# Justification Oriented Proofs in OWL

Matthew Horridge[1], Bijan Parsia[1], and Ulrike Sattler[1]

School of Computer Science
The University of Manchester

**Abstract.** Justifications — that is, minimal entailing subsets of an ontology — are currently the dominant form of explanation provided by ontology engineering environments, especially those focused on the Web Ontology Language (OWL). Despite this, there are naturally occurring justifications that can be very difficult to understand. In essence, justifications are merely the premises of a proof and, as such, do not articulate the (often non-obvious) reasoning which connect those premises with the conclusion. This paper presents justification oriented proofs as a potential solution to this problem.

## 1 Introduction and Motivation

Modern ontology development environments such as Protégé-4, the NeOn Toolkit, Swoop, and Top Braid Composer, allow users to request explanations for entailments (inferences) that they encounter when editing or browsing ontologies. Indeed, the provision of explanation generating functionality is generally seen as being a vital component in such tools. Over the last few years, *justifications* have become the dominant form of explanation in these tools. This paper examines justifications as a kind of explanation and highlights some problems with them. It then presents *justification lemmatisation* as a non-standard reasoning service, which can be used to augment a justification with *intermediate inference steps*, and gives rise to a structure known as a *justification oriented proof.* Ultimately, a justification oriented proof could be used as an input into some presentation device to help a person step though a justification that is otherwise too difficult for them to understand.

### 1.1 Justifications as Explanations

A justification is a minimal subset of an ontology (a set of axioms) that is sufficient for a given entailment to hold. As an example, consider the small ontology $\mathcal{O} = \{A \sqsubseteq B, A(i), C \sqsubseteq D\}$, which *entails* $B(i)$, written $\mathcal{O} \models B(i)$[1]. A justification $\mathcal{J}$ for $\mathcal{O} \models B(i)$ is a *minimal* subset of $\mathcal{O}$ that entails $B(i)$, in this case $\mathcal{J} = \{A \sqsubseteq B, A(i)\}$.

The major benefit of justifications is that they pinpoint and isolate the handfuls of axioms, in what could be a very large ontology, that cause the entailment

[1] $B(i)$ means $i$ is an instance of $B$

to hold. For example, the SNOMED medical ontology contains roughly 400,000 axioms, but a justification for an entailment in this ontology is on average less than ten axioms in size [2].

Unlike full blown proofs, justifications are conceptually simple structures with a natural relation to the ontology development process—they are directly related to what has been asserted or stated in an ontology. Justifications require very little additional knowledge beyond the semantics of the language. This conceptual simplicity, coupled with the fact that the computation of justifications for real ontologies tends to be practical [10], and the fact that off-the-shelf implementations of justification finding services exist, has most likely lead to the large uptake of justifications as a type of explanation.

**Fig. 1.** A justification for $\mathsf{Person} \sqsubseteq \bot$

$\mathsf{Person} \sqsubseteq \neg\mathsf{Movie}$
$\mathsf{RRated} \sqsubseteq \mathsf{CatMovie}$
$\mathsf{CatMovie} \sqsubseteq \mathsf{Movie}$
$\mathsf{RRated} \equiv (\exists \mathsf{hasScript}.\mathsf{ThrillerScript})$
$\quad \sqcup (\forall \mathsf{hasViolenceLevel}.\mathsf{High})$
$\mathsf{Domain}(\mathsf{hasViolenceLevel},\ \mathsf{Movie})$

**Fig. 2.** A justification for $\mathsf{Newspaper}(\mathsf{DailyMirror})$

$\mathsf{InverseProperties}(\mathsf{hasPet},\ \mathsf{isPetOf})$
$\mathsf{isPetOf}(\mathsf{Rex}, \mathsf{Mick})$
$\mathsf{Domain}(\mathsf{hasPet},\ \mathsf{Person})$
$\mathsf{Male}(\mathsf{Mick})$
$\mathsf{reads}(\mathsf{Mick}, \mathsf{DailyMirror})$
$\mathsf{drives}(\mathsf{Mick}, \mathsf{Q123ABC})$
$\mathsf{Van}(\mathsf{Q123ABC})$
$\mathsf{Van} \sqsubseteq \mathsf{Vehicle}$
$\mathsf{WhiteThing}(\mathsf{Q123ABC})$
$\mathsf{Driver} \equiv \mathsf{Person} \sqcap \exists \mathsf{drives}.\mathsf{Vehicle}$
$\mathsf{Driver} \sqsubseteq \mathsf{Adult}$
$\mathsf{Man} \equiv \mathsf{Adult} \sqcap \mathsf{Male} \sqcap \mathsf{Person}$
$\mathsf{WhiteVanMan} \equiv \mathsf{Man} \sqcap \exists \mathsf{drives}.(\mathsf{Van} \sqcap \mathsf{WhiteThing})$
$\mathsf{WhiteVanMan} \sqsubseteq \forall \mathsf{reads}.\mathsf{Tabloid}$
$\mathsf{Tabloid} \sqsubseteq \mathsf{Newspaper}$

## 1.2 Problems With Justifications

However, despite the fact that justifications are a popular form of explanation in the OWL world, observations show there are justifications that people find difficult or impossible to understand. Indeed, the justifications shown in Figure 1 and Figure 2, both from real ontologies, gave many users trouble when trying to understand how they lead to their respective entailments. Indeed, some people questioned whether the justification shown in Figure 1 was a justification at all.

In the case of the justification shown in Figure 1, which is a justification for $\mathsf{Person} \sqsubseteq \bot$[2] spotting that the justification entails $\mathsf{Movie} \equiv \top$[3] is key to understanding how the justification works. Since everything is entailed to be a Movie, and Person is disjoint with Movie, Person is disjoint with $\top$, hence Person is un-

[2] $\bot$ is read as "bottom" and is the same as `owl:Nothing`
[3] $\top$ is read as "top" and is the same as `owl:Thing`

satisfiable. People who fail to realise that $\mathsf{Movie} \equiv \top$ is also entailed generally fail to understand how the justification gives rise to the entailment.

Similarly, the justification shown in Figure 2, is also rather difficult for people to work through. There are fifteen axioms of many different types in the justification. It is far from obvious how these axioms interplay with each other to result in the entailment $\mathsf{Tabloid(DailyMirror)}$. When a user works through this justification, they have to spot intermediate entailments, for example, $\mathsf{WhiteVanMan(Mick)}$ and $\mathsf{Person(Mick)}$, in order to arrive at the conclusion $\mathsf{Tabloid(DailyMirror)}$).

In a exploratory study [5], it was observed that many justifications *for entailments of interest in real ontologies* can be understood by people with a variety of backgrounds, and these kinds of justifications serve extremely well as explanations. However, it was also observed that there are justifications that are difficult or impossible for people to work through. Two obvious reasons for this are: (1) People do not spot key entailments within justifications, that are necessary for them to understand how the justification works (as is the case with the justification in Figure 1, and (2) People find large justifications, with many types of axioms, tedious and therefore difficult to work through (as is the case with the justification in Figure 2). In other words, when people fail, or find it difficult, to spot *intermediate entailments, conclusions or steps* they can fail to understand why a justification supports the entailment in question, and hence fail to understand why the entailment in question holds in their ontology.

### 1.3 From Justifications Towards Proofs

The above notion of "intermediate steps" that could guide a person through understanding a justification, raises the question of whether full blown proofs, such as natural deduction style proofs with inference rules, should be used for explaining entailments in OWL ontologies.

One of the typical claims about natural deduction is that it mimics human reasoning—that is, it has a *strong cognitive adequacy* [18]. However, there is ongoing debate in the field of cognitive psychology about how human reasoning actually works. Some camps favour a "logic" or rule based account [16], while others favour a "model" based account [8]. Even for simple cases of natural language based deduction, it is unclear which account is correct. Moreover, other research [14] shows that relatively untrained people—clearly without having a complete set of deduction rules at their disposal—can successfully work through surprisingly complex reasoning puzzles. It is therefore impossible to say whether or not natural deduction and similar proof systems mimic human reasoning. What is clear, is that representations that have a strong cognitive adequacy are not necessarily useable. Hence, even if natural deduction has a strong cognitive adequacy, there is no guarantee that it is usable as a form of explanation for entailments in ontologies.

In summary, it is likely that natural deduction style proofs are not necessarily the best form of explanation. On the other hand, justifications are an appealing type of explanation. It is known that that a wide range of people can cope with justifications [5]. This includes domain experts who have very little training or background with the Description Logics that underpin OWL. Justifications appeal to these kinds of people because they are conceptually simple—very little training is needed in order to understand how justifications work. The same cannot be said about natural deduction style proofs. Additionally, people are used to seeing axioms, albeit in a frame-based style of presentation, and justifications reflect this familiarity. If natural deduction style proofs were presented to people such as domain experts, they would require special training in order to read the proofs.

## 1.4 Justification Oriented Proofs

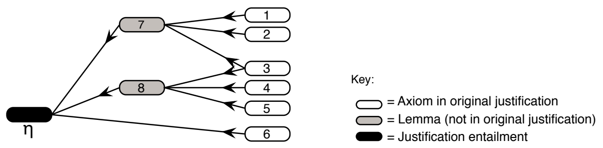
What is needed, is something that lies between justifications and proofs. Given the popularity and conceptual simplicity of justifications, the work presented in this paper uses them as building blocks for structures that begin to look like proofs, but are independent of any calculus or deduction rules. In essence, intermediate steps are introduced into a justification, which are themselves explained with justifications. This results in a directed acyclic proof graph of the form shown in Figure 3. Ultimately a justification is extended with "lemmas" into a *justification oriented proof.*

The main idea behind a justification oriented proof is depicted in Figure 3. The numbered lozenges represent axioms, with the leftmost lozenge, labelled $\eta$, representing the entailment of interest. The white lozenges labelled with "1" – "6" represent exactly the axioms that appear in the original justification $\mathcal{J}$ for the entailment (and are therefore in the ontology as asserted axioms). Grey shaded lozenges represent *lemmas* that are entailed by the deductive closure of $\mathcal{J}$ but are not in $\mathcal{J}$ as asserted axioms. For a given node, its direct predecessors constitute a justification for that node. This produces a weakly connected directed acyclic graph, with one sink node that represents the entailment of interest and a source node for each axiom in the justification. Hence, in the example shown in Figure 3, $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ is a justification for $\eta$ with respect to the ontology that entails $\eta$. Axiom 7 is a lemma for axioms 1, 2 and 3 (conversely, axioms 1, 2 and 3 are a justification for axiom 7). Axiom 8 is a lemma for axioms 3, 4 and 5 (conversely axioms 3, 4 and 5 are a justification for axiom 8). Together axioms 6, 7 and 8 constitute a justification for $\eta$ i.e. the entailment. Notice that axiom 3 participates in different justifications for different lemmas.

Ultimately, the justification oriented proof guides a person through the understanding of the original justification. In essence, lemmas are intermediate steps that may be non-obvious, but may be significant to understanding how the justification results in the entailment. They also provide a chunking mechanism, which can help guide a user through a large and tedious to understand justification.

## 1.5 Contributions

The *main contribution* of this paper is the novel *framework* that is presented for *constructing* justification oriented proofs. This framework is rather different to



1
2
3
4
5
6
7
8
η
Key:
= Axiom in original justification
= Lemma (not in original justification)
= Justification entailment

**Fig. 3.** A schematic of a Justification Oriented Proof — Predecessors of a node represent a justification for that node.

other approaches: First, the framework *does not use any deduction rules* per se to derive the intermediate steps or conclusions. The choice of steps is ultimately governed by a *pluggable* justification complexity model which is used to choose one justification over another during the proof construction. Details of a *practical* model are supplied in this paper, but it is important to realise that this paper shows that the idea of using *a model* to select intermediate steps works well in practice. Second, the framework is entirely *black-box based*. Any reasoner, such as FaCT++, HermiT, Pellet or Racer, that implements a decision procedure for entailment checking for OWL 2 (or any other monotonic logic) may be used for generating the proofs. In other words, the internals of the reasoner need not be modified to extract some kind of intermediate proof. Third, and finally, the presented framework ought to be easily adaptable to deal with other fragments of First Order Logic that may or may not overlap the fragment that corresponds to OWL 2.

## 2 Related Work

The idea of using proofs as forms of explanation is obviously not new. Indeed, in some camps [3, 11], proofs are essentially regarded as *the* main form of explanation. However, the work that is presented in this paper is based on the intuitions mentioned in the introduction. That is, it is arguably more practical and more helpful to *not* show full blown proofs because (1) users already know and understand justifications, and (2) it avoids having to teach users a new calculus or deduction rules.

In [7], Huang acknowledges that Natural Deduction proofs are too fine-grained to be used as explanations, and introduces Natural Deduction Style Proofs at the assertional level, where trivial steps are eliminated from proofs. Parallels may be drawn with the basic motivations presented here. The main difference here is that the proofs here are arguably targeted at an even higher level of abstraction, and that an entirely black-box complexity model based approach is used to generate the proofs rather than extracting them from a theorem prover.

Finally, in [17] Schlobach introduces optimal interpolants, and so called *illustrations* that are intended to bridge the gap between subsumee and subsumer class expressions. The notion of lemmas and justifications oriented proofs as presented here are in the spirit of Schlobach's illustrations. However, the main difference is that Schlobach's work primarily deals with subsumption between

**Table 1.** OWL 2 Class, Object Property and Individual Axioms

| | | |
|---|---|---|
| $C \sqsubseteq D$ | $C \equiv D$ | DisjointClasses$(C_1, \ldots, C_n)$ |
| DisjointUnion$(C, D_1, \ldots, D_n)$ | | |
| $R \sqsubseteq S$ | $R \equiv S$ | DisjointProperties$(R_1, \ldots, R_n)$ |
| InverseProperties$(R, S)$ | Domain$(R, C)$ | Range$(R, C)$ |
| Functional$(R)$ | InverseFunctional$(R)$ | Transitive$(R)$ |
| Symmetric$(R)$ | Asymmetric$(R)$ | Reflexive$(R)$ |
| Irreflexive$(R)$ | | |
| $C(a)$ | $R(a, b)$ | DifferentIndividuals$(a, ..., a_n)$ |
| SameIndividual$(a_1, \ldots, a_n)$ | | |

two class expressions in isolation, whereas the work presented here deals with arbitrary entailments that arise from a sets of axioms.

## 3 Preliminaries

*OWL 2 and Description Logics* The work presented in this paper focuses on OWL 2. OWL 2 [12] is the latest standard in ontology languages from the W3C. An OWL 2 ontology roughly corresponds to a $\mathcal{SROIQ}(D)$ [6] knowledge base. For the purposes of this paper, an *ontology* is regarded as a finite set of axioms $\{\alpha_0, \ldots, \alpha_n\}$ of the form shown in Table 1[4], where $C$ and $D$ are (possibly complex) class expressions, $R$ and $S$ are (possibly inverse or complex) properties, and $a$ and $b$ are individuals. (Note that subscripts are used to represent different occurrences, or class expressions, properties etc.).

**Definition 1 (Justification).** *$\mathcal{J}$ is a justification for $\mathcal{O} \models \eta$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$ and for all $\mathcal{J}' \subsetneq \mathcal{J}$ $\mathcal{J}' \not\models \eta$.*

By a slight abuse of notation, the nomenclature used in this paper also refers to a minimally entailing set of axioms (that is not necessarily a subset of an ontology) as a justification.

*The Structural Transformation — $\delta$* Much of the work presented in the remainder of the paper uses the "well known" structural transformation — referred to here as $\delta$. This transformation takes a set of axioms and flattens out each axiom by introducing names for sub-concepts, transforming the axioms into an equisatisfiable set of axioms. The structural transformation was first described in Plaisted and Greenbaum [15], with a version of the rewrite rules for description logics given in [13]. For the sake of brevity, the structural transformation is not defined here — the interested reader is referred to [13, 4] for a full definition.

[4] For the sake of brevity, axioms involving data properties and data ranges are not presented here. However, the framework extends to these axioms in the obvious way.

## 4 Proof Generation Framework

In what follows the framework for generating justification oriented proofs is presented. The framework consists of two main ideas: (1) The notion of justification lemmatisation. Subsets of a justification may be replaced with simple summarising axioms, which are known as *lemmas*. One justification is lemmatised into another justification. (2) The notion of stitching a series of lemmatised justifications into a justification oriented proof. First a definition of justification lemmatisation is presented and then a definition for justification oriented proofs is given.

### 4.1 Justification Lemmatisation

Given a justification $\mathcal{J}$ for an entailment $\eta$, the aim is to lemmatise $\mathcal{J}$ into $\mathcal{J}'$, so that *$\mathcal{J}'$ is less complex by some measure* and for *some purpose* than $\mathcal{J}$. With this notion in hand, lemmas for justifications can now be defined. First, an informal description is given, then a more precise definition is given in Definition 3.

Informally, a set of lemmas $\Lambda_{\mathcal{S}}$ for a justification $\mathcal{J}$ for $\eta$ is a set of axioms that is entailed by $\mathcal{J}$ which can be used to replace some set $\mathcal{S} \subseteq \mathcal{J}$ to give a new justification $\mathcal{J}' = (\mathcal{J} \setminus \mathcal{S}) \cup \Lambda_{\mathcal{S}}$ for $\eta$. If, additionally, $\mathcal{J}'$ is less complex, by some measure, than $\mathcal{J}$. $\mathcal{J}'$ is called a *lemmatisation of $\mathcal{J}$*.

Various restrictions are placed on the generation of the set of lemmas $\Lambda_{\mathcal{S}}$ that can lemmatise a justification $\mathcal{J}$. These restrictions prevent "trivial" lemmatisations, an example of which will be given below. Before these restrictions are discussed, it is useful to introduce the notion of a *tidy* set of axioms.

Intuitively, a set of axioms is *tidy* if it is consistent, contains no synonyms of $\bot$ (where a class name is a synonym of $\bot$ with respect to a set of axioms $\mathcal{S}$ if $\mathcal{S} \models A \sqsubseteq \bot$), and contains no synonyms of $\top$ (where a class name is a synonym of $\top$ with respect to a set of axioms $\mathcal{S}$ if $\mathcal{S} \models \top \sqsubseteq A$).

**Definition 2 (Tidy sets of axioms).** *A set of axioms $\mathcal{S}$ is* tidy *if $\mathcal{S} \not\models \top \sqsubseteq \bot$, $\mathcal{S} \not\models A \sqsubseteq \bot$ for all $A \in Signature(\mathcal{S})$, and $\mathcal{S} \not\models \top \sqsubseteq A$ for all $A \in Signature(\mathcal{S})$.*

The definition of lemmatisation that follows, mandates that a set of lemmas $\Lambda_{\mathcal{S}}$ must only be drawn from (i) the deductive closure of *tidy* subsets of the set $\mathcal{S} \subseteq \mathcal{J}$, (ii) from the *exact* set of synonyms of $\bot$ or $\top$ over $\mathcal{S}$.

Without the above restrictions on the axioms in $\Lambda_{\mathcal{S}}$, it would be possible to lemmatise a justification $\mathcal{J}$ to produce a justification $\mathcal{J}'$ that, in isolation, is simple to understand, but otherwise bears little or no resemblance to $\mathcal{J}$. For example, consider $\mathcal{J} = \{A \sqsubseteq \exists R.B,\ B \sqsubseteq E \sqcap \exists S.C,\ B \sqsubseteq D \sqcap \forall S.\neg C\}$ as a justification for $A \sqsubseteq \bot$. Suppose that *any* axioms entailed by $\mathcal{J}$, could be used as lemmas (i.e. there are no restrictions on the axioms that make up $\Lambda_{\mathcal{S}}$). In this example, $A$ is unsatisfiable in $\mathcal{J}$, meaning that it would be possible for $\mathcal{J}' = \{A \sqsubseteq E, A \sqsubseteq \neg E\}$ to be a lemmatisation of $\mathcal{J}$. Here, $\mathcal{J}'$ is arguably easier to understand than $\mathcal{J}$, but bears little resemblance to $\mathcal{J}$. In other words, $A \sqsubseteq E$

and $A \sqsubseteq \neg E$ are not helpful lemmas for $\mathcal{J} \models A \sqsubseteq \bot$. Similarly unhelpful results arise if lemmas are drawn from *inconsistent* sets of axioms, or sets of axioms that contain synonyms for $\top$.

Given the above intuitions and the notion of tidy sets of axioms, the notion of justification lemmatisation is defined as follows:

**Definition 3 (Justification Lemmatisation).** *Let $\mathcal{J}$ be a justification for $\eta$ and $\mathcal{S}$ a set of axioms such that $\mathcal{S} \subseteq \mathcal{J}$. Let $\Theta_{\mathcal{S}}$ be the set of tidy subsets of $(\mathcal{S} \cup \delta(\mathcal{S}))$. Recall that $\mathcal{T}^{\star}$ is the deductive closure of a set of axioms $\mathcal{T}$. Let*

$$\Lambda_{\mathcal{S}} \subseteq \bigcup_{\mathcal{T} \in \Theta_{\mathcal{S}}} \mathcal{T}^{\star} \;\cup\; \{\alpha \mid \alpha \textit{ is of the form } A \sqsubseteq \bot \textit{ or } \top \sqsubseteq A, \textit{ and } \exists \mathcal{K} \subseteq (\mathcal{S} \cup \delta(\mathcal{S})) \textit{ that is consistent and } \mathcal{K} \models \alpha\}$$

*$\Lambda_{\mathcal{S}}$ is a* set of lemmas *for a justification $\mathcal{J}$ for $\eta$ if, for $\mathcal{J}' = (\mathcal{J} \setminus \mathcal{S}) \cup \Lambda_{\mathcal{S}}$*

1. *$\mathcal{J}'$ is a justification for $\eta$ over $\mathcal{J}^{\star}$, and,*
2. *$Complexity(\eta, \mathcal{J}') < Complexity(\eta, \mathcal{J})$.*

The ability to lemmatise one justification into another justification *is a key process* in constructing a justification oriented proof. Given a regular justification $\mathcal{J}$ for $\eta$, $\mathcal{J}$ can be lemmatised into $\mathcal{J}_1$ for $\eta$. The axioms in $\mathcal{J}_1$ may then be inspected to determine which of them are lemmas – lemmas are axioms that are not in $\mathcal{J}$. Given a lemma $\alpha \in \mathcal{J}_1$ ($\alpha \notin \mathcal{J}$) a new justification $\mathcal{J}_2 \subseteq \mathcal{J}$ for $\alpha$ can be identified. If necessary, $\mathcal{J}_2$ can then be lemmatised into a simpler justification for $\alpha$. Axioms in $\mathcal{J}_2$ can then be inspected and the process can be repeated as necessary. Ultimately the process builds up a justification oriented proof. Justification oriented proofs are defined as follows:

**Definition 4 (Justification Oriented Proof).** *A justification oriented proof for a justification $\mathcal{J}$ for an entailment $\eta$ in $\mathcal{O}$ is a weakly connected directed acyclic graph $G = (V, E)$ such that $\mathcal{J} \subseteq V \subset \mathcal{J}^{\star}$ and either, $G = (\{\eta\}, \{\langle \eta, \eta \rangle\})$ or,*

1. *$\eta$ is the one and only sink node in $G$,*
2. *$\mathcal{J}$ is the exact set of source nodes in $G$, and*
3. *For a given node, the set of predecessor nodes are a justification for the node over $\mathcal{J}^{\star}$.*

In summary, as shown in Figure 3, a node in a justification oriented proof that has incoming edges, is either a lemma or the entailment (sink node) itself. Source nodes (nodes with no predecessors) are the axioms in the original justification. Finally, given one justification $\mathcal{J}$ for $\eta$, there may be *multiple* justification oriented proofs, even if the set of lemmas in the proof is fixed.

It should be noted that, in the same way that raw unordered justifications are not presented directly to end users, it is unlikely the graph which constitutes a justification oriented proof should be presented *directly* to end users. Instead, the graph can be used as an input into some interactive presentation device.

## 4.2 Complexity Models

As can be seen from Definition 3, justification lemmatisation depends upon the notion of justification complexity. More specifically, it depends upon whether one justification is more complex, by some measure and for some purpose, than another justification. In this framework, *complexity models* are used to assign complexity scores to justifications and determine whether one justification is more complex, than another. The framework makes *no commitment to a particular complexity model.* Indeed, models are intended to be pluggable. A model may depend upon the application in question and the intended audience. In the work presented here, the primary aim is to produce justification oriented proofs, which pick out difficult to spot lemmas, and chunk and summarise sets of heterogeneous axioms in justifications. With these goals in mind, a simple model is presented later in this paper. However, before this model is presented, models that deal with special use cases are first discussed. The main intention here, is to give a feel for how different models can be appropriate for different applications, and how different models may be plugged into the framework.

**A Model for Deriving Proofs for Laconic Justifications** A laconic justification [4] is a justification whose axioms have no superfluous parts and whose parts are as weak as possible. Given $\mathcal{O} \models \eta$, a laconic justification oriented proof consists of a sink node $\eta$, and predecessors of $\eta$ which are either (1) leaf nodes representing axioms contained in $\mathcal{O}$, or (2) are nodes representing axioms entailed by $\mathcal{O}$, for which each one has a predecessor representing an axiom contained in $\mathcal{O}$. Given a justification $\mathcal{J}$ for $\eta$, a simple complexity model for computing such proofs assigns a score of zero to $(\mathcal{J}', \eta)$ if $\mathcal{J}'$ is a laconic justification for $\eta$, a score of zero to $(\mathcal{J}', \alpha)$ if $\alpha \neq \eta$ and $\alpha$ is in the laconic justification in question, and $\mathcal{J}'$ is a singleton set containing an axiom from the original ontology, and otherwise, a score of one.

**A Model for Deriving Proofs for Root/Derived Unsatisfiable Classes** Given an ontology $\mathcal{O}$ which contains unsatisfiable classes ($\mathcal{O} \models A \sqsubseteq \bot$ for some class name $A$ in the signature of $\mathcal{O}$), a root unsatisfiable class [9] is a class in the signature of $\mathcal{O}$ whose unsatisfiability does not depend on the unsatisfiability of any other class in the signature of $\mathcal{O}$. A derived unsatisfiable class is a class whose unsatisfiability depends on the unsatisfiability of some other class in the signature of $\mathcal{O}$. More precisely, given $\mathcal{O} \models A \sqsubseteq \bot$, $A$ is a derived unsatisfiable class if there exists some class $B$ such that $\mathcal{O} \models B \sqsubseteq \bot$ and there is a justification $\mathcal{J}_A \models A \sqsubseteq \bot$ and another justification $\mathcal{J}_B \models B \sqsubseteq \bot$ such that $\mathcal{J}_B \subsetneq \mathcal{J}_A$, otherwise, $A$ is a root unsatisfiable class.

A suitable model that will lemmatise and "collapse" a subset that corresponds to a justification for a root unsatisfiable class (corresponding to $\mathcal{J}_B$ above) is as follows: Given $\mathcal{O} \models A \sqsubseteq \bot$, the model assigns a score of 1 to a justification $\mathcal{J}_A$ for $\mathcal{O} \models \eta$ if there exists a justification $\mathcal{J}' \subset \mathcal{J}$ for $\mathcal{J} \models B \sqsubseteq \bot$, where $\mathcal{J}' \neq \{B \sqsubseteq \bot\}$ and $\mathcal{J}'' = \mathcal{J} \setminus \mathcal{J}' \cup \{B \sqsubseteq \bot\}$ is a justification for $A \sqsubseteq \bot$ over the deductive closure of $\mathcal{O}$, the model otherwise assigns a score of 0.

### 4.3 A General Model for Deriving Justification Oriented Proofs

For the purposes of introducing non-obvious and summarising intermediate steps into justifications, a simple justification complexity model is presented in Table 2. This model was derived partly from intuitions on what makes justifications difficult to understand, and partly from the observations made during a pilot/exploratory study [5] in which people attempted to understand justifications from real ontologies. The model uses various components to produce complexity scores which are summed to produce an overall complexity score for a justification. Broadly speaking, there are two types of components: (1) Structural components, such as C1, which require a syntactic analysis of a justification, and (2) Semantic components, such as C4, which require entailment checking to reveal non-obvious phenomena. Although the model presented in Table 2 is rather simple, it is surprisingly effective in that it produces pleasing justification oriented proofs.

## 5 An Algorithm for Generating Proofs

Given the above definitions, the main algorithms for generating proofs are presented below. There are three main algorithms: 1) GenerateProof, which takes a justification as an input and outputs a proof; 2) LemmatiseJustification, which takes a justification as an input and outputs either a lemmatised justification or the justification itself; 3) ComputeJPlus, which takes a justification and computes a set of axioms that are in the deductive closure of tidy subsets of the justification from which lemmas may be drawn. The GenerateProof algorithm uses the LemmatiseJustification as a sub-routine, and the LemmatiseJustification algorithm uses the ComputeJPlus algorithm as a sub-routine. Note that due to space limitations, the ComputeJPlus algorithm is not specified line by line in this paper—instead, a definition of $\mathcal{J}^+$ (Definition 5) is given below, and it is assumed that the algorithm simply computes $\mathcal{J}^+$ in accordance with this definition.

### 5.1 GenerateProof

The GenerateProof algorithm for computing justification oriented proofs is depicted in Figure 4. The basic idea is that, given an input of a justification $\mathcal{J}$ for $\eta$, a lemmatised justification $\mathcal{J}'$ for $\eta$ is computed. $\mathcal{J}'$ is then used to initialise a justification oriented proof $\mathcal{P}$. For each node $\lambda$ in the proof corresponding to an axiom in $\mathcal{J}'$, if $\lambda$ is not in $\mathcal{J}$ then it is a lemma and a justification needs to be computed for it. In this case a new justification $\mathcal{J}''$ is computed for $\alpha'$ over $\mathcal{J}$. Next, $\mathcal{J}''$ is lemmatised to give $\mathcal{J}'''$ which is inserted into the proof $\mathcal{P}$. The process then repeats for lemmas in $\mathcal{P}$ that do not have any predecessors until none of the leaves in the proof are lemmas. Although not depicted in Figure 4, it is important to note that, in order to comply with Definition 4, there is a test in step 6 to determine whether inserting $\mathcal{J}'''$ as a result of the lemmatisation

**Table 2.** A Simple Complexity Model for Generating Justification Oriented Proofs

| **Name** | **Description** |
|---|---|
| **C1** AxiomTypes | Counts the axiom types in $\mathcal{J}$ and $\eta$. The count is multiplied by a weighting (10.0) and added to the overall complexity score. |
| **C2** ClassConstructors | Counts the class constructors in $\mathcal{J}$ and $\eta$. The count is multiplied by a weighting (10.0) and added to the overall complexity score. |
| **C3** UniversalImplication | If $\alpha \in \mathcal{J}$ and $\alpha$ is of the form $\forall R.C \sqsubseteq D$ or $D \equiv \forall R.C$ a constant (50.0) is added to the overall complexity score. |
| **C4** SynonymOfThing | If $\mathcal{J} \models \top \sqsubseteq A$ for some $A \in \mathsf{Signature}(\mathcal{J})$ and $\top \sqsubseteq A \notin \mathcal{J}$ and $\top \sqsubseteq A \neq \eta$ then a constant (50.0) is added to the complexity score. |
| **C4** SynonymOfNothing | If $\mathcal{J} \models A \sqsubseteq \bot$ for some $A \in \mathsf{Signature}(\mathcal{J})$ and $A \sqsubseteq \bot \notin \mathcal{J}$ and $A \sqsubseteq \bot \neq \eta$ then a constant (50.0) is added to the complexity score. |
| **C5** DomainAndNoExistential | If $\mathsf{Domain}(R, C) \in \mathcal{J}$ and $\mathcal{J} \not\models E \sqsubseteq \exists R.D$ for some class expressions $E$ and $D$ then a constant (50.0) is added to the complexity score. |
| **C6** ModalDepth | The maximum modal depth of all class expressions in $\mathcal{J}$ is multiplied by a weighting (50.0) and added to the overall complexity score |
| **C7** SignatureDifference | For each $A \in \mathsf{Signature}(\eta)$, where $A \notin \mathsf{Signature}(\mathcal{J})$ a weighting (50.0) is added to the overall complexity score |
| **C8** AxiomTypeDifference | If the axiom type of $\eta$ is not the set of axiom types of $\mathcal{J}$ then a weighting (50.0) is added to the overall complexity score |
| **C9** ClassConstructorDifference | For each class constructor in $\eta$ that is not in the set of class constructors of $\mathcal{J}$, a weighting (50.0) is added to the overall complexity score |

process into $\mathcal{P}$ would result in a cyclic graph instead of a DAG. If this is the case, then an alternative lemmatisation of $\mathcal{J}''$ must be chosen (or if there are no alternatives then $\mathcal{J}''$ itself must be chosen) to insert into $\mathcal{P}$. This enforcement of non-cyclical proofs is also part of the mechanism that ensures the GenerateProof algorithm terminates. A discussion on termination is presented later.

## 5.2 LemmatiseJustification

The LemmatiseJustification algorithm is presented in Algorithm 1. The algorithm takes a justification $J$ for $\eta$ as its input and returns a justification $L$ as its output. Either $L$ is a lemmatisation of $J$ or $L$ is equal to $J$. In essence, the algorithm produces a lemmatised justification by computing a *filter* $S$ on the deductive closure of tidy subsets of $J$, which obviously includes axioms that

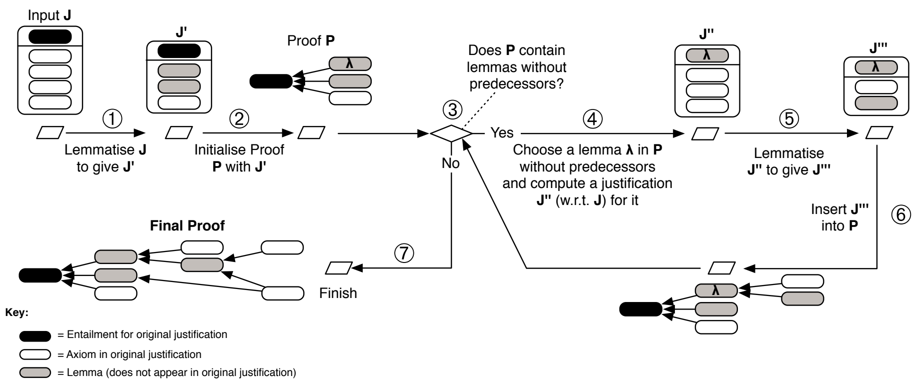

Input J
J'
Proof P
Does P contain lemmas without predecessors?
J''
J'''
λ
1
Lemmatise J to give J'
2
Initialise Proof P with J'
3
Yes
No
4
Choose a lemma λ in P without predecessors and compute a justification J'' (w.r.t. J) for it
5
Lemmatise J'' to give J'''
6
Insert J''' into P
7
Finish
Final Proof
Key:
= Entailment for original justification
= Axiom in original justification
= Lemma (does not appear in original justification)

**Fig. 4.** GenerateProof – A Depiction of an Algorithm for Generating Justification Oriented Proofs. Justification Lemmatisation is used as a Sub-routine.

could lemmatise $J$. Justifications for $\eta$ are then computed with respect to $S$. A complexity score is computed for each justification $L \subseteq S$, which is compared to the complexity of $J$. If the difference between the score for $J$ and the score for $L$ is positive then $L$ is selected as a lemmatisation of $J$. Algorithm 1 always terminates due to the fact that $S$ is finite in size and hence there are a finite number of justifications for $\eta$ with respect to $S$.

### 5.3 ComputeJPlus

Definition 3 mandates that, for a justification $\mathcal{J}$, lemmas must be drawn from the deductive closure of tidy subsets of $\mathcal{J}$. However, the deductive closure of a set of axioms is *infinite*. For practical purposes it is necessary to work with a finite representative of the deductive closure that suffices for computing pleasing lemmatisations and pleasing justification oriented proofs. In addition to these practicalities, a finite representation of the deductive closure is needed because the ability to draw lemmas from an infinite set of axioms could lead to non-termination of the GenerateProof algorithm. In order to ensure termination, not only is it necessary to disallow cycles in the proof, but it is also necessary to

**Algorithm 1** LemmatiseJustification$(J, \eta)$

```
Function-1: LemmatiseJustification$(J, \eta)$
1: $S \leftarrow$ ComputeJPlus$(J, \eta) \setminus \{\eta\}$
2: $justs \leftarrow$ ComputeJustifications$(S, \eta)$
3: $c_1 \leftarrow$ ComputeComplexity$(J, \eta)$
4: $L \leftarrow J$
5: for $J' \in justs$ do
6:     $c_2 \leftarrow$ ComputeComplexity$(J', \eta)$
7:     if $c_2 < c_1$ then
8:         $L \leftarrow J'$
9: return $L$
```

introduce a filter on the deductive closure that produces a *finite* set of axioms, $\mathcal{J}^+$ from which lemmas may be drawn. In essence, $\mathcal{J}^+$ is some finite subset of the deductive closure of $\mathcal{J}$.

The question is, given a justification $\mathcal{J}$, what axioms should $\mathcal{J}^+$ contain? Although there is no definitive answer to this, it must be remembered that the ultimate goal is to include enough in $\mathcal{J}^+$ so that it is possible to produce a series of candidate lemmatised justifications, from which a "nice" one may be chosen using a complexity model. With this in mind, there are a number of possible options for $\mathcal{J}^+$ generation:

**Generation with Sub-Concepts** One possibility is to specify $\mathcal{J}^+$ so that it contains axioms of the forms specified in Table 1, which are build up from sub-concepts of axioms in $\mathcal{J}$. However, while such a strategy can go a long way to producing a set of axioms containing lemmas that could result in pleasing proofs, there could be axioms, which might be lemmas of choice, that are not be contained in the set. For example, given $\mathcal{O} = \{A \sqsubseteq \exists R.B, \exists R.B \sqsubseteq C, \mathsf{Trans}(R)\} \models A \sqsubseteq C$, a lemma of choice might be $\exists R.A \sqsubseteq \exists R.\exists R.B$ (entailed by $A \sqsubseteq \exists R.B$). However, with the above schema, based on sub-concepts, the class expression on the right hand side of the axiom ($\exists R.\exists R.B$) does not exists as a sub-concept in $\mathcal{J}$ and so the axiom would never be generated. What is needed is a set of class expressions that is rich enough so as to be able to build a rich set of axioms that constitute candidate lemmas. This is achieved using nested sub-concepts:

**Generation with Nested Sub-Concepts**

**Definition 5** **($\mathcal{J}^+$).** *For a justification $\mathcal{J}$ for $\eta$, let $\mathcal{S}$ be the set of sub-concepts occurring in the axioms in $\mathcal{J} \cup \{\eta\}$ plus $\top$ and $\bot$. Let $\mathcal{S}'$ be the smallest set of class expressions such that $\mathcal{S}' \supseteq \mathcal{S}$ and $\mathcal{S}'$ contains class expressions of the form:*

- *$\neg C$ where $C \in \mathcal{S}'$ and $C$ is not negated.*
- *$C_1 \sqcap \cdots \sqcap C_i$ or $C_1 \sqcup \cdots \sqcup C_i$ for $2 \leq i \leq |\mathcal{S}|$ and for any $C_j \in \{C_1, \ldots, C_i\}$ it is the case that $C_j \in \mathcal{S}$ or $C_j = \neg C$ for some $C \in \mathcal{S}$ where $C$ is not negated.*

*Now, let $d = |\mathcal{J}| \times c$ where $c$ is the maximum modal depth [1] of the class expressions in $\mathcal{S}$. Let $R$ be a property in the signature of $\mathcal{J}$ and $m$ be the sum of all numbers occurring in cardinality restrictions. Let $\mathcal{S}''$ be the smallest set of class expressions such that $\mathcal{S}'' \supseteq \mathcal{S}'$ and $\mathcal{S}''$ contains class expressions of the form:*

- *$\exists R.C$, $\forall R.C$, $\geq nR.C$ or $\leq nR.C$, where $C \in \mathcal{S}''$, the modal depth of $C$ is no greater than $d$, and $n \leq m$.*
- *$\exists R.\{a\}$, where $a$ and $R$ are in the signature of $\mathcal{J}$ or $\eta$.*
- *$\neg C$ where $C \in \mathcal{S}''$ and $C$ is not negated.*

*Given $\mathcal{S}''$, $\mathcal{J}^+$ is now defined as the set of axioms of the form given in Table 1, where $C$ and $D$ are substituted for class expressions in $\mathcal{S}''$, $R$ and $S$ are substituted for property expressions in $\mathcal{J}$, $a$ and $b$ are substituted for individuals* *in the signature of $\mathcal{J}$, and for each axiom $\alpha \in \mathcal{J}^+$, there exists a tidy subset $\mathcal{J}' \subseteq \mathcal{J}$ such that $\mathcal{J}' \models \alpha$.*

The ComputeJPlus algorithm in now defined to compute $\mathcal{J}^+$ in accordance with Definition 5. Since $\mathcal{S}$ is finite, $\mathcal{S}''$ is also finite and therefore $\mathcal{J}^+$ is also finite. Therefore, there are finite number of justifications for an entailment $\eta$ with respect to $\mathcal{J}^+$, hence GenerateProof algorithm is guaranteed to terminate.

## 6 The Feasibility of Computing Justification Oriented Proofs

The GenerateProof algorithm and its sub-routines, and the complexity model shown in Table 2 were implemented in Java using the OWL API. The algorithm has two basic, but necessary, optimisations. First, $\mathcal{J}^+$ is computed incrementally and the number of entailment checks is minimised in the obvious way, for example, if $\mathcal{J} \not\models A \sqsubseteq B$ then an entailment test is not performed for $A \sqsubseteq B \sqcap C$. Second, justifications in the LemmatiseJustification algorithm are computed one by one rather than all at once. This means that if a justification $\mathcal{J}'$ is found as a lemmatisation of $\mathcal{J}$ this justification is selected rather than continuing to look for one of lower complexity. If necessary, $\mathcal{J}'$ could be lemmatised to produce a justification of possibly lower complexity.

The implemented algorithm, with the Pellet reasoner, was tested against the ontologies listed in Table 3. For each ontology, a maximum of 5 justifications for entailments of the form $A \sqsubseteq B$, $A \sqsubseteq \bot$ and $A(a)$ were computed. Proofs were then computed for these justifications. Times for computing the justifications, and times for computing proofs were measured and averaged.

The implementation, although naive, with plenty of room for further optimisation, shows that it can be practical to compute proofs for entailments in real ontologies. Generally speaking, if it is possible to compute a justification for an entailment, it is possible to compute a justification oriented proof for that justification and entailment. In all cases, the time required to compute the proof is *at least* an order of magnitude higher than the time required to compute a justification. The difference is particularly striking for the Tambis ontology, where there were several justifications for which it took a significant time to perform entailment checking while computing $\mathcal{J}^+$ and then compute justifications over $\mathcal{J}^+$.

## 7 Examples

A selection of videos showing examples of justification oriented proofs may be found online at `http://www.cs.man.ac.uk/~horridgm/2010/iswc/proofs/examples/`. The examples illustrate the kinds of lemmas that get introduced into proofs and illustrate what is possible using the complexity model presented in Table 2. Figure 5 shows a justification oriented proof for the justification shown in 1. It should be noted that the presentation style used for the examples is

**Table 3.** Mean Times for Computing Justifications and Proofs

| **Ontology Expressivity/Axioms** | **Just. Size (Mean/SD/Max)** | **Just. Time (mean ms)** | **Proof Time (mean ms)** |
|---|---|---|---|
| Generations ($\mathcal{ALCOIF}$/38) | 4 / 2.1 / 8 | 31 | 2034 |
| Economy ($\mathcal{ALCH}$/1625) | 2 / 0.6 / 6 | 32 | 144 |
| People+Pets ($\mathcal{ALCHOIN}$/108) | 4 / 2.5 / 16 | 31 | 801 |
| Tambis ($\mathcal{SHIN}$/595) | 8 / 4.1 / 21 | 1047 | 244987 |
| Nautilus ($\mathcal{ALCF}$/38) | 3 / 2.0 / 6 | 20 | 758 |
| Transport ($\mathcal{ALCH}$/1157) | 5 / 2.1 / 9 | 19 | 469 |
| University ($\mathcal{SOIN}$/52) | 5 / 2.1 / 9 | 21 | 1738 |
| PeriodicTable ($\mathcal{ALU}$/100) | 4 / 9.9 / 36 | 72 | 1026 |
| Chemical ($\mathcal{ALCHF}$/114) | 8 / 1.2 / 11 | 38 | 3690 |

merely for illustrative purposes. In the tree presentation used, the children of an axiom represent a justification for that axiom.

## 8 Conclusions and Future Work

This paper has presented justification oriented proofs as possible solution to the problem of people understanding justifications. Justification lemmatisation has been introduced as a new non-standard reasoning service, which is a key component of for producing justification oriented proofs. Justification lemmatisation is based on the notion of a justification having a certain complexity for a given task. In the approach taken here, a simple complexity model based on various structural and non-structural phenomena was used as a basis for producing justification oriented proofs for entailments in real ontologies. Although, there is plenty of room for optimisation, some initial experiments on a series of published ontologies indicate that it is practical to compute justification oriented proofs for entailments in real ontologies.

Entailment : $\mathsf{Person} \sqsubseteq \bot$

- $\mathbf{Person} \sqsubseteq \neg\mathbf{Movie}$
- $\top \sqsubseteq \mathsf{Movie}$
  - $\forall\mathsf{hasViolenceLevel}.\bot \sqsubseteq \mathsf{Movie}$
    - $\forall\mathsf{hasViolenceLevel}.\bot \sqsubseteq \mathsf{RRated}$
      - $\mathbf{RRated} \equiv (\exists\mathbf{hasScript}.\mathbf{ThrillerScript}) \sqcup (\forall\mathbf{hasViolenceLevel}.\mathbf{High})$
    - $\mathsf{RRated} \sqsubseteq \mathsf{Movie}$
      - $\mathbf{RRated} \sqsubseteq \mathbf{CatMovie}$
      - $\mathbf{CatMovie} \sqsubseteq \mathbf{Movie}$
  - $\exists\mathsf{hasViolenceLevel}.\top \sqsubseteq \mathsf{Movie}$
    - $\mathbf{Domain}(\mathbf{hasViolenceLevel}, \mathbf{Movie})$

**Fig. 5.** A schematic of a justification oriented proof for the justification shown in Figure 1

It must be emphasised that the main contribution of this paper has been to formalise the notions of justification lemmatisation, justification oriented proofs and using complexity models to generate pleasing proofs. Preliminary user feedback, garnered from poster presentations at various conferences, has been very positive. However, as future work, a series of detailed user studies will be carried out to ascertain the specific benefit of justification oriented proofs to end users. Smooth presentation and interaction mechanisms will be designed to support this evaluation.

## References

1. Franz Baader, Diego Calvanese, Deborah L. McGuinness, D Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. 2003.
2. Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *KR-MED 08:*, 2008.
3. Alexander Borgida, Diego Calvanese, and Mariano Rodriguez. Explanation in DL-Lite. In *DL 2008*, 2008.
4. Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in OWL. In *ISWC 08*, 2008.
5. Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in OWL. In *DL 2009*, 2009.
6. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *KR 2006*, 2006.
7. Xiaorong Huang. Reconstructing proofs at the assertion level. In *CADE 94*, 1994.
8. Philip N. Johnson-Laird and Ruth M. J. Byrne. *Deduction*. Psychology Press, 1991.
9. Aditya Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, The Graduate School of the University of Maryland, 2006.
10. Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *ISWC 2007*, 2007.
11. Francis King Hei Kwong. Practical approach to explaining $\mathcal{ALC}$ subsumption. Technical report, The University of Manchester, 2005.
12. Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language structural specification and functional style syntax. W3C Recommendation, W3C – World Wide Web Consortium, October 2009.
13. Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In *CADE 21*, 2007.
14. Stephen E. Newstead, Peter Brandon, Simon J. Handley, Ian Dennis, and Jonathan St. B. Evans. *Predicting the Difficult of Complex Logical Reasoning Problems*, volume 12. Psychology Press, 2006.
15. David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 1986.
16. L. J. Rips. *The Psychology of Proof*. MIT Press, Cambridge, MA, 1994.
17. Stefan Schlobach. Explaining subsumption by optimal interpolation. In *JELIA 2004*, 2004.
18. Gerhard Strube. The role of cognitive science in knowledge engineering. In *Proc. of Contemporary Knowledge Engineering and Cognition*, 1992.