S-MODULES – AN APPROACH TO CAPTURE SEMANTICS OF MODULARIZED DL KNOWLEDGE BASES

Krzysztof Goczyła, Aleksander Waloszek and Wojciech Waloszek
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology
Narutowicza Str. 11/12, Gdańsk, Poland

Keywords: Description Logics and Ontologies, Knowledge Engineering, Knowledge-based Systems.

Abstract: Modularity of ontologies has been recently recognized as a key requirement for collaborative ontology engineering and distributed ontology reuse. Partitioning of an ontology into modules naturally gives rise to development of module processing methods. In this paper we describe an algebra of ontology modules developed during our work on a Knowledge Base Management System called RKaSeA. The idea differs from other algebras in the fact that we treat a module semantically, i.e. we focus on the set of a module's models rather that on the set of axioms and assertions included in its representation The algebra has revealed its potential also in the process of reasoning.

1 INTRODUCTION

Subsequent years bring more and more attention into the area of modularization of Description Logics (DL) knowledge bases. The main motivation is the hope to reach the maturity of the collaborative ontology development and reuse process comparable to the one achieved by software engineering methods in the case of software modules. In consequence, many methods and techniques of ontology decomposition and merging have been devised.

During the course of work on a Description Logic-based Knowledge Base Management System (KBMS) called RKaSeA we faced the issue of choosing the proper formalism for modularization. Naturally we strived for allowing the user to create and process a modularized knowledge base, but we wanted also to enable her to extract a fragment of a selected module, combine it with a fragment from other module, etc. Moreover, our aim was also to explore possibilities of automated (i.e. unseen by the user) ontology decomposition in order to obtain shorter time of executing reasoning tasks.

As a result we developed a formalism for describing operations on modules. The formalism is simple, yet robust enough to describe various complicated operations useful for expressing module decomposition and merge. Because we focused strongly on the semantic aspect of modules, we called the formalism *s-module algebra* (pronounced *semodules*, as '*s'* stands here for "semantics").

In the following we introduce basic notions of s-module algebra and present examples of its use. Then we discuss issues connected with decidability and compare our work with related published methods. A short discussion of further directions of development of the algebra concludes the paper.

2 PRELIMINARIES

In the following we assume that we use an arbitrary chosen description logic $\mathcal{L}$.

Let a *signature*, denoted $\mathbf{S} = \mathbf{C} \uplus \mathbf{R} \uplus \mathbf{I}$ for any $\mathcal{L}$, is a disjoint union of *concept names* ($\mathbf{C}$), *role names* ($\mathbf{R}$), and *individual names* ($\mathbf{I}$). $\mathbf{C}(\mathbf{S})$, $\mathbf{R}(\mathbf{S})$, and $\mathbf{I}(\mathbf{S})$, respectively, refer to corresponding part of $\mathbf{S}$. We assume that there exist a set of acceptable names, denoted as $\mathcal{N}$ ($\mathbf{C}, \mathbf{R}, \mathbf{I} \subseteq \mathcal{N}$).

The chosen description logics $\mathcal{L}$ determines the set of (possibly complex) concepts, roles and individuals one can build using the operators in $\mathcal{L}$ and names from a signature $\mathbf{S}$. We denote these sets with $\mathcal{L}_C(\mathbf{S})$, $\mathcal{L}_R(\mathbf{S})$, $\mathcal{L}_I(\mathbf{S})$, respectively. For instance, if $\mathcal{L}$ is $\mathcal{ALC}$, then $\mathcal{L}_C(\mathbf{S}) ::= A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C \mid \top \mid \bot$, where $A \in \mathbf{C}(\mathbf{S})$, $C, D \in \mathcal{L}_C(\mathbf{S})$, $R \in \mathcal{L}_R(\mathbf{S})$.

An $\mathbf{S}$-*interpretation* $\mathcal{I}$ is a pair $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a set called the *domain* of the interpretation and $\cdot^\mathcal{I}$

is an *interpretation function* assigning each $A \in \mathbf{C}(\mathbf{S})$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each $R \in \mathbf{R}(\mathbf{S})$ a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each $a \in \mathbf{I}(\mathbf{S})$ an element $a^{\mathcal{I}}$ of the domain $\Delta^{\mathcal{I}}$. Moreover each description logic $\mathcal{L}$ establishes its own rules for extending the interpretation to complex concepts, e.g. in $\mathcal{ALC}$ $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$.

Each description logic allows for formulating axioms and assertions (sentences), and define the conditions under which a specific interpretation $\mathcal{I}$ satisfies a particular sentence. We denote the set of all axioms and assertions that can be built from a given signature $\mathbf{S}$ as $\mathcal{L}(\mathbf{S})$. The fact that the interpretation $\mathcal{I}$ satisfies a given sentence $\alpha \in \mathcal{L}(\mathbf{S})$ (is its *model*) is denoted as $\mathcal{I} \vDash \alpha$.

3 S-MODULES ALGEBRA

3.1 Basic Definitions

An assumption we take is that a user is interested in conclusions that can be drawn from an ontology rather than in particular axioms and assertions. So, we define an ontology module strictly semantically, focusing only on its interpretations.

Definition 1 (S-module). An *s-module* $M = (\mathbf{S}, \mathbf{W})$ is a pair of a signature $\mathbf{S}$ (called a *signature* of the s-module) and a class $\mathbf{W}$ of $\mathbf{S}$-interpretations (called *models* of the s-module). The two parts of M are denoted as $\mathbf{S}(M)$ and $\mathbf{W}(M)$, respectively. Each $\mathbf{S}$-interpretation from $\mathbf{W}$ we call a *model* of M. ◆

According to this definition, each module simply consists of all its models. We say that s-module satisfies a particular sentence α, denoted $M \vDash \alpha$, iff $\forall \mathcal{I} \in \mathbf{W}(M)$: $\mathcal{I} \vDash \alpha$.

We think that creators of an ontology is not able to foresee all its future uses. By necessity, the creators have to focus on a small set of chosen contexts of use of particular modules, and the contexts of their choice may not be adequate for a particular application of a knowledge base with this ontology. So, the s-modules algebra puts stress on various methods of manipulation for s-modules; that in general allow for changing and combining signature and models.

3.2 Operators

In this section we introduce operators of s-module algebra. Some of the operators are non-primitive and can be derived from others.

We assume that description logic $\mathcal{L}$ and domain set $\boldsymbol{\Delta}$ are chosen and fixed (both assumptions can be alleviated, see Section 3.3). We denote a set of all modules as $\boldsymbol{M}$, a set of all signatures as $\boldsymbol{\Sigma}$, and a set of all $\boldsymbol{S}$-interpretations as $\boldsymbol{I}(\boldsymbol{S})$. We also use the notion of a *projection* $\mathcal{I}' = \mathcal{I}|\mathbf{S}'$ of an $\mathbf{S}$-interpretation $\mathcal{I}$ to a signature $\mathbf{S}'$ ($\mathbf{S}' \subseteq \mathbf{S}$). $\mathcal{I}'$ is an $\mathbf{S}'$-interpretation for which the following holds: $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}}$ and $X^{\mathcal{I}'} = X^{\mathcal{I}}$ for every $X \in \mathbf{S}'$.

In the subsequent descriptions: for unary operators, their operand is denoted as M, for binary operators the operands are denoted as M_1 and M_2. M' always represents a result.

Extend

$\varepsilon_{\mathbf{S}}$: $\boldsymbol{M} \to \boldsymbol{M}$, $\mathbf{S} \in \boldsymbol{\Sigma}$;

$\varepsilon_{\mathbf{S}}(M) = (\mathbf{S}(M) \cup \mathbf{S},\ \{\mathcal{I} \in \boldsymbol{I}(\mathbf{S}(M) \cup \mathbf{S}): \mathcal{I}|\mathbf{S}(M) \in \mathbf{W}(M)\})$. ◆

Extension extends a signature of a given module M by names from a given signature $\mathbf{S}$. The allowed set of interpretations of each original name is preserved, and so are the relationships between original concepts, roles, and individuals (e.g. if $M \vDash \alpha$, $\alpha \in \mathcal{L}(\mathbf{S}(M))$, then also $\varepsilon_{\mathbf{S}}(M) \vDash \alpha$).

Project

$\pi_{\mathbf{S}}$: $\boldsymbol{M} \to \boldsymbol{M}$, $\mathbf{S} \in \boldsymbol{\Sigma}$, $\mathbf{S} \subseteq \mathbf{S}(M)$;

$\pi_{\mathbf{S}}(M) = (\mathbf{S}, \{\mathcal{I}|\mathbf{S}: \mathcal{I} \in \mathbf{W}(M)\})$. ◆

Projection reduces a signature of a given module. However, relationships between original concepts, roles, and individuals whose names remain in the signature are preserved (e.g. if $M \vDash \alpha$, $\alpha \in \mathcal{L}(\mathbf{S})$, then also $\pi_{\mathbf{S}}(M) \vDash \alpha$).

Rename

ρ_{γ}: $\boldsymbol{M} \to \boldsymbol{M}$, $\boldsymbol{\gamma}$ is a signature mapping;

$\rho_{\gamma}(M) = (\gamma(\mathbf{S}), \gamma(\mathbf{W}))$. ◆

Renaming uses the notion of a *signature mapping*. Signature mapping γ is a triple: $(\gamma_C, \gamma_R, \gamma_I)$, each of them being a bijection from $\mathcal{N}$ to $\mathcal{N}$. By $\gamma(\mathbf{S})$ we mean $\gamma_C(\mathbf{C}(\mathbf{S})) \uplus \gamma_R(\mathbf{R}(\mathbf{S})) \uplus \gamma_I(\mathbf{I}(\mathbf{S}))$, and by $\gamma(\mathcal{I})$, where $\mathcal{I}$ is an $\mathbf{S}$-interpretation, we mean an $\gamma(\mathbf{S})$-interpretation $\mathcal{I}'$ such that $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}}$ and $\gamma(X)^{\mathcal{I}'} = X^{\mathcal{I}}$ for every $X \in \mathbf{S}$. Rename preserves relationships between concepts, roles, and individuals, however with respect to their name changes (e.g. if $M \vDash \alpha$, $\alpha \in \mathcal{L}(\mathbf{S}(M))$, then $\rho_{\gamma}(M) \vDash \gamma(\alpha)$, where $\gamma(\alpha)$ is α transformed in such a way that all names in α have been systematically changed according to γ).

Select

σ_{α}: $\boldsymbol{M} \to \boldsymbol{M}$, $\alpha \in \mathcal{L}(\mathbf{S}(M))$;

$\sigma_{\alpha}(M) = (\mathbf{S}, \{\mathcal{I} \in \mathbf{W}(M): \mathcal{I} \vDash \alpha\})$. ◆

Selection leaves only these interpretations that are models of a sentence α. Obviously $\sigma_\alpha(M) \vDash \alpha$.

Union

$\cup$: $\boldsymbol{M} \times \boldsymbol{M} \to \boldsymbol{M}$, $\mathbf{S}(M_1) = \mathbf{S}(M_2)$;
$M_1 \cup M_2 = (\mathbf{S}(M_1), \mathbf{W}(M_1) \cup \mathbf{W}(M_2))$. ◆

Union performs a set-theoretic union of sets of models of s-modules. The condition that $\mathbf{S}(M_1) = \mathbf{S}(M_2)$ is not very restrictive because we can easily upgrade this operation to a *generalized* union $\cup_g$: $M_1 \cup_g M_2 = \varepsilon_{\mathbf{S}(M_2)}(M_1) \cup \varepsilon_{\mathbf{S}(M_1)}(M_2)$.

Intersection

$\cap$: $\boldsymbol{M} \times \boldsymbol{M} \to \boldsymbol{M}$, $\mathbf{S}(M_1) = \mathbf{S}(M_2)$;
$M_1 \cap M_2 = (\mathbf{S}(M_1), \mathbf{W}(M_1) \cap \mathbf{W}(M_2))$. ◆

Difference

$-$: $\boldsymbol{M} \times \boldsymbol{M} \to \boldsymbol{M}$, $\mathbf{S}(M_1) = \mathbf{S}(M_2)$;
$M_1 - M_2 = (\mathbf{S}(M_1), \mathbf{W}(M_1) - \mathbf{W}(M_2))$. ◆

Intersection and difference are analogous to the union, and can be generalized in the similar way to the case when signatures of the operands differ.

Union and difference are non-linguistic (strictly semantic), i.e. their use may lead to generation of a s-module M for which there does not exist any corresponding set of sentences S in $\mathcal{L}$. This issue will be elaborated on later in the paper.

I-Join (Intersecting Join)

$\times$: $\boldsymbol{M} \times \boldsymbol{M} \to \boldsymbol{M}$;
$M_1 \times M_2 = (\mathbf{S}', \mathbf{W}')$;
$\mathbf{S}' = \gamma_1(\mathbf{S}(M_1)) \cup \gamma_2(\mathbf{S}(M_2))$;
$\mathbf{W}' = \{\mathcal{I} \in \mathbb{I}(\mathbf{S}')\colon\ \mathcal{I}|\gamma_1(\mathbf{S}(M_1)) \in \gamma_1(\mathbf{W}(M_1)) \wedge \mathcal{I}|\gamma_2(\mathbf{S}(M_2)) \in \gamma_2(\mathbf{W}(M_2))\}$. ◆

I-Join is an operation on two s-modules. It uses two signature mappings γ_1, γ_2, each of them preceding every terminological name in a signature with a unique prefix. Thus, I-Join helps to solve potential naming conflict between s-modules. I-Join preserves relationships between original concepts and roles in both modules (if $M_1 \vDash \alpha$, $\alpha \in \mathcal{L}(\mathbf{S}(M_1))$, then $M' \vDash \gamma_1(\alpha)$, analogically for M_2).

I-Join is a non-primitive operation, as $M_1 \times M_2$ can be expressed as $\rho_{\gamma_1}(M_1) \cap_g \rho_{\gamma_2}(M_2)$. This derivation justifies the name "intersecting join" as we may perceive this join as a "safe" way of intersecting modules.

U-Join (union join)

$\bowtie$: $\boldsymbol{M} \times \boldsymbol{M} \to \boldsymbol{M}$;
$M_1 \bowtie M_2 = (\mathbf{S}', \mathbf{W}')$;
$\mathbf{S}' = \gamma_1(\mathbf{S}(M_1)) \cup \gamma_2(\mathbf{S}(M_2))$;
$\mathbf{W}' = \{\mathcal{I} \in \mathbb{I}(\mathbf{S}')\colon\ \mathcal{I}|\gamma_1(\mathbf{S}(M_1)) \in \gamma_1(\mathbf{W}(M_1)) \vee \mathcal{I}|\gamma_2(\mathbf{S}(M_2)) \in \gamma_2(\mathbf{W}(M_2))\}$. ◆

U-Join is a counterpart of I-Join. γ_1, γ_2 have the same meaning as above. U-Join offers the "safe" way of performing union. Naturally, $M_1 \bowtie M_2 = \rho_{\gamma_1}(M_1) \cup_g \rho_{\gamma_2}(M_2)$.

Put-Under

υ_C: $\boldsymbol{M} \times \boldsymbol{M} \to \boldsymbol{M}$, $C \in \mathcal{L}_C(\mathbf{S}(M_2))$;
$M_1\ \upsilon_C\ M_2 = (\mathbf{S}', \mathbf{W}')$;
$\mathbf{S}' = \mathbf{S}(M_1) \cup \mathbf{S}(M_2)$;
$\mathbf{W}' = \{\mathcal{I} \in \mathbb{I}(\mathbf{S}')\colon\ \mathcal{I}|\mathbf{S}(M_2) \in \mathbf{W}(M_2) \wedge (\mathcal{I}|\mathbf{S}(M_1) \cap C^{\mathcal{I}}) \in \mathbf{W}(M_1)\}$. ◆

Put-Under correlates the domains of two s-modules. We use here a *restriction* of an **S**-interpretation $\mathcal{I}$: by $\mathcal{I} \cap \Delta'$ we mean an interpretation $\mathcal{I}' = (\Delta', \cdot^{\mathcal{I}'})$ such that $X^{\mathcal{I}'} = X^{\mathcal{I}} \cap \Delta'$ for every $X \in \mathbf{S}$. As each s-module induces a set of laws that enforce certain relationships between concepts, roles, and individuals, then Put-Under can be perceived as a restriction of the scope of these laws to a fragment of a larger domain.

3.3 Remarks on S-module Algebra

First we distinguish between primitive and non-primitive operators. Such a partition is to some extent arbitrary (e.g. we could treat I-Join rather than Extend a primitive operation). As far as Rename is concerned: if we need to change elements of a signature **S** of a s-module M into elements of a new signature **S′**, we might think of the following scenario. Instead of using ρ operation, we extend the signature of M by **S′** (with use of ε). Then we assure the extensional equality between terms from **S** and **S′** by formulating appropriate axioms (like $A \equiv B$, we assume that $A \in \mathbf{S}$ and $B \in \mathbf{S}'$). Finally, we remove the unwanted terms by using π. However, this may not always be possible as not every description logic allows for formulating such axioms (e.g. in $\mathcal{ALC}$ we cannot express equality of roles). This is why we consider Rename primitive.

Earlier we mentioned possibility of alleviation of the two assumptions: that a single description logic $\mathcal{L}$ is used in all the s-modules, and that a particular domain set $\boldsymbol{\Delta}$ is chosen and fixed. The latter is simple to be lifted, the consequence is that sets of interpretations in s-modules become classes. The former assumption relates to the issue of multi-Description Logics modules and its alleviation is a more complicated. In general, the exploited description Logic $\mathcal{L}$ is not used explicitly in the definition of a s-module. Yet, it is indirectly contained in the set of interpretations, because each interpretation of a s-module follows the rules of expanding base-term interpretation into the interpretation of complex roles and concepts. Therefore, in order to translate a s-module M_1 "expressed" in $\mathcal{L}_1$ into a s-module "expressed" in $\mathcal{L}_2$ we just need to "contract" each interpretation $\mathcal{I} \in \mathbf{W}(M_1)$ to its base interpretation and then "expand" it in according to the rules of expansion appropriate for the logic $\mathcal{L}_2$. This, however, may lead to over-expressiveness of some s-modules.

Example 1 (Over-expressiveness). Let us take any s-module M and assume that it contains $\mathcal{SHIQ}$ interpretations. We extend the signature of the module by a new role R and select only these interpretations that satisfy the axiom $\mathsf{Trans}(R)$, i.e. we build $M' = \sigma_{\mathsf{Trans}(R)}(\varepsilon_{\{R\}}(M))$. Then we change the logic of M' to $\mathcal{ALC}$. Interpretations contained in $\mathbf{W}(M')$ no longer map concepts like $\geq 3R.\top$, but the role R remains transitive in all interpretations in $\mathbf{W}(M')$, although there is no possibility of expressing this fact in $\mathcal{ALC}$. ◆

The effect described in the above example seems to reveal a crucial obstacle in using multi-logic modules. However, as it is shown below over-expressiveness is not specific only to multi-logic.

To elaborate on this we introduce a notion of a *linguistic s-module*. To ease the task we return to the assumptions of chosen and fixed $\mathcal{L}$ and $\mathbf{\Delta}$.

Definition 2 (Linguistic S-module). We call a s-module $M = (\mathbf{S}, \mathbf{W})$ *linguistic* iff there exists a set of sentences $S \subseteq \mathcal{L}(\mathbf{S})$ such that $\mathbf{W} = \{\mathcal{I} \in \mathcal{I}(\mathbf{S}): \forall\alpha \in S: \mathcal{I} \vDash \alpha\}$. ◆

In other words, linguistic s-modules are modules whose models are all models of a particular set of sentences. We denote a linguistic s-module as $M(\mathbf{S}, S)$, where S is the set of sentences, or simply as $M(S)$ if module signature can be inferred from the context, e.g. $M(\{A \sqsubseteq B, A(a)\})$ with signature $\mathbf{S} = \{A, B, a\}$.

In general, when using specific operations we have no guarantee that the outcome is linguistic even if all operands are. These operations are called *non-linguistic* and they are: Union (∪), Difference (–), Projection (π), and U-Join (⋈). They may result in obtaining a s-module M which does not correspond directly to any set of sentences from $\mathcal{L}(\mathbf{S}(M))$.

Example 2 (Non-linguistic S-module). Consider a union M' of two linguistic s-modules $M' = M(\{A \sqsubseteq B\}) \cup M(\{B \sqsubseteq A\})$. Neither $M' \vDash A \sqsubseteq B$ nor $M' \vDash A \sqsubseteq B$ is true. But after an intersection: $M'' = M' \cap_g M(\{A \sqcap \neg B(a), \neg A \sqcap B(b)\})$, we obtain a s-module with no models: $\mathbf{W}(M'') = \emptyset$. ◆

4 EXAMPLES

There is similarity between the s-module algebra and the relational algebra. Both algebras aim at delivering operators on defined units of data (or knowledge) that allow a user to combine the units in order to obtain a new unit, presumably better suited for a specific purpose. Like the relational algebra, the s-module algebra may be a base for development of a language for the end-user, and such a language would have a desired property of closeness.

The following example shows how to merge information from two s-modules in order to obtain desired piece of knowledge.

Example 3 (Intersecting S-modules). Consider two s-modules: M_1 describes human resources and M_2 the structure of a hospital.

$M_1 = M(\{\exists isManagerIn.HTBusinessUnit \sqsubseteq Expert,$
$\quad Expert \sqsubseteq Employee\})$
$M_2 = M(\{leadsDepartment(johnSmith, neurosurgery), Department(neurosurgery)\})$.

To merge the information from the two s-modules in order to infer that *johnSmith* is an expert, we first create an intersection of the s-modules: $M' = M_1 \cap_g M_2$, and then restrict the set of models by introducing additional "bridge" axioms: $M'' = M' \cap_g M(\{leadsDepartment \sqsubseteq isManagerIn, Department \sqsubseteq HTBusinessUnit\})$. The last step can also be done by double selection. ◆

In the example we did not encounter any name conflict between s-modules being merged. In general, such a conflict may occur and I-Join operator should be used. Below we show how to align two s-modules in which the same set of terms is used to express different meanings.

Example 4 (Joining S-modules). Consider two s-modules: M_1 and M_2. They contain assessment of several rooms for rent, and use the same categorization and signature $\mathbf{S} = \{HSRoom, ASRoom, LSRoom\}$, where the concepts denote high, average and low standard rooms. But in M_1 and M_2 different criteria were used for categorization, as in the first case we were looking for a room to spend just one day and in the second case to stay for a longer period of time. We "import" the assessment from M_1 to M_2 performing necessary translation of classification between the s-modules.

1. In the first step we simply I-Join the modules. As a result we obtain a module $M' = M_1 \times M_2$. The concepts have been renamed, so $\mathbf{S}(M) = \{1{:}HSRoom, 2{:}HSRoom, 1{:}ASRoom, \ldots\}$.

2. Next, we make the criteria of assessment explicit. In this example we use only one criterion: a

bathroom. So, we extend the signature of M' appropriately: $M'' = \varepsilon_{\{RoomWithBathroom\}}(M')$.

3. Afterwards, we bind the criteria with the assessment. In M_1 rooms with bathrooms were automatically considered high standard. According to the criteria used in M_2 no room with bathroom can be considered more than low standard.

$M''' = M'' \cap_g M(\{RoomWithBathroom \sqsubseteq 1{:}HSRoom,$
$\neg RoomWithBathroom \sqsubseteq 2{:}LSRoom\})$.

Naturally, the second axiom is valid only if the domain consists of only rooms (which is assumed).

4. Finally we remove unwanted terms from the module signature: $M = \pi_{\{2:HSRoom,\ 2:ASRoom,\ 2:LSRoom\}}(M''')$.

In these steps all the translations possible to perform were done. All average or low standard rooms from M_1 were considered low standard in accordance with criteria from M_2. ◆

We do not claim that the s-module algebra is to replace the existing methods of modularization, like DDL, ε-connections (see e.g. d'Aquin *et al.*, 2008) for a brief survey), etc. Each of the methods has its own goals and offers apparatus for solving certain problems (e.g. locality is in fact beyond the scope of the current version of the s-module algebra). Instead we argue that the s-module algebra is a tool that might be useful in describing some common aspects of wide range of actions involving merging and transformation of knowledge.

The s-module algebra cannot capture some mappings proposed by other methods. For instance, it is impossible to map role instances into concept instances, as proposed in (Ghidini *et al.*, 2007) for a version of DDL. The reason is that there is no DL sentence for such a relationship. However, if we extend the language with Horn rules (see e.g. Motik *et al.*, 2005), we can obtain somewhat similar result with use of a rule like $C(a) \leftarrow R(a, b)$.

Example 5 (Union). Consider the s-module M:

$M = M(\{\top \sqsubseteq \leq 1 murdered.\{victim\},$
$\exists accuses^{-}.TrustedWitness \sqsubseteq \exists murdered^{-}.\{victim\},$
$TrustedWitness \sqsubseteq \exists presentAt.CrimeScene\})$.

The s-module contains knowledge that there is only one murderer of a victim and it is the one who is accused by a trusted witness (who was present at the crime scene). Let consider two (mutually exclusive) versions of facts:

$M_1 = M \cap M(\{TrustedWitness(johnShady),$
$accuses(johnShady, tedInnocent)\}$

$M_2 = M \cap M(\{TrustedWitness(henryBrillant),$
$accuses(henryBrillant, markGuilty)\}$.

We can hold the two version in a single s-module $M' = M_1 \cup M_2$. No unambiguous conclusion about the murder can be drawn from M'. Fortunately, the further information collected (John Shady actually was not present at the crime scene) allows to rule out one of the versions: $M'' = M' \cap M(\{\neg\exists presentAt.CrimeScene(johnShady)\})$; $M'' \models murdered(markGuilty, victim))$. ◆

5 DECIDABILITY ISSUES

Since the use of the s-modules algebra may result in exceeding the original expressiveness of a chosen description logic, it is necessary to investigate whether basic reasoning problems remain decidable.

In our analysis we focus on the concept satisfiability problem (*csat*), which for s-modules can be formulated as follows: Given a module M and a concept $C \in \mathcal{L}_C(\mathbf{S}(M))$; check if there exists an interpretation $\mathcal{I} \in \mathbf{W}(M)$ for which $C^{\mathcal{I}} \neq \emptyset$.

Factors that influence decidability are: the used description logic $\mathcal{L}$ (for multi-logic modules we assume the most expressive logic) and the set of allowed algebra operators. Each problem instance can be then described as a triple (p, $\mathcal{L}$, A), where p is the problem name, $\mathcal{L}$ is the logic used, and A is the set of primitive operators (**A** denotes all operators: $\{\varepsilon,\pi,\rho,\sigma,\cup,\cap,-,\upsilon\}$). We also make an assumption that the only s-module constants we use correspond to linguistic s-modules. Preliminary results are described by the following theorems.

Theorem 8. Problem (*csat*, $\mathcal{SHOIQ}$, **A**–{–}) is decidable.

Sketch of proof. We associate each s-module with one or more sets of sentences (denoted $M \rhd \{S_1, S_2, \ldots\}$). Linguistic modules of the form $M(S)$ are associated with S ($M \rhd \{S\}$). Each algebraic operation produces a new module associated with newly constructed sets: $\varepsilon(M \rhd \{S_1, S_2 \ldots\}) = M' \rhd \{S_1, S_2, \ldots\}$, $\pi(M \rhd \{S_1, S_2, \ldots\}) = M' \rhd \{S_1, S_2, \ldots\}$, $\rho_\gamma(M \rhd \{S_1, S_2, \ldots\}) = M' \rhd \{\gamma(S_1), \gamma(S_2), \ldots\}$, $M_1 \rhd \{S_{1\text{-}1}, S_{1\text{-}2}, \ldots\} \cup M_2 \rhd \{S_{2\text{-}1}, S_{2\text{-}2}, \ldots\} = M' \rhd \{S_{1\text{-}1}, S_{1\text{-}2}, \ldots, S_{2\text{-}1}, S_{2\text{-}2}, \ldots\}$, $M_1 \rhd \{S_{1\text{-}1}, S_{1\text{-}2}, \ldots\} \cap M_2 \rhd \{S_{2\text{-}1}, S_{2\text{-}2}, \ldots\} = M' \rhd \{S_{1\text{-}1} \cup S_{2\text{-}1}, S_{1\text{-}1} \cup S_{2\text{-}2}, \ldots, S_{1\text{-}2} \cup S_{2\text{-}1}, S_{1\text{-}2} \cup S_{2\text{-}2}, \ldots\}$, $M_1 \rhd \{S_{1\text{-}1}, S_{1\text{-}2}, \ldots\}\ \upsilon_C\ M_2 \rhd \{S_{2\text{-}1}, S_{2\text{-}2}, \ldots\} = M' \rhd \{\upsilon_C(S_{1\text{-}1}) \cup S_{2\text{-}1}, \upsilon_C(S_{1\text{-}1}) \cup S_{2\text{-}2}, \ldots, \upsilon_C(S_{1\text{-}2}) \cup S_{2\text{-}1}, \upsilon_C(S_{1\text{-}2}) \cup S_{2\text{-}2}, \ldots\}$. Two operators needs special attention. π may constrain the signature in the way that some of the sentences from S_1, S_2, … may fall outside $\mathcal{L}(\mathbf{S}(M'))$. Nevertheless, we keep track of these sentences, watching if there are no naming conflicts (if someone tries to reintroduce one or more of the "dummy" names, they have to be changed). υ_C

needs a special "translation" of the sentences to constrain their influence only to $C^{\mathcal{I}}$ (e.g. $D \sqsubseteq E$ to $C \sqcap D \sqsubseteq C \sqcap E$). Under these assumption we prove that checking satisfiability of a concept C in $M \rhd \{S_1, S_2, \ldots\}$ can be performed by checking its satisfiability against each set of sentences S_1, S_2, … (with standard $\mathcal{SHOIQ}$ reasoner). ◆

Theorem 9. Problem (*csat*, $\mathcal{ALB}$, **A**) is decidable.

Proof. The same technique as in the previous proof can be used. Details are omitted for brevity ◆

A question arises whether the result from Theorem 9 can be generalized to more expressive logics. This question is open, although we are pessimistic in this matter because the line of argument used in the proof suggests that the problem is similar to extending such logics with role negations.

6 RELATED WORK

The s-module algebra is inspired by relational algebra, especially in its version published in (Hall *et al*., 1975). In our work we draw an analogy between a knowledge base (or ontology) module and a relation (or a table), and between a model and a tuple (a record in a table).

Some similarities can also be noticed with the algebra of software modules proposed in (Herrmann *et al*., 2007). Here we should compare a s-module to software model and an interpretation to implementation of a system.

Several ontology and module algebra have been developed so far. One of examples is described in (Mitra and Wiederhold, 2004). The authors share with us some motivations (they treat ontologies as contexts, each context represents some viewpoint). However their algebra represents ontologies as RDF graphs. Moreover, explicit mapping between ontologies have to be defined in some extra-ontological language.

One of the most recent algebra for modules (*Neon algebra*) has been proposed in (d'Aquin *et al*., 2008). It also contains operations on a signature (in the case of Neon algebra called an *interface*) and on module contents. However, the contents of the module is understood there as purely linguistic (which also makes some of definitions not intuitive, e.g. the definition of difference states that $(M_1 - M_2) \vDash \alpha$ iff $M_1 \vDash \alpha$ and $M_2 \not\models \alpha$.). Additionally NeOn algebra does not define any join operation.

7 SUMMARY

The s-module algebra provides a broad range of tools for manipulating knowledge. So it constitutes a framework in which many issues concerning modularity of knowledge bases can be efficiently described. It also provides a user with methods of manipulation of contents of s-modules and enables her to override the original (source) way of modularization of an ontology.

Further planned development of the algebra is connected mainly with its practical application: implementation of a KBMS allowing for storing s-modules, creation of knowledge manipulation language for users, and efficient methods of query execution and optimization.

ACKNOWLEDGEMENTS

The work has been supported by Polish Ministry of Research and High Education under grant No. N N516 4115 33.

REFERENCES

d'Aquin, M., *et al*, 2008. NeOn Formalisms for Modularization: Syntax, Semantics, Algebra. *Neon Project, Deliverable D1.1.3*.

Ghidini, C., Serafini, L., and Tessaris, S. 2007. On relating heterogeneous elements from different ontologies. In *Proceedings of the 20th International Workshop of Description Logics*, pp. 283–290.

Herrmann, C., Krahn, H., Rumpe, B., Schindler, M., and Völkel, S. 2007. An Algebraic View on the Semantics of Model Composition. In *Model Driven Architecture - Foundations and Applications*, Springer-Verlag, pp. 99–113.

Hall, P., Hitchcock, P., and Todd, S. 1975. An algebra of relations for machine computation. In *Proceedings of the 2nd ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, Palo Alto, pp. 225–232.

Mitra P., Wiederhold, G. 2004. An Ontology-Composition Algebra. In *Handbook on Ontologies*, Springer-Verlag, pp. 171–216.

Motik, B., Sattler, U., and Studer, R. 2005. Query Answering for OWL-DL with Rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web 3* (1), pp. 41–60.