Signatures, Typed Feature Structures and RDFS

Matthias Denecke*

*Interactive Systems Laboratories
Carnegie Mellon University
Pittsburgh, PA, 15213
U.S.A.
denecke@cs.cmu.edu

Abstract

In this paper, we examine how attribute logic signatures and typed feature structures can be serialized using emerging semantic web standards RDF and RDFS. Inversely, we also consider to which degree the logic of typed feature structure is capable of representing and drawing inferences over RDF and RDFS documents.

1. Introduction

The representation of task related concepts is part of many natural language processing systems. Recently, there have been several projects involving several sites and project partners each of which develops a component that needs to receive data from other components, process it and resend the output to a subsequent component. Consequently, there is the issue to agree upon, to specify and to represent the concepts the system can understand. This can be seen as the vocabulary of the language in which components communicate with each other or in which knowledge sources are serialized. Note that this issue is orthogonal to the issue of deciding in which *format* the concepts should be exchanged. This can be seen as the syntax of the language in which components interact. For example, the fact that the exchange format has been decided to be XML does not describe the content of the exchanged messages.

For this reason, implemented systems often incorporate, or have access to, an ontology that represents the understood concepts and the relationships between them. This is particular prevalent in machine translation systems that employ an interlingua representation (see for example (Woszczyna et al., 2000)) and task oriented spoken dialogue systems.

In several systems, typed feature structures (Carpenter, 1992) have been shown to be popular representations. Originally developed with applications in grammar development in mind, the fact that the atoms of typed feature structures are partially ordered in a type hierarchy make this particular formalism appealing for task oriented natural language processing. In addition, efficient and simple implementations of subsumption and unification algorithms are available.

On the other hand, emerging W3C standards are concerned with semantic annotations of web resources. These efforts include the Resource Description Framework (RDFMS, 1999) and its Schema language RDFS (RDF Schema, 2000), and, building on top of RDFS, the *Darpa Agent Markup Language* (DAML for short) and and the *Ontology Inference Layer* (OIL for short). Both efforts recently joined forces and produced the DAML+OIL language.

In this paper, we look at how the logics of typed feature structures and the RDF and RDFS semantic web standards complement each other. The paper is organized as follows. In section 2, we give an overview of RDFS. In section 3, we give an overview of those aspects of typed feature structures that are relevant to our discussion. In section 4, we compare the two representations. In section 5, we preent mappings from signatures to RDFS documents and from typed feature structures to RDF documents, respectively. In section 6, we provide inverse mappings from particular RDFS documents to signatures and form particular RDF documents to typed feature structures, respectively. In section 7, we discuss the application of the techniques here presented to task oriented spoken language processing.

2. RDF und RDF Schema

RDF defines a simple data model for describing properties of and interrelationships between resources. It does not, however, provide any typing mechanism or content restriction. This functionality is provided by RDF Schema. A schema defines the kind of resources that are being described, and imposes semantic restrictions on the relationships between the described resources.

The RDF schema (RDFS for short) provides thus a basic type model to be used in RDF data models.

2.1. Classes

RDFS allows for the introduction of concepts by means of the `rdfs:class` tag. For example,

```
<rdfs:Class rdf:ID="motorvehicle"/>
```

declares a class `motorvehicle`. It is possible to create subclasses using the `rdfs:subClassOf` tag, as in

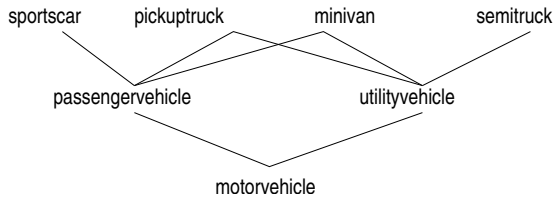

Figure 1: A class hierarchy.

```
<rdfs:Class rdf:ID="utilityvehicle">
   <rdfs:subClassOf
      rdf:resource="motorvehicle">
</rdfs:Class>
<rdfs:Class rdf:ID="passengervehicle">
   <rdfs:subClassOf
      rdf:resource="motorvehicle">
</rdfs:Class>
```

Multiple inheritance is equally possible:

```
<rdfs:Class rdf:ID="minivan">
   <rdfs:subClassOf
      rdf:resource="utilityvehicle">
   <rdfs:subClassOf
      rdf:resource="passengervehicle">
</rdfs:Class>
```

Instances of classes are RDF descriptions. Their class membership is indicated by means of the `rdf:type` tag.

```
<rdf:Description rdf:ID="MyCar">
   <rdf:type rdf:resource="#sportsvehicle">
</rdf:Description>
```

Note that a description may have multiple types as in

```
<rdf:Description rdf:ID="OldVan">
   <rdf:type rdf:resource="utilityvehicle">
   <rdf:type rdf:resource="passengervehicle">
</rdf:Description>
```

Figure 1 shows the class hierarchy used in the examples in this paper.

It should be noted that the relationship between classes induced by the `rdfs:subClassOf` tags may contain circular references (RDFMT, 2002). This is a recent change from the previous requirement disallowing circular references.

2.2. Properties

Properties are resources that express relationships between resources. The types of resources a property relates can be restricted with the `rdfs:domain` and `rdfs:range` tags. For example, the declaration

```
<rdf:Property rdf:ID="requiresLicense">
   <rdfs:domain rdf:resource="MotorVehicle">
   <rdfs:range rdf:resource="License">
</rdf:Property>
```

allows for the expression of relationships between motor vehicles and the licenses necessary to operate them.

The `rdfs:subPropertyOf` tag represents the relationship corresponding to the `rdf:subClassOf` relationship between classes. For example,

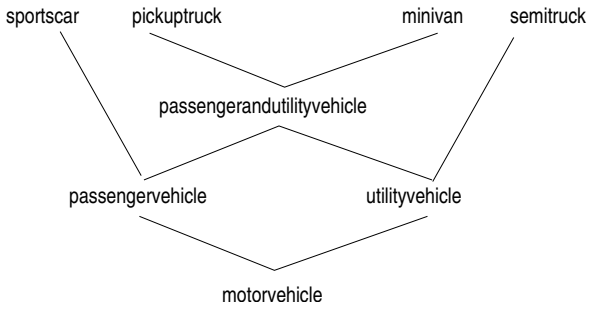

Figure 2: A type hierarchy.

```
<rdf:Property rdf:ID="requiresSpecialLicense">
   <rdfs:domain rdf:resource="SemiTruck">
   <rdfs:range rdf:resource="SpecialLicense">
</rdf:Property>
```

allows for the expression of relationships between semi trucks and the special kind of licenses required for them.

As a note, it is possible to have multiple domain and range constraints for the same property, in which case they are interpreted *conjunctively* (RDFMT, 2002), i.e. the properties need to set in relationship objects that are members of all classes declared in the domain and range constraints.

3. Signatures and Typed Feature Structures

Carpenter (1992) (LTFS henceforth) proposes to represent concepts in a so-called *type hierarchy* where the underlying partial order is a meet-semilattice, i.e. if any pair of types has upper bounds there is a unique upper bound.

3.1. Ordering of the Types

Type hierarchies $\langle \mathsf{Type}, \sqsubseteq \rangle$ are required to be meet semi lattices. This implies that if any set of types $S \subseteq \mathsf{Type}$ has an upper bound, this upper bound is uniquely determined. A consequence of this and the fact that unification of typed feature structures is defined in terms of upper bounds makes unification of typed feature structures unique. Throughout the paper, we will use the type hierarchy shown in figure 2 as an example.

3.2. Appropriateness Conditions

Appropriateness conditions restrict feature structures in constraining both the features that are appropriate for a given type as well as the values features may take.

Definition 3..1 (Appropriateness Specifications) *An* appropriateness specification *over the type hierarchy* $\langle \mathsf{Type}, \sqsubseteq \rangle$ *and a set of features* Feat *is a partial function* $Approp : \mathsf{Type} \times \mathsf{Feat} \rightarrow \mathsf{Type}$ *that meets the following conditions*

(i) **Feature Introduction**
for every feature $f \in \mathsf{Feat}$*, there is a most general type* $Intro(f) \in \mathsf{Type}$ *such that* $Approp(Intro(f), f)$ *is defined, and*

(ii) **Upward Closure/Right Monotonicity**
if $Approp(\sigma, f)$ *is defined and* $\sigma \sqsubseteq \tau$*, then* $Approp(\tau, f)$ *is also defined and such that* $Approp(\sigma, f) \sqsubseteq Approp(\tau, f)$

A type hierarchy $\langle \mathsf{Type}, \sqsubseteq \rangle$, a set of features Feat and an appropriateness condition $Approp$ over $\langle \mathsf{Type}, \sqsubseteq \rangle$ and Feat is called a *signature* (Penn, 2001).

An example for appropriateness conditions is given by

$$Approp(motorvehicle, requiredLicense) = license$$
$$Approp(semitruck, requiredLicense) = trucklicense$$

3.3. Typed Feature Structures

Typed feature structures are labelled rooted acyclic graphs in which each node is annotated with a type symbol $\tau \in \mathsf{Type}$ and each arc is annotated with a feature symbol $f \in \mathsf{Feat}$. Furthermore, for each node there may be at most one arc that is annotated with a given feature f. Figure 3 shows three typed feature structures.

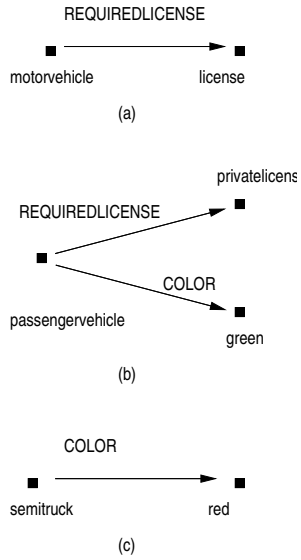

Figure 3: Three typed feature structures in graph notation.

More formally:

Definition 3..2 (Typed Feature Structure) *A feature structure $F = \langle Q, \bar{q}, \theta, \delta \rangle$ consists of a set of nodes Q with a distinguished root node $\bar{q} \in Q$, a total type function θ that assigns each node q its type $\theta(q)$ and a partial feature function δ defining the structure of the graph in that $\delta(q, f)$ gives the root node of the structure if feature f is applied to q.*

There is a structural algorithm that decides if one feature structure F' is at least as informative as F, or in other words, if F subsumes F'.

Definition 3..3 (Subsumption) *A feature structure F subsumes another feature structure F' iff there is a homomorphism h from the nodes in F to the node in F' such that*

1. *$h(\bar{q}) = \bar{q}'$ (the correspondent of the root node in F is the root node in F'),*
2. *if $\delta(q, f)$ is defined then $\delta'(h(q), f)$ is also defined and such that $h(\delta(q, f)) = \delta'(h(q), f)$ (h is a homomorphism w.r.t. the graph that underlies the feature structure),*
3. *if $h(q) = q'$ then $\theta(q) \sqsubseteq \theta(q')$ (if q' is the correspondent of some node q then the type of q' is at least as informative as the type of q.*

Informally, conditions (i) and (ii) ensure that the underlying graph of F is a subgraph of the underlying graph of F' while condition (iii) ensures that the type information in F' is at least as specific as the type information in F.

In figure 3, feature structure (a) subsumes feature structure (b) but not feature structure (c).

4. Comparison between RDF Schema and Type Hierarchies

4.1. Classes and Types

Both LTFS and RDFS provide representations of relations between concepts. Since the requirements on a type hierarchy are stricter than those on a class hierarchy in RDFS, type hierarchies can easily represented in RDFS but not any set of classes in RDFS is a type hierarchy. We will see below how in certain cases a set of classes can be converted in order to satisfy the conditions on type hierarchies. The contentious conditions include the fact that the `rdfs:subClassOf` relationship is not necessarily a meet semilattice; indeed due to circular references it might not even be a partial order.

4.2. Appropriateness Specifications and Properties

It is impossible in LTFS to express inheritance relationships between features. In other words, a signature exhibits a "flat" partial order between features where any two different features f_i and f_j are incomparable. On the other hand, there is no equivalent to the increasing specificity of the appropriateness constraints (see definition 3..1 (ii)) as domain and range constraints are interpreted conjunctively[1] (RDFMT, 2002). This means that the naive translation of the appropriateness specification 3.2. into

```
<rdf:Property rdf:ID="requiresLicense">
   <rdfs:domain rdf:resource="semitruck">
   <rdfs:range rdf:resource="speciallicense">
</rdf:Property>
<rdf:Property rdf:ID="requiresLicense">
   <rdfs:domain rdf:resource="motorvehicle">
   <rdfs:range rdf:resource="license">
</rdf:Property>
```

is equivalent to

```
<rdf:Property rdf:ID="requiresLicense">
   <rdfs:domain rdf:resource="semitruck">
   <rdfs:range rdf:resource="speciallicense">
</rdf:Property>
```

which, in turn, does not correspond to the appropriateness specification.

4.3. Instantiations of Classes

As noted above, RDF descriptons can be instances of two or more classes. There is no possibility to represent equivalent information in LTFS. The workaround described

[1] Note that in an earlier draft (RDF Schema, 2000), the interpretation of multiple range restrictions was disjunctively

in more detail below consists of adding additional types as needed if descriptions are instances of multiple classes.

Furthermore, due to the fact that instantiations of RDFS classes can have multiple types, together with the fact that the language does not contain negation, there are no inconsistent RDF descriptions. In LTFS, however, the process of unification assigns multiple types to nodes. If there is no upper bound for these types, the description is said to be inconsistent, and the unification fails.

5. Representing Signatures in RDFS

After having compared the differences and similarities between signatures and RDFS in the last section, we proceed to define a mapping m from signatures to RDFS document fragments and from typed feature structures to RDF document fragments.

5.1. Mapping Types

The mapping m of a type hierarchy $\langle \mathsf{Type}, \sqsubseteq \rangle$ to an RDFS class hierarchy is straightforward, as $\langle \mathsf{Type}, \sqsubseteq \rangle$ is a partial order. The distinguished type $\bot$ is mapped onto

```
<rdfs:Class rdf:ID="bot"/>
```

All other types t with direct super types $t_1, \ldots t_n$ are mapped onto

```
<rdfs:Class rdf:ID="t"/>
   <rdfs:subClassOf rdf:resource="#t1"/>
      ⋮        ⋮
   <rdfs:subClassOf rdf:resource="#tn"/>
</rdfs:Class>
```

5.2. Mapping Appropriateness Conditions

The mapping of appropriateness conditions takes advantage of the trick described in the previous section. In order to avoid redundancy in the specification, we consider the appropriateness function $Approp$ for those places where it changes.

In the following, an equation of the form $Approp(\tau, f) = \sigma$ for types σ, τ and a feature f is called an *appropriateness constraint*. An appropriateness constraint is called *redundant* if it can be inferred form another appropriateness constraint. For example, if $\tau \sqsubseteq \tau_1$ and $Approp(\tau, f) = \sigma$, then $Approp(\tau_1, f) = \sigma$ is redundant. A non-redundant appropriateness constraint is called *informative*. The *name* of an appropriateness constraint $Approp(\tau, f) = \sigma$ is f_τ_σ[2]. An appropriateness constraint $Approp(\tau, f) = \sigma$ subsumes another appropriateness constraint $Approp(\tau_1, g) = \sigma_1$ iff $f = g$ and $\tau \sqsubseteq \tau_1$ and $\sigma \sqsubseteq \sigma_1$.

As an example, we consider again the appropriateness specification 3.2.. The constraint $Approp(passengervehicle, requiredlicense) = license$ is redundant as it can be inferred from $Approp(motorvehicle, requiredlicense) = license$, together with the knowledge that $motorvehicle \sqsubseteq passengervehicle$. On the other hand, $Approp(semitruck, requiredlicense) = trucklicense$ is informative.

Given a signature, we construct property constraints for all informative appropriateness constraints induced by $Approp$. Let $Approp(\tau, f) = \sigma$ be an informative appropriateness constraint. In order to correctly construct RDFS properties, we need to distinguish between the "roots" in the constraint hierarchy (the constraints covered by the *feature introduction* condition in definition 3..1) and those that specialize other constraints (the constraints covered by the *Upward Closure* condition in definition 3..1). This can be determined by looking at τ: if $\tau = intro(f)$, then the first case holds, otherwise the second case holds.

Let first $\tau = intro(f)$. The generated RDFS fragment is then given by

```
<rdfs:Property rdf:ID="f_τ_σ">
   <rdfs:domain rdf:resource="τ"/>
   <rdfs:range rdf:resource="σ"/>
</rdfs:Property>
```

In the other case, the appropriateness constraint specializes one or more less specific appropriateness constraints $Approp(\tau_i, f) = \sigma_i$. In this case, we generate the following RDFS code.

```
<rdfs:Property rdf:ID="f_τ_σ">
   <rdfs:domain rdf:resource="τ"/>
   <rdfs:range rdf:resource="σ"/>
   <rdfs:subPropertyOf rdf:resource="#f_τ1_σ1">
      ⋮        ⋮
   <rdfs:subPropertyOf rdf:resource="#f_τn_σn">
</rdfs:Property>
```

The appropriateness conditions above generate the following document fragment.

```
<rdfs:Property
rdf:ID="reqLicense_motorvehicle_license">
   <rdfs:domain rdf:resource="#motorvehicle"/>
   <rdfs:range rdf:resource="#license"/>
</rdfs:Property>
<rdfs:Property
      rdf:ID="reqLicense_semitruck_trucklicense">
   <rdfs:domain rdf:resource="#semitruck"/>
   <rdfs:range rdf:resource="#trucklicense"/>
   <rdfs:subPropertyOf
rdf:resource="#reqLicense_motorvehicle_license">
</rdfs:Property>
```

5.3. Mapping Feature Structures

In mapping feature structures to RDF documents, we need to make sure to capture all information from the feature structures in RDF. We propose to create a new blank node for each node in the typed feature structure, and to link the nodes through properties. More specifically, if $F = \langle Q, \bar{q}, \theta, \delta \rangle$ then for each node $q \in Q$ with $t = \theta(q)$ we generate the following RDF fragment.

```
<rdf:Description rdf:ID="nodeID">
   <rdf:type rdf:resource="t"/>
</rdf:Description>
```

[2] For expository reasons, we assume that the character '_' does not appear in the names of the classes and properties. If it does, some more elaborate name mangling mechanisms would have to take place

Here $nodeID$ is a name uniquely generated for each node $q \in Q$.

In addition, if $\delta(q, f)$ is defined in q for any feature f, then we need to add that information to the description as well. This is done by mapping features to properties. The name of the property is chosen to be the most specific one that subsumes the the property given by $f, \theta(q)$ and $\theta(q')$. In this case, the description created for that node becomes

```
<rdf:Description rdf:ID="node − ID">
   <rdf:type rdf:resource="t"/>
   <featurename1 rdf:resource="#nodeID1"/>
      ⋮      ⋮
   <featurenamen rdf:resource="#nodeIDn"/>
</rdf:Description>
```

For example, the feature structure shown in figure 3 (b) shown here in attribute value matrix notation,

$$\begin{bmatrix} \textit{passengervehicle} & \\ \text{REQUIREDLICENSE} & \textit{privatelicense} \\ \text{COLOR} & \textit{green} \end{bmatrix}$$

is mapped to the document fragment

```
<rdf:Description rdf:ID="node_3">
   <rdf:type rdf:resource="passengervehicle"/>
   <requiredLicense_passvehicle_prlicense
      rdf:resource="#node_5"/>
   <color rdf:resource="#node_4"/>
</rdf:Description>
<rdf:Description rdf:ID="node_5">
   <rdf:type rdf:resource="privateLicense"/>
</rdf:Description>
<rdf:Description rdf:ID="node_4">
   <rdf:type rdf:resource="green"/>
</rdf:Description>
```

The graph representing the feature structure is shown in figure 4.

Note that the RDF graph constructed that way is a plain graph in RDFS parlance, meaning that it does not contain literals. On the other hand, since the nodes of feature structures are mapped to blank nodes in RDF descriptions, the graph is not ground.

5.4. Properties of the mapping

Before we discuss how subsumption is preserved through the mapping, we need to introduce the concept of *simple entailment* and RDFS *entailment*. In the process, we will refer to RDFS closure rules which are defined in (RDFMT, 2002). According to (RDFMT, 2002), one graph E simply entails another graph E' if every interpretation that satisfies every member of E also satisfies every member of E'. In this section, we will use the graph notation of document fragments. Furthermore, one graph RDFS entails another if the *closure* of the RDFS graph simple entails the closure of the other graph. The closure operation has the function of pulling in all the schema information. For example, the closure of the graph shown in figure 4 will add the nodes and the arc

$$node_0 \overset{rdf:type}{\rightarrow} motorvehicle$$

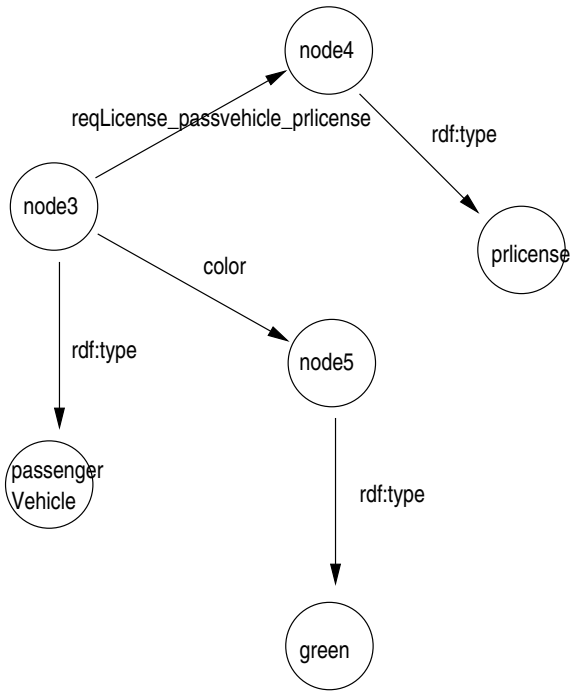

Figure 4: A feature structure in graph notation.

since the schema contains

$$passengervehicle \overset{rdfs:subClassOf}{\rightarrow} motorvehicle$$

and the graph representing the feature structure contains

$$node_0 \overset{rdf:type}{\rightarrow} passengervehicle$$

In addition, the closure will also generate the transitive closure of the `rdfs:subClassOf` and `rdfs:subPropertyOf` relationships.

The mapping from typed feature structures to RDF documents has the desirable property that $F \sqsubseteq F'$ implies that $m(F)$ is rdfs-entailed by $m(F')$. This can be seen as follows. $F \sqsubseteq F'$ implies the existence of the homomorphism h with the properties given in definition 3..3. Then, for each pair of nodes $q \in Q, q' \in Q'$ with $h(q) = q'$ we know by assumption that $\theta(q) \sqsubseteq \theta'(q')$. The generation of the RDF schema from the signature using the proces described above ensures that the graph contains a chain of subclass relationships

$$\begin{array}{lll} m(\theta'(q')) & rdfs:subClassOf & c_1 \\ \vdots & & \vdots \\ c_n & rdfs:subClassOf & m(\theta(q)) \end{array}$$

The closure rule RDFS 8 adds the statement

$$m(\theta(q)) \quad rdfs:subClassOf \quad m(\theta'(q')) \qquad (1)$$

The RDF document fragments representing F and F' contain the triples

$$m(q) \quad rdf:type \quad m(\theta(q)) \qquad (2)$$

and

$$m(q') \quad rdf:type \quad m(\theta'(q')) \qquad (3)$$

respectively. Closure rule RDFS 9, when applied to 1 and 3 will add the statement

$$m(q') \quad rdf:type \quad m(\theta(q)) \tag{4}$$

to the document fragment representing F'. This is due to the fact that the sublass relationship in (3) holds.

Similar closure rules exist for properties that inherit from superproperties. To summarize, the closure rules will add all weaker class membership and property membership statements to the graph. Since by assumption, the class and property membership statements of $m(F)$ are weaker than those of $m(F')$ the closure of $m(F')$ is a superset of the closure of $m(F)$ modulo node IDs. This is sufficient to prove simple entailment.

As an example, the feature structure shown in figure 3 (a) shown here in attribute value matrix notation,

$$\begin{bmatrix} \textit{motorvehicle} & \\ \text{REQUIREDLICENSE} & \textit{license} \end{bmatrix}$$

is mapped to the document fragment

$$\begin{array}{lll} node_0 & rdf:type & motorv \\ node_0 & reqLicense\_motorv\_license & node_1 \\ node_1 & rdf:type & license \end{array} \tag{5}$$

The document fragment generated from the feature structure (b), shown above, shown in graph notation:

$$\begin{array}{lll} node_3 & rdf:type & passengerv. \\ node_3 & reqLicense\_passv\_prlicense & node_4 \\ node_3 & color & node_5 \\ node_4 & rdf:type & prlicense \\ node_5 & rdf:type & green \end{array} \tag{6}$$

The closure rules RDFS 6 and RDFS 9 (see (RDFMT, 2002)) add the following triples to the graph, respectively:

$$\begin{array}{lll} node_3 & reqLicense\_motorv\_license & node_4 \\ node_3 & rdf:type & motorv. \\ node_4 & rdf:type & license \end{array} \tag{7}$$

If the triples in 6 and 7 are true, then the triples in 5 are also true, choosing the assignment where $node_0$ and $node_3$, as well as $node_1$ and $node_4$ are assigned the same resource.

6. Interpreting RDFS as Signatures

The previous section addressed the issue of generating an RDFS document that truthfully represents a signature. In this section, we consider the inverse problem, that is, to interpret an RDFS document as a signature. The requirements on signatures are stricter than those on RDF schemas. Consequently, and as also can be deduced from the discussion in the last section, there is no hope that all RDFS documents can be interpreted as signatures: in the last section, we restricted degrees of freedom in designing the property constraints to express appropriateness conditions; if, in the inverse conversion, the RDFS document does not meet these restrictions, there is no hope that it can be expressed as a signature. There are, however, some requirements on signatures that, if not satisfied in the RDF schema, can be mechanically restored. This includes the meet semilatticehood condition.

6.1. Mapping Classes

The difficulty in interpreting an RDFS class hierarchy as type hierarchy $\langle \mathsf{Type}, \sqsubseteq \rangle$ is due to the fact that the latter is required to be a meet semi lattice while the former is only guaranteed to be a partial order. This implies in particular, that if a set of classes has an upper bound, there need not be a uniquely determined least upper bound. The solution to this problem is to apply the meet semilattice completion algorithm (Penn, 2001) to the class hierarchy. This algorithm inserts additional dummy classes where needed in order to fulfil the meet semilatticehood condition. In addition, we also add a distinguished element $\bot$ to the resulting order, if there is no least element in the class hierarchy.

For example, the meet semilattice completion algorithm, if applied to the class hierarchy shown in figure 1 will generate the meet semilattice shown in figure 2.

It is legal in RDFS to define circular inheritance relationships, i.e. relationships of the form a_1 `rdfs:subClassOf` $a_2 \ldots$ a_n `rdfs:subClassOf` aa_1. The intended meaning is to state that the classes a_1 to a_n are equivalent. We can transform an RDF schema specification with circular inheritance relationships into one without circular inheritance relationships by factorization, whereby two classes a, b are equivalent $a \equiv b$ iff a `rdfs:subClassOf`* b and b `rdfs:subClassOf`* a. This needs to be done before the MSL algorithm is applied.

6.2. Mapping Properties

Multiple domain and range restrictions are dealt with as follows. Let $c_1, \ldots, c_n, n > 1$ be the domain (range) restrictions of any given property. If there is a class c that is a direct subclass of $c_1, \ldots, c_n, n > 1$ but that is not a direct subclass of any other class, then replace the multiple domain (range) restrictions with that class. If there is no such class add the declaration

```
<rdfs:Class rdf:ID="c">
   <rdfs:subClassOf rdf:resource="c1">
   ⋮      ⋮
   <rdfs:subClassOf rdf:resource="cn">
</rdfs:Class>
```

and replace any occurrence of `rdfs:subClassOf rdf:resource="`c_i`"` in other class definitions with `rdfs:subClassOf rdf:resource="`c`"`.

As mentioned above, signatures do not allow to represent feature hierarchies. However, since we can encode appropriateness conditions in property hierarchies, we should inversely be able to interpret property hierarchies as appropriateness conditions. This, of course, can only succeed if the property hierarchy in question satisfies the conditions of the appropriateness conditions.

7. Applications to Task-Oriented Natural Language Processing

The mappings from signatures and typed feature structures to RDFS and RDF document fragments, respectively, demonstrate that RDF(S) can be used as a convenient (and standardized) way to serialize signatures and typed feature

structures. Denecke and Waibel (1997), Johnston (1998) and Denecke and .Yang (2000) showed how signatures can be used as domain models for spoken dialogue systems, as the basis for generating efficient clarification questions and for fusion of multimodal input streams in multimodal dialogue systems. Denecke and Waibel (1999) demonstrate how the domain model of a spoken dialogue system can be assembled from several smaller ontologies. This technique is also supported by RDFS documents, their URIs serving at the same time as namespaces.

Inversely, as the limited mappings from RDF and RDFS documents to signatures and typed feature structures suggest, LTFS can be used to represent and reason with certain RDF(S) documents.

8. Conclusion

In this paper, we presented algorithms to convert signatures for typed feature structures and typed feature structures to RDFS and RDF document fragments, respectively. The resulting document fragments represent all the information encoded in the original signatures and feature structures. We also presented algorithms that can convert some RDFS and RDF document fragments to signatures and typed feature structures, respectively. Furthermore, we gave examples of RDFS and RDF document fragments that cannot be converted to signatures or typed feature structures without loss of information. In short, the result of the paper is that RDFS documents without the `rdfs:subPropertyOf` tag can be converted to signatures and RDF documents in which there are no two arcs with the same label leaving any node can be converted to a typed feature structure.

Applications of the algorithms include the exchange of ontologies and semantic representation between components in natural language processing applications.

Future work includes extension of the algorithms presented here to treat containers and integral datatypes correctly.

9. References

B. Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press.

M. Denecke and A.H. Waibel. 1997. Dialogue Strategies Guiding Users to their Communicative Goals. In *Proceedings of Eurospeech, Rhodos, Greece*. Available at http://www.is.cs.cmu.edu.

M. Denecke and A. H. Waibel. 1999. Integrating Knowledge Sources for a Task-Oriented Dialogue System. In *Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Stockholm, Sweden*. Available at http://www.is.cs.cmu.edu.

M. Denecke and J .Yang. 2000. Partial Information in Multimodal Dialogue Systems. In *Proceedings of the International Conference on Multimodal Interfaces*. Available at http://www.is.cs.cmu.edu.

M. Johnston. 1998. Unification-Based Multimodal Parsing. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 98), Montreal, Canada*. Association for Computational Linguistics Press. Available at http://www.cse.ogi.edu.

G. Penn. 2001. Tractability and Structural Closures in Attribute Logic Signatures. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001), Toulouse, France*.

RDF Schema. 2000. Resource description framework schema specification. Technical report.

RDFMS. 1999. Resource description framework model and syntax specification.

RDFMT. 2002. RDF model theory. Technical report, W3C.

M. Woszczyna, M. Broadhead, D. Gates, M. Gavalda, A. Lavie, L. Levin, and A. Waibel. 2000. Evaluation of a practical interlingua for task-dependent dialogue. In *Proceedings of the AMTA SIG-IL Third Workshop on Interlinguas and Interlingua Approaches, Seattle, Washington*. Available at http://www.is.cs.cmu.edu.