assertions by the user should conform to the grammar of a controlled natural language, with interpretations in the prescribed fragment of OWL. From the semantics of OWL we can define possible problems such as redundancy or inconsistency. The task therefore lends itself to formal modelling, as pioneered especially by Hamblin [5, 6].

In Hamblin's mathematical models, a dialogue is defined as an ordered set of *locutions*, each produced by a *participant*. Rules are laid down for the syntax and semantics of locutions, and also their pragmatic effects: for instance, the assertion of proposition P by a participant, if unopposed, commits *all* participants to the assumption that P is true.[5] Having thus modelled the effects of a locution, criteria can be stated for *evaluating* locutions as either legitimate or anomalous. For instance, once a participant has asserted P, it would be inappropriate for any participant to assert P again, or to assert a proposition that blatantly contradicts P. In this way, Hamblin defines a subset of dialogues that are *legal*—i.e., fully conformant to these criteria. Of course the criteria of legality are not arbitrary, but based on a over-riding assumption about the purpose of the dialogue: that of promoting the 'efficient exchange of information' [6].

As will be obvious, many of these ideas apply directly to the dialogues that concern us here. Like Hamblin, we have formal rules of syntax and semantics; we also have a set of propositions that have been asserted and accepted—namely, the ontology under development.[6] We have noted above examples of assertions by the user that are flawed, in ways that correspond to criteria like redundancy and consistency in Hamblin's rules for legality. However, there are differences too. In Hamblin's models, the participants are on the same footing; in our case, user and program have different roles associated with different constraints. To be useful, our model cannot merely reject some locutions by the user as 'illegal', so invalidating the entire dialogue; instead, we have to accept that some contributions will be flawed, and provide rules for generating helpful responses.

3.1 General principles

Following Hamblin's approach, we will develop rules for a *series* of dialogue models. In the first model, the subject of this paper, the user is permitted only to assert potential axioms; elsewhere we will propose further models in which the user may also retract assertions, ask questions, and so forth. This approach is merely a convenience: obviously there would be little practical value in a dialogue that allowed assertion but not retraction. In considering how the program should respond to an assertion, we therefore keep in mind that the model will later be expanded to allow other dialogue moves as well.

The model developed here resembles in its scope Hamblin's Systems 1 and 2 [6], which also allow assertions but not retractions.[7] We

[5] This does not mean that participants are required to believe all they are told, only that they either contest the proposition, or assume it for purposes of the dialogue.

[6] Hamblin calls this set of propositions a 'commitment slate' or a 'commitment store'.

[7] In System 1 a locution is illegal if it *repeats* an earlier proposition; in System 2 it is also illegal if it is *entailed by* earlier propositions.

1. The participants are a human user and a computer program.
2. The purpose of the dialogue is to build an ontology that encodes information about a domain.
3. At the start of the dialogue, the user has some knowledge of the domain, while the program has none. The program has no other source of domain knowledge except the user.
4. The program can remember exactly what it has been told, and reason with its knowledge reliably. The user is fallible both in memory and reasoning.
5. The *product* of the dialogue (the ontology) should be judged by criteria of *quality* including accuracy, consistency, completeness, and compactness.
6. The *process* of authoring the ontology should be judged by criteria of *efficiency* including the time and effort demanded of the user.

Figure 1. General principles for Ontology Authoring Dialogues

will label this model *OAD-1*, with the evident intention of developing successors labelled *OAD-2*, *OAD-3*, etc. in which the user may also peform retractions, questions, and so forth. The acronymn *OAD* stands for Ontology Authoring Dialogue, and reminds us that all these envisaged models have something in common. Leaving aside details of which moves are allowed and which are not, they all address a context in which a human user collaborates with a computer program in order to encode domain knowledge in an ontology. We should therefore begin with a clear statement of principles common to *all* these models, and this we provide in figure 1.

Note that these principles are at least partly informal: for instance, we have given criteria of quality and efficiency without saying exactly how they might be measured. Their value lies rather in the guidance they provide when constructing detailed models. To give just one example, consider the issue (raised above) of how the program should respond when the user asserts an axiom that is already entailed. On grounds of *quality*, the assertion should probably be rejected, leaving the ontology equally informative while more compact. However, as the principles in figure 1 point out, we have to take account also of the *efficiency* of the authoring process, and the *fallibility* of the user: perhaps it is better to retain a redundant axiom in case the axioms that entail it are later retracted (see assertion 4 in the introduction for a case in point).

3.2 Outline model

Figure 2 gives rules for *OAD-1*, a simple dialogue in which the only move available to the user is to assert axioms from a restricted subset of OWL, and the only move by the program is to respond to the user's last assertion. In responding, the program has three options: either accept the statement as it is, or accept it in some modified form, or reject it. If an axiom can be refactored into multiple statements, they may receive different responses. For instance, a sentence like 'Every kennel is a kennel that is located near a house' (assertion 6 in the introductory example) can be refactored into 'Every kennel is a kennel' and 'Every kennel is located near a house'; the program should reject the former as a tautology, but might accept the latter.

To state more precisely the grounds for simplifying or rejecting an assertion, we need to classify systematically the various kinds of redundancy and contradiction that can be found in our logical fragment (roughly $\mathcal{EL}$++), and it is to this we now turn.

4 REDUNDANCY

We have mentioned that both classes and statements can be redundant, and moreover, that they can be redundant in two ways, which we will call *inherent* redundancy and *contextual* redundancy. Crossing these distinctions (class vs statement, inherent vs contextual) we obtain a fourfold classification, which we can illustrate by a variant of the sample dialogue in the introduction. Suppose axioms 1-2 below have already been asserted, and consider options 3a-3d for axiom 3:

1	Every dog lives in a kennel.
2	Rover is a dog.
3a	Every dog that is a dog is a pet.
3b	Every dog that is a pet is a pet.
3c	Every dog that lives in a kennel is a pet.
3d	Rover lives in a kennel.

All of 3a-3d are redundant, but in different ways:

- 3a has an *inherently redundant class*, 'dog that is a dog'. This class is redundant because it can be simplified to 'dog'; moreover, it would be redundant in any context, not just the context provided by axioms 1-2.
- 3b is an *inherently redundant statement* because it would be true in any context (the predicate is contained in the subject).
- 3c has a *contextually redundant class* because once we have asserted (axiom 1) that every dog lives in a kennel, 'dog that lives in a kennel' can be simplified to 'dog'. The statement then becomes 'Every dog is a pet'.
- 3d is a *contextually redundant statement* because although it is stated in its simplest form, it already follows from axioms 1-2. There is therefore no need to assert it at all.

Formal definitions of these redundancy categories can be given as follows, using the usual extensional semantics for description logic.

1. A constructed class C_R is *inherently redundant* if it contains a constituent class C that has the same extension as C_R under all interpretations.
2. A constructed class C_R is *contextually redundant* if it contains a constituent class C that has the same extension as C_R for any interpretation satisfying all other axioms in the ontology. (This is the same as saying that the other axioms entail $C \equiv C_R$.)
3. A subsumption statement of the form $C \sqsubseteq D$ is *inherently redundant* if the extension of C is a subset of the extension of D under all interpretations.
4. A subsumption statement of the form $C \sqsubseteq D$ is *contextually redundant* if the extension of C is a subset of the extension of D for any interpretation satisfying all other axioms in the ontology. (This is the same as saying that the statement is entailed by the other axioms.)

To be fully precise, the definition of a contextually redundant class or statement should also stipulate that the class/statement is not *inherently* redundant (otherwise, any inherently redundant expression will also be contextually redundant).

1. The dialogue comprises a series of two-utterance exchanges, the first in each pair by the user, the second by the program.
2. Within each exchange, the first locution (by the user) asserts an axiom in the controlled language (OSE); the second locution (by the program) responds to this assertion, and may include English sentences outside OSE.
3. The axioms asserted by the user should conform to the description logic fragment known as $\mathcal{EL}$++ but with the further restriction that literals, datatypes and data properties are disallowed.
4. If the axiom asserted by the user is a tautology (inferable in any context), the program should point this out and reject it.
5. If the axiom has an equivalent simpler form in any context, the program should point this out and accept it only in this simpler form.
6. If the axiom has been asserted before, the program should point this out and refuse to add it again.
7. If the axiom would make the ontology inconsistent or incoherent, when added to those already present, the program should point this out and reject it.
8. If the axiom, although not asserted already, is entailed by the axioms already present, the program should accept it but with a warning pointing this out.
9. If the axiom can be stated more simply by taking account of axioms already present, the program should accept it in full but with a warning pointing out the simpler form.
10. If the axiom can be refactored into two simpler statements, the program should repond to these statements separately applying the rules given above.
11. If the axiom is accepted (possibly in part, possibly simplified), the program should give feedback on entailments (if any) that result from adding it to the ontology.

Figure 2. Outline rules for dialogue model *OAD-1*

5 CONTRADICTION

In description logic it is customary to distinguish two kinds of contradiction: inconsistency, and incoherence. An ontology is *inconsistent* if it has no interpretation. It is *incoherent* if at least one named class is unsatisfiable (i.e., can have no members without introducing inconsistency).

For contradiction as well as redundancy we can distinguish statements that are *inherently contradictory* from statements that are *contextually contradictory*. We thus obtain another fourfold classification:

1. A statement is *inherently inconsistent* if it has no interpretation (e.g., $Rover \in \bot$).
2. A statement is *contextually inconsistent* if it has no interpretation that also satisfies the other axioms in the ontology (e.g., $Rover \in cat \sqcap dog$ in an ontology that entails that cats and dogs are disjoint).
3. A statement is *inherently incoherent* if there is no interpretation in which all its named classes are satisfiable (e.g., $dog \sqsubseteq \bot$).
4. A statement is *contextually incoherent* if there is no interpretation satisfying the other axioms in the ontology in which all its named classes are satisfiable (e.g., $pekinese \sqsubseteq cat \sqcap dog$ in an ontology that entails that cats and dogs are disjoint).

As before, to be pedantically precise we should include in the definition of a contextual contradiction that it is not inherently contradictory.

6 REFORMULATION

The rules proposed for *OAD-1* (figure 2) require the program to *reformulate* assertions by the user when they contain redundant classes, or when they can be divided into multiple assertions which can be evaluated separately. These reformulations are based on two assumptions concerning the optimal encoding of an ontology:

1. Minimise the number of constructed classes.
2. Minimise the average complexity of axioms.

These objectives often work together, since by splitting up a complex axiom like $C \sqsubseteq D \sqcap E$ into two simpler ones $C \sqsubseteq D$ and $C \sqsubseteq E$, we may also obviate the need for the constructed class $D \sqcap E$, so favouring the first objective as well as the second. In proposing them, we are refining point 5 in our statement of general principles (figure 1), relating to the *quality* of the ontology, and specifically its compactness. We are saying, in effect, keep your classes and your axioms as simple as possible, even if this means that you need more axioms. Reducing the number of axioms might be desirable in itself, but in case of conflict, give priority to reducing axiom complexity.

This policy probably has intuitive appeal: it reminds us of familiar precepts from books on literary style, such as Strunk's maxim 'Omit needless words!' [13]. Requirements for an ontology are not necessarily the same, but I would suggest the following as supportive arguments:

- In computing the entailments of an ontology, it is convenient to restrict their number by considering only subsumption relationships between classes that occur in the axioms. This task is simplified if we keep the number of such classes to a minimum.
- Explanations of entailments typically show how they are derived from axioms. If we allow axioms that aggregate a number of statements, we may find that some of these are relevant to an explanation and some are not; this would mean that we have to included an extra step in the explanation, pointing out which part of the axiom is relevant.
- If the user asserts an aggregated axiom such as 'Rover is a dog that lives in a kennel', he/she might later wish to retract *part* of this assertion (e.g., 'Rover lives in a kennel'), leaving the rest intact. This can be implemented more efficiently if we divide the assertion into its parts, so that retraction consists in removing a simple axiom rather than simplifying a complex one.

- Similarly, if the user makes an assertion that partially duplicates an assertion made earlier (e.g., 'Rover is a dog that lives in a kennel' following 'Rover is a dog that is owned by a farmer'), the duplication can be checked more easily if the axioms have already been disaggregated, allowing the program to add 'Rover lives in a kennel' while ignoring the repetition of 'Rover is a dog'.

Leaving aside the trade-off between axiom number and axiom complexity, note that the principles stated above also oppose redundancy, which increases both the number of constructed classes and the average complexity of axioms, without yielding any compensating advantage at all.

6.1 Removing aggregation and inherent redundancy

Having defended these general principles of reformulation, let us consider how the program should proceed when responding to a user assertion that may flout them. An outline procedure—not the only possible one—is shown in figure 3. Note that the purpose of this procedure is not to decide which statements should be added to the ontology, but rather to draw up a list of candidates that meet our standards on formulation. Whether these candidates are actually added will depend on the context of the axioms already asserted, and in particular on criteria of duplication and contradiction that will be discussed in a later section.

So far as I can see, the only alternative to the procedure in figure 3 is to reverse the order of the first two steps. Suppose for instance that the user asserts 'Rover is a dog that is a thing'. Following figure 3, the first step is to simplify the inherently redundant class 'dog that is a thing' to 'dog'; starting instead with disaggregation, the first step would be to divide the assertion into two parts, 'Rover is a dog' and 'Rover is a thing'. The final result will be the same, since the inherently redundant statement 'Rover is a thing' will be removed in step 3. However, if we disaggregate first, we need to add a further step checking that the set of disaggregated statements has no duplicates: this would result for example from 'Rover is a dog that is a dog'. Following the order in figure 3, the class 'dog that is a dog' is already simplified to 'dog' in step 1, so this extra check is unnecessary.

Our current implementation of step 1 employs a set of rules for simplifying class expressions in the relevant fragment of description logic, along with a strategy for applying them. The task can be compared with the simplification of an arithmetical expression such as $((2 \times 3) + 4) + (5 \times 6)$; as well as knowing the rules for adding and multiplying, we need a strategy for which constituent to simplify first. For arithmetic many people would adopt a left-to-right depth-first strategy, and thus begin by evaluating 2×3, then $6 + 4$, then 5×6, then finally $10 + 30$. Our task is a little different since not all expressions can be simplified, but given a complex class like $\exists P.C \sqcap \exists P.(\top \sqcap C)$ we could still follow a left-to-right depth-first strategy, first confirming that $\exists P.C$ cannot be simplified, then simplifying $\top \sqcap C$ to C, then simplifying $\exists P.C \sqcap \exists P.C$ to $\exists P.C$.[8]

The class simplification rules can be stated most easily if we assume that intersections never have arguments that are also intersections: in other words, an expression like $C \sqcap (D \sqcap E)$ is flattened out to $C \sqcap D \sqcap E$. Otherwise it is harder to detect repetitions such as $C \sqcap (D \sqcap C)$ where the second C is nested further down. At present we use the following rules:

(a) If an intersection contains the argument $\bot$, replace the whole intersection by $\bot$ (e.g., $C \sqcap D \sqcap \bot \Rightarrow \bot$).

(b) If an intersection contains the argument $\top$ (one or more times), remove it (e.g., $C \sqcap \top \sqcap \top \Rightarrow C$).

(c) If an intersection contains the same argument more than once, remove all repetitions (e.g., $C \sqcap D \sqcap C \sqcap C \Rightarrow C \sqcap D$).

(d) If a restriction is defined over the argument $\bot$, replace the whole restriction by $\bot$ (e.g., $\exists P.\bot \Rightarrow \bot$).

If when applying any of these rules we reduce the arguments of an intersection to only one, then as usual we replace the intersection by this argument. Applying these rules to the example at the start of this section, we obtain:

$\exists P.C \sqcap \exists P.(\top \sqcap C)$
$\equiv \exists P.C \sqcap \exists P.C$ [by rule (b)]
$\equiv \exists P.C$ [by rule (c)]

A corresponding explanation in natural language could be given as follows:

> Your assertion can be simplified:
> **Step 1**
> Rover lives in a kennel and lives in a *thing that is a kennel*.
> Rover lives in a kennel and lives in a *kennel*.
> ('thing' adds nothing since it applies to everything)
> **Step 2**
> Rover *lives in a kennel* and *lives in a kennel*.
> Rover *lives in a kennel*.
> (the phrase in italics was repeated)
> The axiom added to the ontology is therefore: Rover lives in a kennel.

A more elegant approach to class redundancy would be to prove equivalence using the reasoner, thus dispensing with class simplification rules altogether.[9] In outline, this method would work as follows. First, list all class expressions that occur in the axiom, including constituents (thus $C \sqcap \top \sqsubseteq \exists P.\bot$ would have five class expressions, namely C, $\top$, $C \sqcap \top$, $\bot$, and $\exists P.\bot$). Next, for each pair (C_i, C_j) of classes, determine whether the subsumption relationships $C_i \sqsubseteq C_j$ and $C_j \sqsubseteq C_i$ both hold, in which case the classes are equivalent. Finally, for each pair of equivalent classes, determine whether one is simpler than the other, as measured for instance by the number of atomic terms it contains. In our example we will find that $C \sqcap \top$ (two atomic terms) is equivalent to C (one atomic term), and that $\exists P.\bot$ (two atomic terms) is equivalent to $\bot$ (one atomic term); the original axiom can therefore be simplified to $C \sqsubseteq \bot$. (Note that in checking these subsumption relationships the reasoner would be running on an empty ontology, because we are concerned with inherent redundancy, not contextual redundancy.) We still have to consider how this approach would allow us to simplify an axiom in stages, allowing a step-by-step explanation such as that given above.

After disaggregating (point 2 in figure 3), we obtain a set of potential axioms in which no classes are inherently redundant, and no statement has an intersection as its predicate class. The next task (point 3 of figure 3) is to remove any potential axioms that are inherently redundant (i.e., tautologies). Here there are just three cases for the logical fragment under consideration:

- The statement has a predicate class included in its subject class (e.g., it has the form $C \sqsubseteq C$, or $C \sqcap D \sqsubseteq C$, etc.).[10] Examples: 'Every dog is a dog', 'Every dog that lives in a kennel is a dog'.
- The statement's subject class is the bottom class $\bot$ (i.e., it has the form $\bot \sqsubseteq C$). Example: 'Every nonexistent entity is a dog'.[11]
- The statement's predicate class is the top class $\top$ (i.e., it has the form $C \sqsubseteq \top$). Example: 'Every dog is a thing'.

[8] One can find cases in which this strategy is inefficient, because it may lead to evaluation of a constituent that is later discarded. Applied for example to $((2 \times 3) + 4) \times 0$, it would waste time evaluating the left constituent to 10, before multiplying it by zero. But such refinements would take us too far afield.

[9] We are indebted to an anonymous reviewer for this suggestion.

[10] We say a class C_1 is *included in* a class C_2 either if C_2 is identical to C_1, or if C_2 is an intersection of classes one of which is C_1.

[11] This sentence sounds odd since there is no natural phrase for the bottom class in English, nor any reason to make generalisations about members of a class that has no members. Such statements are therefore very unlikely to occur in practice, and the rule is given only for completeness.

1. Check the subject and predicate classes of the new assertion, and simplify them if necessary to remove inherent class redundancy.
2. Check whether the resulting assertion can be disaggregated (i.e., whether its predicate class is an intersection), and if so, replace it by an equivalent set of statements which cannot be further disaggregated.
3. Check whether each statement in this set is inherently redundant (i.e., a tautology), and if so remove it.

Figure 3. Outline procedure for reformulating an assertion

6.2 Noting contextual redundancy

Having refactored the user's original assertion as a set of disaggregated statements containing no inherent redundancy, the next step is to check whether there are possible simplifications that depend on the context—that is, on the axioms already present. Recall that the purpose of these checks is to *advise* rather than *correct*: contextually-based simplifications are reported as warnings, but not directly implemented, in case the user later decides to retract the assertions on which they are based (see rules 8 and 9 in figure 2).

The process of simplifying a contextually redundant class is similar to that for an inherently redundant class: simplification rules are applied to a complex expression, using some navigation strategy such as left-right depth-first. In this case only one simplification rule is needed, and as before it can be stated most easily if we assume that embedded intersections such as $C \sqcap (D \sqcap E)$ are flattened out, in this case to $C \sqcap D \sqcap E$. follows:

> If an intersection of two or more classes contains two classes C and D for which the subsumption relationship $C \sqsubseteq D$ can be inferred from the ontology (either as an axiom or an entailment), remove D from the intersection.

Thus we may simplify 'dog that is owned by a farmer and lives in a kennel' to 'dog that is owned by a farmer' if the statement 'Every dog lives in a kennel' can be inferred from the axioms already asserted.

In principle a complication can occur here: what if two classes in an intersection are equivalent, so that we can infer both $C \sqsubseteq D$ and $D \sqsubseteq C$? Plainly we should remove either C or D, but not both; but how do we choose which? I would suggest, as a solution, removing whichever class is more complex or, if they are equally complex, removing the class that was introduced later. Complexity can be measured by counting the number of atomic terms (classes, individuals or properties) that occur in the class expression: thus for example 'dog that is owned by the queen' has complexity equal to 3 since it contains one atomic term of each kind.

Identification of contextually redundant statements is even simpler: a statement is contextually redundant if it can be inferred from the axioms already asserted, either as axiom or entailment.

7 ACCEPT OR REJECT

So far we have disaggregated, removed inherent redundancy, and noted contextual redundancy. The next step in computing the program's response is to decide, for each disaggregated statement, whether it should be added to the ontology. This requires consideration of two issues: first, does the statement duplicate an axiom already present; secondly, does the statement introduce a contradiction, as discussed in section 4.

As pointed out already, detecting duplication is facilitated by our policy of disaggregating the axioms proposed by the user. However, there remains a problem of what we should do when the new axiom is *equivalent* to an existing one, but not syntactically identical. Here again one can distinguish two kinds of equivalence, inherent and contextual. As an illustration of this distinction, suppose axioms 1-3 below have already been asserted, and consider options 4a-4b for axiom 4:

1	Every dog is a domestic canine.
2	Every domestic canine is a dog.
3	Every pet that is a dog is a pet dog.
4a	Every dog that is a pet is a pet dog.
4b	Every domestic canine that is a pet is a pet dog.

The question at issue is whether to accept axiom 4, or whether to reject it as equivalent to axiom 3. The answer might depend on whether the user asserts 4a or 4b. The equivalence between 3 and 4a is *inherent*, because it depends only on the commutivity of the intersection operator $\sqcap$, which means that the classes $C \sqcap D$ and $D \sqcap C$ will be equivalent whatever the values of C and D. The statements 3 and 4b are instead only *contextually* equivalent, because their equivalence depends on other axioms in the ontology, specifically on axioms 1 and 2. If one or both of these axioms was later retracted, axioms 3 and 4b would no longer be equivalent.

We would suggest, then, that a new statement should be rejected as a duplicate either if it is syntactically identical to an existing axiom, or inherently equivalent to it. For our logical fragment, inherent equivalence can be due only to a different ordering of the terms in an intersection.

The other reason for rejecting a statement is contradiction, and here we suggest the strict policy of rejecting all statements that introduce a contradiction, of whatever kind (inconsistency or incoherence, inherent or contextual). Probably the only debatable issue here is whether one should allow statements that introduce contextual incoherence—that is, a statement like 'Every corgi is a cat' in a context that already asserts that every corgi is a dog, and no dog is a cat. The trouble is that if a named class like *corgi* is unsatisfiable (i.e., equivalent to the bottom class $\bot$), then every statement of the form $corgi \sqsubseteq C$, for any C, can be inferred, so flooding the ontology with absurd entailments. Against this, it might be argued that no harm is done if the ontology is temporarily incoherent: perhaps the new statement should be admitted for now, but with advice on which existing axioms need to be retracted to restore coherence. This is another trade-off between quality of ontology and efficiency of process (see points 5 and 6 in figure 1).

8 FEEDBACK ON NEW ENTAILMENTS

We have discussed at some length responses by the program to assertions by the user that are in some way flawed—either redundant or contradictory, wholly or partially. Although we believe it is useful to perform these checks and corrections, we would not expect them to be needed often. Unless the user is particularly obtuse or eccentric, an authoring dialogue should consist mostly of non-redundant non-contradictory assertions which can be safely and efficiently added to the ontology. In this case, we assume that the program's reply should give helpful feedback on any new entailments that result from the latest assertion.

N	Assertion	Axioms and entailments
1	Rover is a dog	Rover is a dog
2	Every dog is an animal that lives in a kennel	Every dog is an animal Every dog lives in a kennel *Rover is an animal* *Rover lives in a kennel*
3	Every kennel is a shelter	Every kennel is a shelter
4	Every animal that lives in a shelter is a pet	Every animal that lives in a shelter is a pet *Every dog is a pet* *Rover is a pet*

Table 3. Adding axioms and their entailments

To illustrate this task, table 3 shows the development of an ontology (initially empty) as the user adds four assertions, numbered in the left column. The right column represents the program's growing knowledge, and includes not only all axioms asserted up to that point, possibly reformulated, but also some entailments that can be derived at each stage (these are shown in italics). One possible policy would be to respond to each new assertion by reporting all or some of the new entailments.

To implement such a policy, we need some method for generating useful entailments. A reasoner such as FACT++ [14] will correctly determine whether a specified statement is entailed, but there are always many such statements, most of them of no interest to the user. In row 4 of table 3, for example, we could have included the following, all new entailments:

Every dog that lives in a kennel is a pet.
Rover is a pet that lives in a shelter.
Rover is a pet that is a thing that is a thing . . . [etc.]

Why are these entailments less useful than those in table 3? We might answer this question by appealing to the concepts of redundancy and aggregation discussed above: for instance, by excluding entailments that contain inherently redundant classes like 'pet that is a thing', or aggregations such as 'pet that lives in a shelter'. However, we still need an efficient method for *generating* the potential entailments that satisfy these requirements.

The simplest approach to this problem, in our view, is to begin by listing all the classes that are essential in order to state the current set of axioms. Let us call this the set of *primary* classes. It will include all named classes, and also all classes constructed using intersection and existential restriction. To these we can add all classes $\{I\}$ that contain only a named individual I, thus reducing all statements to subsumption relationships between classes.[12] In a dialogue containing only assertions, the list of primary classes will grow as more assertions are added, as shown (again in English) in table 4.

Having determined the primary classes at a given stage in the dialogue, we can adopt the policy of reporting only entailments that express subsumption relationships among primary classes; we may call these *primary entailments*. This means that the reasoner only has to consider N^2 potential entailments for N primary classes (or more precisely, $N(N-1)$, since we can eliminate the trivial cases of the form $C \sqsubseteq C$). These could be represented by an array of $N \times N$ cells in which each cell $[i, j]$ corresponds to a potential entailment $C_i \sqsubseteq C_j$, and a tick inside a cell means that the subsumption relationship holds. The program could proceed by constructing such an array, ticking all cells along the diagonal ($i = j$), ticking all cells corresponding to axioms, and then submitting all cells that are still empty to the reasoner.

[12] Thus a class membership relationship $I \in C$ will be reduced to the equivalent subsumption relationship $\{I\} \sqsubseteq C$.

N	Assertion	Primary classes
1	Rover is a dog	Rover dog
2	Every dog is an animal that lives in a kennel	animal kennel lives in a kennel
3	Every kennel is a shelter	shelter
4	Every animal that lives in a shelter is a pet	lives in a shelter animal that lives in a shelter pet

Table 4. Adding primary classes

Having computed the set of primary entailments, the program must finally decide which are worth reporting. This is partly a subjective judgement, since it implies an evaluation of which statements are most interesting or helpful to the user. In the absence of empirical studies, we would suggest that the following factors might be relevant:

- Entailments that hold trivially should not be reported. In table 4, for example, 'animal that lives in a shelter' and 'animal' are both primary classes, yielding the entailment 'Every animal that lives in a shelter is an animal', an inherently redundant statement.
- Entailments should be given priority when they share either their subject class or their predicate class with the latest assertion, so that the user perceives a link between the two statements.
- On grounds of efficiency one might prefer entailments that depend on a larger set of axioms. On this basis, the entailment 'Rover is a pet' might be preferred to 'Every dog is a pet', which requires a less complex inference.

The first of these suggestions is intuitively obvious, but the others are debatable. We think the topic is worth exploring, not only in the context of ontology authoring, but as a skill relevant to all dialogues in which a listener wants to signal understanding.

9 CONCLUSION

We have sought precise rules for responding to an assertion, in a dialogue between an ontology author and a program with logical competence but no knowledge of the domain. We find that the program requires a surprising range of analytical skills in order to recognise possible flaws in the assertion and decide whether to accept it, either in its original form or in a revised form. When an assertion is accepted, another set of skills is brought into play, to decide which of many implications should be reported to the user. Ontology authoring provides a well-defined context in which these generic dialogue skills can be studied formally.

As will be obvious, the model presented here is based on logical analysis and common sense; we have not yet tried to ascertain empirically which response strategies are most helpful to users. This is a complex issue since user preferences are likely to vary, depending for instance on their familiarity with OWL. To support such an investigation we are now developing a prototype authoring system in which the response strategy can be configured: the user can decide, for example, whether the program will allow or disallow the various kinds of redundancy and contradiction.

By approaching the issue theoretically rather than empirically (at least in the first instance), we have covered some cases that are probably very rare in practice. Only the most eccentric of users would compose an assertion such as 'Every dog that is a dog lives in a kennel' which contains a class that is inherently redundant. However, for one reason or other such statements do occur in human discourse, and we believe it is worth taking a little trouble so that the program detects the redundancy, and corrects it if that is the prescribed strategy. Humans, we would argue, do this too. Before the 1997 general election in the UK, a well-known politician announced 'Our priorities are education, education, and education': surely only the most dull-witted student would transcribe such a sentence to his/her notebook without attempting a more concise reformulation.

It should be admitted, finally, that even within the restricted scope of our enquiry (responses to assertions), we have considered only the most direct responses, ignoring many kinds of indirect response which might occur in human conversation. To give just one example, consider responses based on *comparisons* between the current focus of attention and related topics covered earlier. Suppose that after describing dogs the user moves on to rabbits, and asserts that every rabbit lives in a hutch. With a little lateral thinking the program might draw attention to the *distinction* between rabbits and dogs ('A hutch as opposed to a kennel?'); it might even ask how hutches differ from kennels, if this has not already been stated. We have also grossly simplified by allowing only exchanges comprising two locutions (assertion followed by response), so excluding for example the follow-up sequences that would naturally occur when the program finds a contradiction (at the very least, the program could offer a choice of which axioms should be retracted to restore consistency or coherence). Thus for all its complexity, the present analysis only scratches the surface; what we find appealing is that the ontology authoring task can embrace so many conversational phenomena (comparisons included) in a formal application requiring no prior domain knowledge.

ACKNOWLEDGEMENTS

The paper has benefitted from comments by colleagues and referees, whom I thank for their help.

REFERENCES

[1] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, and ..., 'Gene ontology: tool for the unification of biology. The Gene Ontology Consortium.', *Nature Genetics*, **25**(1), 25–29, (May 2000).

[2] F. Baader, S. Brandt, and C. Lutz, 'Pushing the $\mathcal{EL}$ envelope', in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, Edinburgh, UK, (2005). Morgan-Kaufmann Publishers.

[3] James R. Campbell, 'Managing clinical terminology in a post-classification era', in *Representing and sharing knowledge using SNOMED. Proceedings of the 3rd International Conference on Knowledge Representation in Medicine KR-MED 2008*, ed., Kent Spackman Ronald Cornet, (2008).

[4] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler, 'Owl 2: The next step for owl', *Web Semant.*, **6**(4), 309–322, (November 2008).

[5] C. L. Hamblin, *Fallacies*, Methuen, London, UK, 1970.

[6] C. L. Hamblin, 'Mathematical models of dialogue', *Theoria*, **37**, 130–155, (1971).

[7] Glen Hart, Martina Johnson, and Catherine Dolbear, 'Rabbit: Developing a control natural language for authoring ontologies', in *ESWC*, pp. 348–360, (2008).

[8] Ian Horrocks, 'Ontologies and the semantic web', *Communications of the ACM*, **51**(12), 58–67, (December 2008).

[9] K. Kaljurand and N. Fuchs, 'Verbalizing OWL in Attempto Controlled English', in *Proceedings of OWL: Experiences and Directions*, Innsbruck, Austria, (2007).

[10] A. Parvizi, C. Jay, C. Mellish, J. Pan, Y. Ren, R. Stevens, and K. van Deemter, 'A Pilot Experiment in Knowledge Authoring as Dialogue', in *Proceedingsof the 10th International Workshop on Computational Semantics*, Potsdam, Germany, (2013).

[11] Richard Power, 'OWL Simplified English: A Finite-State Language for Ontology Editing.', in *CNL*, eds., Tobias Kuhn and Norbert E. Fuchs, volume 7427 of *Lecture Notes in Computer Science*, pp. 44–60. Springer, (2012).

[12] Rolf Schwitter and Thomas Meyer, 'Sydney OWL Syntax - towards a Controlled Natural Language Syntax for OWL 1.1', in *OWLED: OWL Experiences and Directions*, (2007).

[13] William Strunk, Jr. and E. B. White, *The Elements of Style*, Macmillan, third edn., 1979.

[14] D. Tsarkov and I. Horrocks, 'FaCT++ Description Logic Reasoner: System Description', in *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pp. 292–297. Springer, (2006).