# Structure Formation to Modularize Ontologies

Serge Autexier and Dieter Hutter [*]

German Research Center for Artificial Intelligence
Bibliotheksstr. 1, 28359 Bremen, Germany
{autexier|hutter}@dfki.de

**Abstract.** It has been well recognized that the structuring of logic-based databases like formal ontologies is an important prerequisite to allow for an efficient reasoning on such databases. Development graphs have been introduced as a formal basis to represent and reason on structured specifications. In this paper we present an initial methodology and a formal calculus to transform unstructured specifications into structured ones. The calculus rules operate on development graphs allowing one to separate specifications coalesced in one theory into a concisely structured graph.

## 1 Introduction

It has been long recognized that the modularity of specifications is an indispensable prerequisite for an efficient reasoning in complex domains. Algebraic specification techniques provide frameworks for structuring complex specifications and the authors introduced the notion of an development graph [4, 1, 6] as a technical means to work with and reason about such structured specifications. Typically ontologies are large and, even if structured, bear the problem of inconsistencies as any large set of axioms. For instance, the SUMO ontology [7] turned out to be inconsistent [10]. Recently Kurz and Mossakowski presented an approach [5] to prove the consistency of an ontology in a modular way using the (structured) architectural specification in CASL and provide mechanisms to compose the models of the individual components to a global one. Furthermore, there has been work [9] to (re-)structure ontologies following locality criteria into modules which cover all aspects about specific concepts. However, since ontology languages have simple imports without renaming, duplicated sub-ontologies that are for instance equal up to renaming remain hidden, thus making the ontologies unnecessarily large, not only from a modeling point of view, but also from a verification point of view as the same derived properties must be derived over an over again.

In this paper we present further steps towards the support for structure formation in large specifications that additionally allows to factorize equivalent sub-specification, i.e. sub-ontologies. The idea is to provide a calculus and a corresponding methodology to crystallize intrinsic structures hidden in a specification and represent them explicitly in terms of development graphs. We start

[*] This work was funded by the German Federal Ministry of Education and Research under grants 01 IS 11013B and 01 IW 10002 (projects SIMPLE and SHIP)

with a trivial development graph consisting of a single node that contains the entire specification. Step by step, the specification is split to different nodes in the development graph resulting in an increasingly richer graph. On the opposite, common concepts that are scattered in different specifications are identified and unified in a common theory (i.e. node).

## 2 Prerequisites

We base our framework on the notions of development graphs to specify and reason about structured specifications. Development graphs $\mathcal{D}$ are acyclic, directed graphs $\langle \mathcal{N}, \mathcal{L} \rangle$, the nodes $\mathcal{N}$ denote individual theories and the links $\mathcal{L}$ indicate theory inclusions with respect to signature morphisms attached to the links.

This approach is based on the notion of institutions [3] $(\mathbf{Sign}_\mathcal{I}, \mathbf{Sen}_\mathcal{I}, \mathbf{Mod}_\mathcal{I}, \models_{\mathcal{I},\Sigma})$, where (i) **Sign** is a category of *signatures*, (ii) $\mathbf{Sen}\colon \mathbf{Sign} \longrightarrow \mathbf{Set}$ is a functor giving the set of *sentences* $\mathbf{Sen}(\Sigma)$ over each signature $\Sigma$, and for each signature morphism $\sigma\colon \Sigma \longrightarrow \Sigma'$, the sentence translation function $\mathbf{Sen}(\sigma)\colon \mathbf{Sen}(\Sigma) \longrightarrow \mathbf{Sen}(\Sigma')$, where often $\mathbf{Sen}(\sigma)(\varphi)$ is written as $\sigma(\varphi)$, (iii) $\mathbf{Mod}\colon \mathbf{Sign}^{op} \longrightarrow \mathcal{CAT}$ is a functor giving the category of *models* over a given signature, and for each signature morphism $\sigma\colon \Sigma \longrightarrow \Sigma'$, the *reduct functor* $\mathbf{Mod}(\sigma)\colon \mathbf{Mod}(\Sigma') \longrightarrow \mathbf{Mod}(\Sigma)$, where often $\mathbf{Mod}(\sigma)(M')$ is written as $M'|_\sigma$, (iv) $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in |\mathbf{Sign}|$ is a satisfaction relation,[1] such that for each $\sigma\colon \Sigma \longrightarrow \Sigma'$ in **Sign**, $M' \models_{\Sigma'} \sigma(\varphi) \Leftrightarrow M'|_\sigma \models_\Sigma \varphi$ holds for each $M' \in \mathbf{Mod}(\Sigma')$ and $\varphi \in \mathbf{Sen}(\Sigma)$ (*satisfaction condition*).

However, the abstractness of the signatures in the definition of institution (they are just a category) makes it difficult to cope with modularization of signatures. Hence, we equip institutions with an additional structure such that signatures behave more set-like. *Institutions with pre-signatures* [2] are defined as institutions equipped with an embedding $|\_| : \mathbf{Sign} \to \mathbf{Set}$, the *symbol functor*, and a map $sym\colon \bigcup_{\Sigma \in |\mathbf{Sign}|} \mathbf{Sen}(\Sigma) \to |\mathbf{Set}|$, such that $\varphi \in \mathbf{Sen}(\Sigma)$ iff $sym(\varphi) \subseteq |\Sigma|$ for all $\varphi \in \bigcup_{\Sigma \in |\mathbf{Sign}|} \mathbf{Sen}(\Sigma)$. The map $sym$ gives the set of symbols used in a sentence, and sentences are uniform in the sense that a well-formed sentence is well-formed over a certain signature iff its symbols belong to that signature. Pre-signatures are sets and pre-signature morphisms $\bar{\sigma}$ mapping pre-signatures are related to corresponding signature morphisms $\sigma$.

Given an $\mathcal{I}$-institution with pre-signatures, each node $N \in \mathcal{N}$ of the graph is a tuple $(sig^N, ax^N, lem^N)$ such that $sig^N$ is a $\mathcal{I}$-pre-signature called the *local signature* of $N$, $ax^N$ a set of $\mathcal{I}$-sentences called the *local axioms* of $N$, and $lem^N$ a set of $\mathcal{I}$-sentences called the *local lemmas* of $N$. $\mathcal{L}$ is a set of global definition links $M \overset{\bar{\sigma}}{\Longrightarrow} N$. Such a link imports the mapped theory of $M$ (by the pre-signature $\bar{\sigma}$) as part of the theory of $N$. Thus we obtain corresponding notions of global (pre-)signatures, axioms and lemmata that are defined inductively as follows:

[1] $|C|$ is the class of objects of a category $C$.

1. $\mathit{Sig}_{\mathcal{D}}(N) = \mathit{sig}^N \cup \bigcup_{M \overset{\bar{\sigma}}{\Longrightarrow} N \in S} \bar{\sigma}(\mathit{Sig}_{\mathcal{D}}(M))$
2. $\mathit{Ax}_{\mathcal{D}}(N) = \mathit{ax}^N \cup \bigcup_{M \overset{\bar{\sigma}}{\Longrightarrow} N \in \mathcal{D}} \sigma(\mathit{Ax}_{\mathcal{D}}(M))$
3. $\mathit{Lem}_{\mathcal{D}}(N) = \mathit{lem}^N \cup \bigcup_{M \overset{\bar{\sigma}}{\Longrightarrow} N \in \mathcal{D}} \sigma(\mathit{Lem}_{\mathcal{D}}(M))$

A node $N \in \mathcal{N}$ is globally reachable from a node $M \in \mathcal{N}$ via a pre-signature morphism $\bar{\sigma}$, $\mathcal{D} \vdash M \overset{\bar{\sigma}}{\rightarrowtail} N$ for short, iff 1. either $M = N$ and $\bar{\sigma} = \mathit{id}$ is the identity pre-signature morphism, or 2. $M \overset{\bar{\sigma}'}{\Longrightarrow} K \in \mathcal{L}$, and $\mathcal{D} \vdash K \overset{\bar{\sigma}''}{\rightarrowtail} N$, with $\bar{\sigma} = \bar{\sigma}'' \circ \bar{\sigma}'$.

The *maximal nodes* (root nodes) $\lceil\mathcal{D}\rceil$ of a graph $\mathcal{D}$ are all nodes without outgoing links. $\mathit{Dom}_{\mathcal{D}}(N) := \mathit{Sig}_{\mathcal{D}}(N) \cup \mathit{Ax}_{\mathcal{D}}(N) \cup \mathit{Lem}_{\mathcal{D}}(N)$ is the set of all signature symbols, axioms and lemmata visible in a node $N$. The *local domain* of $N$, $\mathit{dom}^N := \mathit{sig}^N \cup \mathit{ax}^N \cup \mathit{lem}^N$ is the set of all local signature symbols, axioms and lemmata of $N$. The *imported domain* $\mathit{Imports}_{\mathcal{D}}(N)$ of $N$ in $\mathcal{D}$ is the set of all signature symbols, axioms and lemmas imported via incoming definition links. $\mathit{Dom}_{\mathcal{D}} = \bigcup_{N \in \mathcal{N}} \mathit{Dom}_{\mathcal{D}}(N)$ is the set of all signature symbols, axioms and lemmata occurring in $\mathcal{D}$. Analogously we define $\mathit{Sig}_{\mathcal{D}}$, $\mathit{Ax}_{\mathcal{D}}$, $\mathit{Lem}_{\mathcal{D}}$, and $\mathit{Ass}_{\mathcal{D}}$. $\mathit{Dom}_{\lceil\mathcal{D}\rceil} = \bigcup_{N \in \lceil\mathcal{D}\rceil} \mathit{Dom}_{\mathcal{D}}(N)$ is the set of all signature symbols, axioms and lemmata occurring in the maximal nodes of $\mathcal{D}$.

A node $N$ has a well-formed signature iff $\mathit{Sig}_{\mathcal{D}}(N)$ is a valid $\mathcal{I}^N$-signature. A development graph has a well-formed signature iff all its nodes have well-formed signatures. $\mathit{Sig}^{loc}_{\mathcal{D}}(M) := \langle \mathit{sig}^M \cup \mathit{sym}(\mathit{ax}^M) \cup \mathit{sym}(\mathit{lem}^M)\rangle_{\mathit{Sig}_{\mathcal{D}}(M)}$ is the *local signature* of $N$. A node $N$ is well-formed iff it has a well-formed signature $\mathit{Sig}_{\mathcal{D}}(N)$ and $\mathit{Ax}_{\mathcal{D}}(N), \mathit{Lem}_{\mathcal{D}}(N) \subseteq \mathbf{Sen}(\mathit{Sig}_{\mathcal{D}}(N))$. A development graph is well-formed, if all its nodes are well-formed.

Given a node $N \in \mathcal{N}$ with well-formed signature, its associated class $\mathbf{Mod}^{\mathcal{D}}(N)$ of models (or $N$-models for short) consists of those $\mathit{Sig}_{\mathcal{D}}(N)$-models $n$ for which (i) $n$ satisfies the local axioms $\mathit{ax}^N$, and (ii) for each $K \overset{\bar{\sigma}}{\Longrightarrow} N \in \mathcal{S}$, $n|_\sigma$ is a $K$-model. In the following we denote the class of $\Sigma$-models that fulfill the $\Sigma$-sentences $\Psi$ by $\mathbf{Mod}_\Sigma(\Psi)$. A well-formed development graph $\mathcal{D} := \langle \mathcal{N}, \mathcal{L}\rangle$ is *valid* iff for all nodes $N \in \mathcal{N}$ $\mathbf{Mod}^{\mathcal{D}}(N) \models \mathit{lem}^N$.

Given a signature $\Sigma$ and $\mathit{Ax}, \mathit{Lem} \subseteq \mathbf{Sen}(\Sigma)$, a *support mapping Supp for Ax and Lem* assigns each lemma $\varphi \in \mathit{Lem}$ a subset $H \subseteq \mathit{Ax} \cup \mathit{Lem}$ such that (i) $\mathbf{Mod}_{\langle \mathit{sym}(H) \cup \mathit{sym}(\varphi)\rangle_\Sigma}(H) \models \varphi$ (ii) The relation $\sqsubset \subseteq (\mathit{Ax} \cup \mathit{Lem}) \times \mathit{Lem}$ with $\Phi \sqsubset \varphi \Leftrightarrow (\Phi \in \mathit{Supp}(\varphi) \vee \exists \psi . \Phi \in \mathit{Supp}(\psi) \wedge \psi \sqsubset \varphi)$ is a well-founded strict partial order. If $\mathcal{D}$ is a development graph, then a support mapping $\mathit{Supp}$ is a *support mapping for* $\mathcal{D}$ iff for all $N \in \mathcal{D}$ $\mathit{Supp}$ is a support mapping for $\mathit{Ax}_{\mathcal{D}}(N)$ and $\mathit{Lem}_{\mathcal{D}}(N)$.

## 3 Development Graphs for Structure Formation

As a first step towards structure formation we will formalize requirements on development graphs that reflect our intuition of an appropriate structuring for (formal) ontologies in particular (and formal specifications in general) in the following principles.

The first principle is *semantic appropriateness*, saying that the structure of the development graph should be a syntactical reflection of the relations between the various concepts in our ontology. This means that different concepts are located in different nodes of the graph and the links of the graph reflect the logical relations between these concepts. The second principle is *closure* saying, for instance, that deduced knowledge should be located close to the concepts guaranteeing the proofs. Also the concept defined by the theory of an individual node of a development graph should have a meaning of its own and provide some source of deduced knowledge. The third principle is *minimality* saying that each concept (or part of it) is only represented once in the graph. When splitting a monolithic theory into different concepts common foundations for various concepts should be (syntactically) shared between them by being located at a unique node of the graph.

In the following we translate these principles into syntactical criteria on development graphs and also into rules to transform and refactor development graphs. In a first step we formalize technical requirements to enforce the minimality-principle in terms of development graphs. Technically, we demand that each signature symbol, each axiom and each lemma has a unique location in the development graph. When we enrich a development graph with more structure we forbid to have multiple copies of the same definition in different nodes. We therefore require that we can identify for a given signature entry, axiom or lemma a *minimal theory* in a development graph and that this minimal theory is unique. We define:

**Definition 1 (Providing Nodes).** *Let $\langle \mathcal{N}, \mathcal{L} \rangle$ be a development graph. An entity $e$ is* provided *in $N \in \mathcal{N}$ iff $e \in Dom_{\langle \mathcal{N}, \mathcal{L} \rangle}(N)$ and $\forall M \xRightarrow{\bar{\sigma}} N.\ e \notin Dom_{\langle \mathcal{N}, \mathcal{L} \rangle}(M)$. Furthermore,*

1. *$e$ is* locally *provided in $N$ iff additionally $e \in dom^N$ holds.*
2. *$e$ is provided* by a link *$l : M \xRightarrow{\bar{\sigma}} N$ if not locally provided in $N$ and $\exists e' \in Dom_{\langle \mathcal{N}, \mathcal{L} \rangle}(M).\ \sigma(e') = e$. In this case we say that $l$ provides $e$ from $e'$. $e$ is* exclusively *provided by $l$ iff $e$ is not provided by any other link $l' \in \mathcal{L}$.*

Finally, the closure-principle demands that there are no spurious nodes in the graph that do not contribute anything new to a concept. We combine these requirements into the notion of location mappings:

**Definition 2 (Location Mappings).** *Let $\mathcal{D} = \langle \mathcal{N}, \mathcal{L} \rangle$ be a development graph. A mapping $loc_{\mathcal{D}} : Dom_{\mathcal{D}} \to \mathcal{N}$ is a* location *mapping for $\mathcal{D}$ iff*

1. *$loc_{\mathcal{D}}$ is surjective (closure)*
2. *$\forall N \in \mathcal{N}.\ \forall e \in dom^N.\ loc_{\mathcal{D}}(e) = N$*
3. *$\forall e \in Dom_{\mathcal{D}}.\ loc_{\mathcal{D}}(e)$ is the only node providing $e$ (minimality)*

*Furthermore, for a given $loc_{\mathcal{D}}$ we define $loc_{\mathcal{D}}^{-1} : \mathcal{N} \to 2^{Dom_{\mathcal{D}}}$ by $loc_{\mathcal{D}}^{-1}(N) := \{e \in Dom_{\mathcal{D}} | loc_{\mathcal{D}}(e) = N\}$. We will write $loc$ and $loc^{-1}$ instead of $loc_{\mathcal{D}}$ and $loc_{\mathcal{D}}^{-1}$ if $\mathcal{D}$ is clear from the context.*

Based on the notion of location mappings we formalize our intuition of a *structuring.* The idea is that the notion of being a structuring constitutes the invariant of the structure formation process and guarantees both, requirements imposed by the minimality-principle as well as basic conditions on a development graph to reflect a given formal specification or ontology.

**Definition 3 (Structuring).** *Let $\mathcal{D} = \langle \mathcal{N}, \mathcal{L} \rangle$ be a valid development graph, $loc : Dom_{\mathcal{D}} \to \mathcal{N}$, $\Sigma \in |\mathbf{Sign}|$, $Ax, Lem \subseteq \mathbf{Sen}(\Sigma)$ and Supp be a support mapping for $\mathcal{D}$. Then $(\mathcal{D}, loc, Supp)$ is a* structuring *of $(\Sigma, Ax, Lem)$ iff*

1. *loc is a location mapping for $\mathcal{D}$.*
2. *let $Dom_{\lceil \mathcal{D} \rceil} = \Sigma' \cup Ax' \cup Lem'$ then $\Sigma = \Sigma'$, $Ax = Ax'$ and $Lem \subseteq Lem'$.*
3. *$\forall \phi \in Lem_{\mathcal{D}} . \forall \psi \in Supp(\phi). \exists \bar{\sigma}. \, loc(\psi) \overset{\bar{\sigma}}{\Longrightarrow} loc(\phi) \wedge \bar{\sigma}(\psi) = \psi$*

## 4 Refactoring Rules

In the following we present the transformation rules on development graphs that transform structurings again into structurings. Using these rules we are able to structure the initially trivial development graph consisting of exactly one node that comprises all given concepts step by step. This initial development graph consisting of exactly one node satisfies the condition of a structuring provided that we have an appropriate support mapping at hand.

We define four types of structuring-invariant transformations: (i) horizontal splitting and merging of development graph nodes, (ii) vertical splitting and merging of development graph nodes, (iii) factorization and multiplication of development graph nodes, and (iv) removal and insertion of specific links. Splitting and merging as well as factorization and multiplication are dual operations. For lack of space and because we are mainly interested in rules increasing the structure of a development graph we will omit the formal specification of the merging and multiplication rules here.

We illustrate our rules with the help of a running example in mathematics. We start from a flat theory specifying a Ring over two operations $+$ and $\times$. A structure $(R, +, \times)$ is a Ring, if $(R, \times)$ is an abelian group, $(R, +)$ a monoid and $\times$ distributes over $+$. Furthermore, an abelian group is a monoid for which additionally commutativity holds and inverse exists. The initial development graph consists of a single node (without any links) containing all the symbol definitions, axioms and theorems of the example.

*Vertical Split.* The first refactoring rule aims at the split of specifications in different theories. In terms of the development graph a node is replaced by two nodes one of them importing the other; each of them contains a distinct part from a partitioning of the specification of the original node. While all outgoing links start at the top node, we are free to reallocate incoming links to either node. To formalize this rule we need constraints on how to split a specification in different chunks such that local lemmata are always located in a node which provides also the necessary axioms and lemmata to prove it.

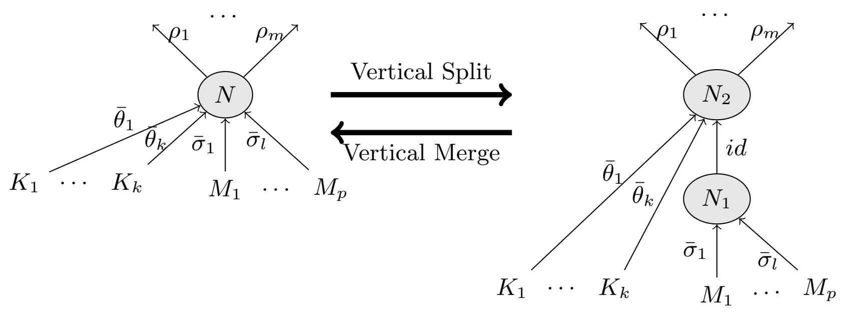

**Fig. 1.** Vertical Split and Merge

**Definition 4.** *Let $\mathcal{S} = (\mathcal{D}, loc, Supp)$ be a structuring of $(\Sigma, Ax, Lem)$ and $N \in \mathcal{N}_\mathcal{D}$. A partitioning $\mathcal{P}$ for $N$ is a set $\{N_1, \dots, N_k\}$ with $k > 1$ such that 1. $sig^N = sig^{N_1} \uplus \dots \uplus sig^{N_k}$, $ax^N = ax^{N_1} \uplus \dots \uplus ax^{N_k}$, $lem^N = lem^{N_1} \uplus \dots \uplus lem^{N_k}$ 2. $sig^{N_i} \cup ax^{N_i} \cup lem^{N_i} \neq \emptyset$ for $i = 1, \dots, k$. A node $N_i \in \mathcal{P}$ is* lemma independent *iff $Supp(\psi) \cap (ax^N \cup lem^N) \subseteq (ax^{N_i} \cup lem^{N_i})$ for all $\psi \in lem^{N_i}$.*

**Definition 5 (Vertical Split).** *Let $\mathcal{S} = (\langle \mathcal{N}, \mathcal{L} \rangle, loc, Supp)$ be a* structuring *of $(\Sigma, Ax, Lem)$ and $\mathcal{P} = \{N_1, N_2\}$ be a partitioning for some $N \in \mathcal{N}$ such that $N_1$ is lemma independent. Then, the vertical split $\mathcal{S}$ wrt. $N$ and $\mathcal{P}$ is $\mathcal{S}' = (\mathcal{D}', loc', Supp)$ with $\mathcal{D}' = \langle \mathcal{N}', \mathcal{L}' \rangle$ where*

$$\mathcal{N}' := \{N_1, N_2\} \uplus (\mathcal{N} \setminus N)$$

$$\mathcal{L}' := \{ M \xRightarrow{\bar{\sigma}} M' \in \mathcal{L} | M \neq N \wedge M' \neq N\} \cup \{ N_1 \xRightarrow{id} N_2\}$$

$$\cup \{ M \xRightarrow{\bar{\sigma}} N_1 \mid M \xRightarrow{\bar{\sigma}} N \in \mathcal{L}\} \cup \{ N_2 \xRightarrow{\bar{\sigma}} M \mid N \xRightarrow{\bar{\sigma}} M \in \mathcal{L}\}$$

$$loc'(e) = \begin{cases} N_2 & \text{if } loc(e) = N \text{ and } e \in Dom_{\mathcal{D}'}(N_2) \\ N_1 & \text{if } loc(e) = N \text{ and } e \notin Dom_{\mathcal{D}'}(N_2) \\ loc(e) & \text{otherwise} \end{cases}$$

*such that $Sig_{\mathcal{D}'}(N_i), i = 1, 2$, are valid signatures and $ax_i, lem_i \subseteq \mathbf{Sen}(Sig_{\mathcal{D}'}(N_i))$, $i = 1, 2$. Conversely, $\mathcal{S}$* is a vertical merge of $N_1$ and $N_2$ in $\mathcal{S}'$.

*Horizontal Split.* Similar to a vertical split we introduce a horizontal split which divides a node into two independent nodes. In order to ensure a valid new development graph, each of the new nodes imports the same theories as the old node and contributes to the same theories as the old node did.

**Definition 6 (Horizontal Split).** *Let $\mathcal{S} = (\langle \mathcal{N}, \mathcal{L} \rangle, loc, Supp)$ be a* structuring *of $(\Sigma, Ax, Lem)$, $\mathcal{P} = \{N_1, \dots, N_k\}$ be a partitioning for some node $N \in \mathcal{N}$ such that each $N_i \in \mathcal{P}$ is lemma independent and $loc^{-1}(N) = dom^N$. The* horizontal split *of $\mathcal{S}$ wrt. $N$ and $\mathcal{P}$ is $\mathcal{S}' = (\mathcal{D}', loc', Supp)$ with $\mathcal{D}' = \langle \mathcal{N}', \mathcal{L}' \rangle$ where*
*1.* $\mathcal{N}' := \{N_1, \dots, N_k\} \uplus (\mathcal{N} \setminus N)$

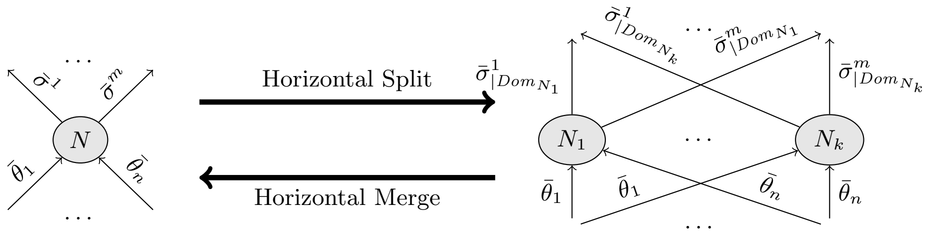
Horizontal Split
Horizontal Merge
$N$
$N_1$
$N_k$
$\bar{\sigma}^1$
$\bar{\sigma}^m$
$\bar{\theta}_1$
$\bar{\theta}_n$
$\bar{\sigma}^1_{|Dom_{N_1}}$
$\bar{\sigma}^1_{|Dom_{N_k}}$
$\bar{\sigma}^m_{|Dom_{N_1}}$
$\bar{\sigma}^m_{|Dom_{N_k}}$
. . .

**Fig. 2.** Horizontal Split and Merge

2. $\mathcal{L}' := \{ M \xRightarrow{\bar{\sigma}} M' \in \mathcal{L} | M \neq N \wedge M' \neq N \}$
$\cup \{ M \xRightarrow{\bar{\theta}} N_i | M \xRightarrow{\bar{\theta}} N \in \mathcal{L}, i \in \{1, \dots, k\}\}$
$\cup \{ N_i \xRightarrow{\bar{\tau}_{|Dom_{N_i}}} M | N \xRightarrow{\bar{\tau}} M \in \mathcal{L}, i \in \{1, \dots, k\}\}$
3. *$loc'(e) := N_i$ if $e \in dom^{N_i}$ for some $i \in \{1, \dots, k\}$ and $loc'(e) := loc(e)$ otherwise.*

*such that $Sig_{\mathcal{D}'}(N_i)$ are valid signatures and $ax_i, lem_i \subseteq \mathbf{Sen}(Sig_{\mathcal{D}'}(N_i))$ for $i = 1, \dots, k$.*

Using the transformation rules, the flat initial theory of our running example can be refactored (cf. Fig. 3). We apply the vertical split rule twice to extract $R$ and the distributivity law, followed by the horizontal-split rule to separate the two instances of a monoid. Finally, we apply the vertical-split rule to extract the extra axioms from $Monoid(R, \times)$. We are left with two copies of a monoid which we like to generalize to a common abstract monoid. This can be achieved by the following rule.

*Factorization* The factorization rule allows one to merge equivalent concepts into a single generalized concept and then to represent the individual ones as instantiations of the generalized concept. A precondition of this rule is that all individual concepts inherit the same (underlying) theories.

**Definition 7 (Factorization).** *Let $\mathcal{S} = (\langle \mathcal{N}, \mathcal{L} \rangle, loc, Supp)$ be a* structuring *of $(\Sigma, Ax, Lem)$. Let $K_1, \dots, K_n, M_1, \dots, M_p \in \mathcal{N}$ with $p > 1$ such that $sig^{M_j} \cup ax^{M_j} \neq \emptyset$ and $\exists \bar{\sigma}_{i,j}.\ K_i \xRightarrow{\bar{\sigma}_{i,j}} M_j \in \mathcal{L}$ for $i = 1, \dots, n, j = 1, \dots, p$.*

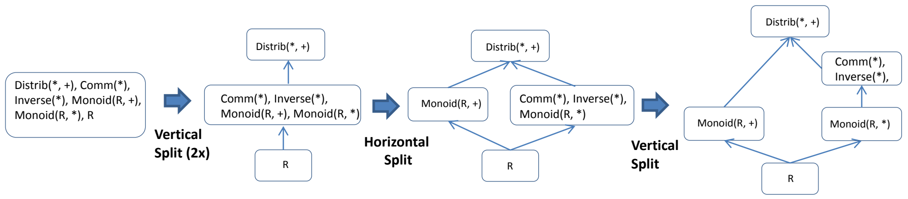

**Fig. 3.** Applying the split rules to our ring example

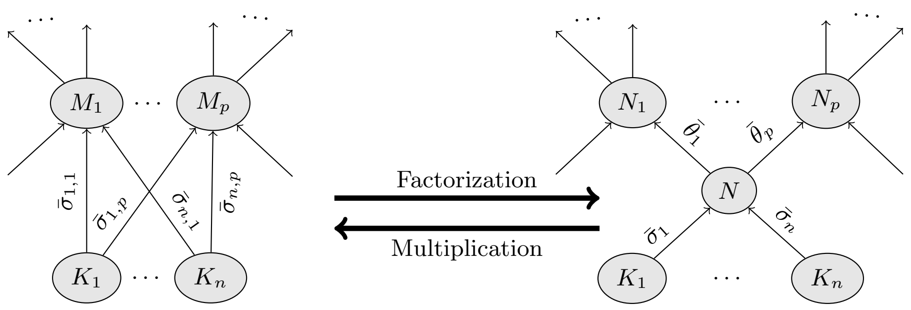

**Fig. 4.** Factorization and Multiplication (with $\bar{\sigma}_{i,j} := \bar{\theta}_j \circ \bar{\sigma}_i$)

*Suppose there are sets sig, ax and lem with* $(sig \cup ax \cup lem) \cap Dom_{\mathcal{D}} = \emptyset$ *and pre-signature morphisms* $\theta_1, \ldots, \theta_p$ *and* $\sigma_1, \ldots, \sigma_n$ *such that*
- $\forall e \in Dom_{\mathcal{D}}(K_i).\ \theta_j(\sigma_i(e)) = \sigma_{i,j}(e)$ *and* $\sigma_{i,j}(e) = e \vee \sigma_{i,j}(e) \notin Dom_{\mathcal{D}}$
- $sig^{M_j} \subseteq \theta_j(sig) \subseteq Dom_{\mathcal{D}}(M_j)$, $ax^{M_j} \subseteq \theta_j(ax) \subseteq Dom_{\mathcal{D}}(M_j)$
- $\forall e \in lem$ *holds* $\exists l \in \{1, \ldots p\}.\ \theta_l(e) \in lem^{M_l}$, $\theta_i(e) = \theta_j(e)$ *implies* $i = j$ *and* $\theta_j(e) \in Dom_{\mathcal{D}}$ *implies* $loc(\theta_j(e)) \in M_j$
- *there is a support mapping* $Supp_N$ *for* $ax \cup \bigcup_{i=1,\ldots,n} \sigma_i(Dom_{\mathcal{D}}(K_i))$ *and lem.*

*Then* $\mathcal{S}' = (\langle \mathcal{N}', \mathcal{L}' \rangle, loc', Supp')$ *is a factorization of* $\mathcal{S}$ *with respect to* $M_1, \ldots, M_p$ *and* $Supp_N$ *iff*

$$\mathcal{N}' := \{N\} \cup \{N_j | j \in \{1, \ldots p\}\} \cup \mathcal{N} \setminus \{M_1, \ldots M_p\}$$
$$\text{with } N = \langle sig, ax, lem \rangle, N_j = \langle \emptyset, \emptyset, lem^{M_j} \setminus \theta_j(lem) \rangle$$
$$\mathcal{L}' := \{ K \xRightarrow{\bar{\sigma}} K' \in \mathcal{L} | K, K' \notin \{M_1, \ldots M_p\}$$
$$\cup \{ K_i \xRightarrow{\bar{\sigma}_i} N | K_i \xRightarrow{\bar{\sigma}_{i,j}} M_j, j \in \{1, \ldots p\}, i \in \{1, \ldots n\}\}$$
$$\cup \{ N \xRightarrow{\theta_j} N_j | j \in \{1, \ldots p\}\}$$
$$\cup \{ K \xRightarrow{\bar{\tau}} N_j | K \xRightarrow{\bar{\tau}} M_j \wedge (\forall i \in \{1, \ldots n\}. K \neq K_i \wedge \bar{\tau} \neq \bar{\sigma}_{i,j})$$
$$\cup \{ N_j \xRightarrow{\bar{\tau}} K | M_j \xRightarrow{\bar{\tau}} K \in \mathcal{L}, j \in \{1, \ldots p\}\}$$
$$loc'(x) := \begin{cases} N & \text{if } x \in Dom_{\mathcal{D}'}(N) \setminus \bigcup_{i=1,\ldots,n} Dom_{\mathcal{D}'}(K_i) \\ N_j & \text{if } x \in Dom_{\mathcal{D}}(N_j) \text{ and } \forall K \xRightarrow{\bar{\sigma}} N_j.\ x \notin Dom_{\mathcal{D}'}(K) \\ loc(x) & \text{otherwise.} \end{cases}$$
$$Supp' := Supp \cup Supp_N.$$

Applying the factorization rule allows us to introduce a generic concept of monoids which is instantiated to obtain both previous copies of a monoid.

The factorization rule only covers a sufficient criterion demanding that each theory imported by a definition link to one concept is also imported via definition links by all other concepts. The more complex case in which a theory is imported

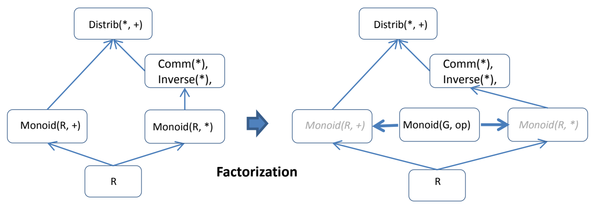

**Fig. 5.** Factorization of our ring example

via a path of links can be handled by allowing one to shortcut a path in a single global link. This results in the following rule.

**Definition 8 (Transitive Enrichment).** *Let $\mathcal{S} := (\langle\mathcal{N},\mathcal{L}\rangle, loc, Supp)$ be a structuring of $(\Sigma, Ax, Lem)$, $K, N \in \mathcal{N}$ and there is a path $K \succ\!\xRightarrow{\bar{\sigma}} N$ between both. Then, $(\langle\mathcal{N},\mathcal{L} \cup \{K \xRightarrow{\bar{\sigma}} N\}\rangle, loc, Supp)$ is a transitive enrichment of $\mathcal{S}$.*

Definition links in a development graph can be redundant, if there are alternatives paths which have the same morphisms or if they are not used in any reachable node of the target. We formalize these notions as follows:

**Definition 9 (Removable Link).** *Let $\mathcal{S} = (\mathcal{D}, loc, Supp)$ $(\mathcal{D} = \langle\mathcal{N},\mathcal{L}\rangle)$ be a structuring of $(\Sigma, Ax, Lem)$. Let $l \in \mathcal{L}$ and $\mathcal{D}' = \langle\mathcal{N},\mathcal{L} \setminus \{l\}\rangle$. $l$ is removable from $\mathcal{S}$ and $\mathcal{S}' = (\mathcal{D}', loc, Supp)$ is a* reduction *of $\mathcal{S}$ iff*

1. *$\forall l' : M \xRightarrow{\bar{\sigma}} N$. if $l'$ provides exclusively $\sigma(e)$ from some $eDom_{\mathcal{D}}(M)$ then $e \in Dom_{\mathcal{D}'}(N)$ and $l \neq l'$;*
2. *$\forall e \in Dom_{\mathcal{D}}.\forall M \in DGRoots\mathcal{D}$. if $loc(e) \succ\!\xRightarrow{\bar{\sigma}} M$ then there exists $M' \in \lceil\mathcal{D}'\rceil$ such that $loc(e) \succ\!\xRightarrow{\bar{\sigma}} M'$;*
3. *$\forall \phi \in Lem_{\mathcal{D}}$. $Supp(\phi) \subseteq Dom_{\mathcal{D}'}(N)$ and $\forall Sig^{loc}_{\mathcal{D}}(N) \subseteq Dom_{\mathcal{D}'}(N)$.*

**Theorem 1 (Structuring Preservation).** *Let $\mathcal{S} := (\mathcal{D}, loc, Supp)$ $(\mathcal{D} = \langle\mathcal{N},\mathcal{L}\rangle)$ be a structuring of $(\Sigma, Ax, Lem)$. Then*
1. *every horizontal split of $\mathcal{S}$ wrt. some $N \in \mathcal{N}$ and partitioning $\mathcal{P}$ of $N$,*
2. *every horizontal merge of $\mathcal{S}$ wrt. nodes $\{N_1, \ldots, N_k\} \subseteq \mathcal{N}$,*
3. *every vertical split of $\mathcal{S}$ wrt. some $N \in \mathcal{N}$ and partitioning $\mathcal{P}$ of $N$,*
4. *every factorization of $\mathcal{S}$ wrt. nodes $M_1, \ldots M_p \in \mathcal{N}$,*
5. *every multiplication of $\mathcal{S}$ wrt. $N$,*
6. *every transitive enrichment of $\mathcal{S}$, and*
7. *every reduction of $\mathcal{S}$*

*is a structuring of $(\Sigma, Ax, Lem)$.*

## 5 Refactoring Process

The refactoring rules presented above provide the necessary instruments to externalize the structure inherent in a given flat theory. Nevertheless we need

**C**har
Sig. $\mathsf{char}, A, B, \ldots, Z$
Ax. $\forall x : \mathsf{char}.\ x = A \vee \cdots \vee Z$
Lem. —
**S**tring
Sig. $\mathsf{str}, \mathsf{strnil}, \mathsf{addc}$
Ax. $\forall c : \mathsf{char}, x : \mathsf{str}.\ \mathsf{strnil} \neq \mathsf{addc}(c, x)$
$\forall x : \mathsf{str}.\ x = \mathsf{strnil} \vee \exists c : \mathsf{char}, y : \mathsf{str}.$
$x = \mathsf{addc}(c, y)$
$\forall c, c' : \mathsf{char}, x, x' : \mathsf{str}.\ \mathsf{addc}(c, x) = \mathsf{addc}(c', x')$
$\to c = c' \wedge x = x', \ldots\}$
Lem. —
**S**tringops
Sig. $\mathsf{strapp}, \mathsf{strlen}$
Ax. $\mathsf{strlen}(\mathsf{strnil}) = 0$
$\forall c : \mathsf{char}, x : \mathsf{str}.\ \mathsf{strlen}(\mathsf{addc}(c, x)) = \mathsf{succ}(x)$
$\forall x : \mathsf{str}.\ \mathsf{strapp}(\mathsf{strnil}, x) = x, \ldots$
Lem. $\forall x, y, z : \mathsf{str}.\ \mathsf{strapp}(\mathsf{strapp}(x, y), z)$
$= \mathsf{strapp}(x, \mathsf{strapp}(y, z)), \ldots$

**N**at
Sig. $\mathsf{nat}, 0, \mathsf{succ}$
Ax. $\forall x : \mathsf{nat}.\ 0 \neq \mathsf{succ}(x)$
$\forall x : \mathsf{nat}.\ x = 0 \vee \exists y : \mathsf{nat}.\ x = \mathsf{succ}(y), \ldots$
Lem. —
**N**atlist
Sig. $\mathsf{nlist}, \mathsf{natnil}, \mathsf{addn}$
Ax. $\forall n : \mathsf{nat}, x : \mathsf{nlist}.\ \mathsf{natnil} \neq \mathsf{addn}(n, x)$
$\forall x : \mathsf{nlist}.\ x = \mathsf{natnil}$
$\vee \exists n : \mathsf{nat}, y : \mathsf{nlist}.\ x = \mathsf{addn}(n, y)$
$\forall n, n' : \mathsf{nat}, x, x' : \mathsf{nlist}.\ \mathsf{addn}(n, x) = \mathsf{addn}(n', x')$
$\to n = n' \wedge x = x', \ldots$
Lem. —
**N**atlistops
Sig. $\mathsf{natapp}, \mathsf{nlen}$
Ax. $\mathsf{nlen}(\mathsf{natnil}) = 0$
$\forall n : \mathsf{nat}, x : \mathsf{nlist}.\ \mathsf{nlen}(\mathsf{addn}(n, x)) = \mathsf{succ}(x)$
$\forall x : \mathsf{nlist}.\ \mathsf{natapp}(\mathsf{natnil}, x) = x, \ldots$
Lem. $\forall x, y, z : \mathsf{nlist}.\ \mathsf{natapp}(\mathsf{natapp}(x, y), z)$
$= \mathsf{natapp}(x, \mathsf{natapp}(y, z)) \ldots$

**Fig. 6.** Heuristic motivated partitioning of a flat theory

appropriate heuristics to determine suitable partitions for horizontal or vertical splits, in order to group together strongly related entities and afterwards make explicit analogous groupings of entities by factorization. One heuristic is, for instance, to group together all axioms and lemmas exclusively devoted to a specific sort. E.g. all axioms about characters, or all axioms about natural numbers. From there we carve out those entities about a different sort *including* an already classified sorts, and so on. Examples for these are strings or natural numbers, and in the next iteration lists of strings (cf. Fig. 6). These identified subsets can then be further partitioned into those defining the basic datatype and the (inductive) functions and predicates over these. This separates the definition of lists of natural numbers from the append functions on these, for instance.

Such heuristics guide the application of the horizontal and vertical split rules. From there we can identify nodes with isomorphic local entries and, using the transitivity and reduction rules together with further applications of the split rules try to enable the application of the factorization rule.

We illustrate the refactoring process with the help of a further example, where we start with flattened theories, and step by step carve out the intrinsic structure. The example from Fig. 6 is about lists of natural numbers, strings formed over characters and length functions operating on such lists. All the signature symbols, axioms, and lemmas occur in a single node in the initial development graph. For sake of readability we partition the set of these entities into pairwise disjunct subsets and name them accordingly (cf. [8])

Fig. 7 illustrates the structure formation process for this example. In the first step the definitions of Nat and Char are separated from the rest by applying the vertical split rule twice. In the next steps we form the individual theories of String and Stringops respectively. Notice that Stringops includes strlen counting characters in a string such that Stringops has to import Nat. After applying again a vertical split we obtain the theories of Natlist and Natlistops. Since the local axioms of String and Natlist are renamings of each other we factorize these local axioms to a new theory of generic lists.

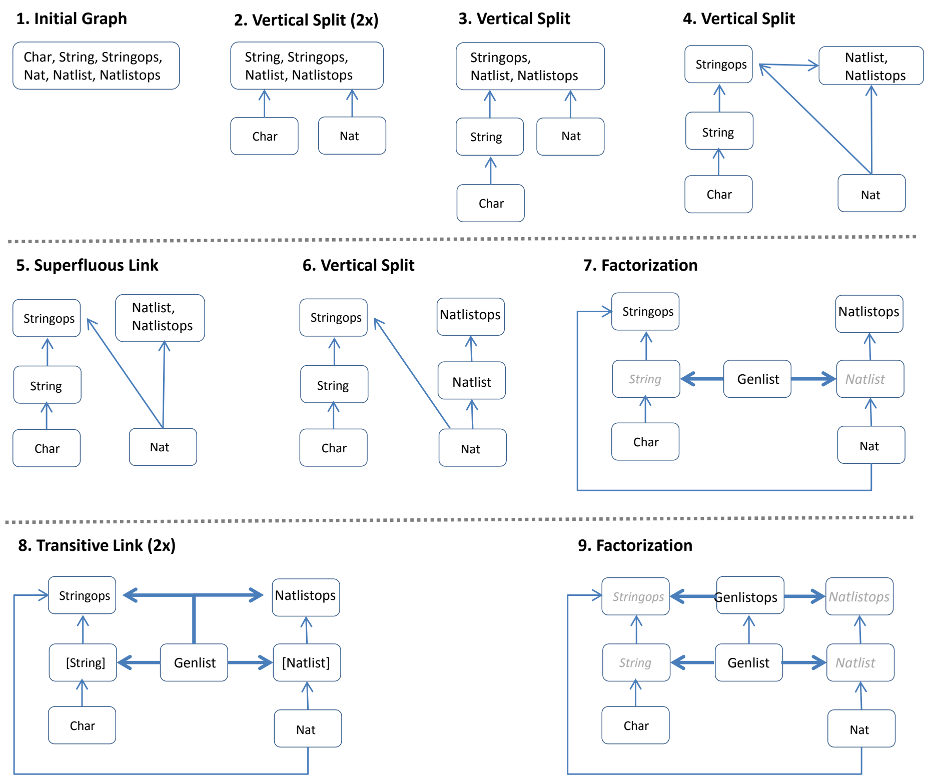

**Fig. 7.** Lists

This node consists of the renamed local axioms of String (or Natlist, respectively) plus the necessary signature definitions to obtain a well-formed development graph node. This theory is imported via definition links to String and Natlist using corresponding pre-signature morphisms to map it to list of chars and numbers, respectively. Both, String and Natlist have now empty local signatures and axioms. In Fig 7 we indicate those nodes with empty local signature, empty local axioms and empty local lemmata by a light-gray color of the node name.

**Genlist**
Sig. list, nil, add
Ax. $\forall n : \mathsf{elem}, x : \mathsf{list}.\ \mathsf{nil} \neq \mathsf{add}(n, x)$
$\forall x : \mathsf{list}.\ x = \mathsf{nil}$
$\vee\ \exists n : \mathsf{elem}, y : \mathsf{list}.\ x = \mathsf{add}(n, y)$
$\forall n, n' : \mathsf{elem}, x, x' : \mathsf{list}.\ \mathsf{add}(n, x) = \mathsf{add}(n', x')$
$\rightarrow n = n' \wedge x = x', \ldots$
Lem. —

## 6 Related Work

There is related work to (re-)structure ontologies (after flattening) following locality criteria into modules containing all knowledge about specific concepts. However, since ontology languages have simple imports without renamings, duplicated sub-ontologies that are for instance equal up to renaming remain hidden.

To excavate these ontologies requires other heuristics than pure logic based ones to guide the structuring process, and therefore we deliberately did not impose specific criteria how to apply the rules beyond the minimal conditions that the dependency relations among concepts, relations and facts and derived facts and relations is preserved. It is easy to see that the modularizations of ontologies not including derived lemmas obtained by using the locality criteria from [9] can be constructed with our rules by starting from a single flattened ontology and singling out the modules.

## 7 Conclusion

Based on the definition of structurings as concise development graphs, we presented transformation rules that allow one to make explicit common structures hidden in a flat theory in terms of development graphs. It provides a framework to modularize flattened ontologies in a useful way as illustrated by two simple examples. An implementation is planned to provide an interactive tool to modularize and factorize large ontologies as well as (semi-)automatic procedures.

## References

1. S. Autexier and D. Hutter. Mind the gap - maintaining formal developments in MAYA. In *Festschrift in Honor of J.H. Siekmann.* Springer, LNCS 2605, 2005.
2. S. Autexier, D. Hutter, and T. Mossakowski. Change management for heterogeneous development graphs. In S. Siegler and N. Wasser, editors, *Verification, Induction, Termination Analysis (Festschrift for Christoph Walther)*, volume 6463 of *LNAI*, pages 54–80. Springer, 2010.
3. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. Predecessor in: LNCS 164, 221–256, 1984.
4. D. Hutter. Management of change in verification systems. In *Proceedings 15th IEEE International Conference on Automated Software Engineering, ASE-2000*, pages 23–34. IEEE Computer Society, 2000.
5. O. Kutz and T. Mossakowski. A modular consistency proof for DOLCE. In *The Association for the Advancement of Artificial Intelligence (AAAI-2011)*, 2011.
6. T. Mossakowski, S. Autexier, and D. Hutter. Development graphs - proof management for structured specifications. *Journal of Logic and Algebraic Programming, special issue on Algebraic Specification and Development Techniques*, 67(1-2):114–145, april 2006.
7. I. Niles and A. Pease. Towards a standard upper ontology. In *Formal Ontology in Information Systems (FOIS-2001).* ACM Press, 2001.
8. I. Normann and M. Kohlhase. Extended formula normalization for $\epsilon$ -retrieval and sharing of mathematical knowledge. In *Towards Mechanized Mathematical Assistants (Calculemus/MKM).* Springer, LNCS 4573, 2007.
9. C. D. Vescovo, B. Parsia, U. Sattler, and T. Schneider. The modular structure of an ontology: Atomic decomposition. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'10)*, pages 2232–2237, 2010.
10. A. Voronkov. Inconsistencies in ontologies. In *JELIA 2006.* Springer LNCS 4160, 2006.