Development Graphs – Proof Management for Structured Specifications

Till Mossakowski [a], Serge Autexier [b], Dieter Hutter [b]

[a] *BISS, Department of Computer Science, University of Bremen*

[b] *DKFI GmbH, Saarbrücken*

Abstract

Development graphs are a tool for dealing with structured specifications in a formal program development in order to ease the management of change and reusing proofs. In this work, we extend development graphs with hiding (e.g. hidden operations). Hiding is a particularly difficult to realize operation, since it does not admit such a good decomposition of the involved specifications as other structuring operations do. We develop both a semantics and proof rules for development graphs with hiding. The rules are proven to be sound, and also complete relative to an oracle for conservative extensions. We also show that an absolutely complete set of rules cannot exist.

The whole framework is developed in a way independent of the underlying logical system (and thus also does not prescribe the nature of the parts of a specification that may be hidden). We also show how various other logic independent specification formalisms can be mapped into development graphs; thus, development graphs can serve as a kernel formalism for management of proofs and of change.

Key words: Algebraic specification; structuring; proof calculus; institutions.

1 Introduction

It has long been recognized that *specifications in the large* are only manageable if they are built in a structured way. Specification languages, like Casl [19], provide various mechanisms to combine basic specifications to structured specifications. Analogously, verification tools have to provide appropriate mechanisms to structure the corresponding logical axiomatizations. In practice, a formal program development is an evolutionary process [12]. Specification and verification are mutually intertwined. Failed proofs give rise to changes of the specification which in turn will render previously found proofs invalid. For practical purposes it is indispensable to restrict the effects of such changes to a minimum in order to preserve as much proof effort as possible after a change of the specification.

Various structuring operations have been proposed (e.g. [7,21,22]) in order to modularize specifications and proof systems have been described to deal with them (e.g. [6,11]). Traditionally, the main motivations for modularization have been the sharing of sub-specifications within one specification, the reuse of specifications, and the structuring of proof obligations as well as applicable lemmas. However, the structure of specifications can also be exploited when the effects of changes are analyzed.

In [3], development graphs have been introduced as a tool for dealing with structured specifications in a way easing management of change and reusing proofs. Also, a translation of structured specifications in CASL, an international standard for algebraic specification languages, to development graphs has been set up. However, this translation does not cover the case of hiding yet. In this work, we extend development graphs in a way that allows also to deal with hiding. Compared with other structuring operations like union, renaming and parameterization, hiding is a particularly difficult to realize operation. This is because hiding does not admit such a good decomposition of the involved specifications as other structuring operations do.

This work is an extended version of [17]. After presenting a motivating example in Sect. 2, we recall notions for the abstract formalizations of logics and proof systems in Sect. 3. Section 4 introduces the main concept of the paper, development graphs with hiding. Proof rules are given in Sect. 5, illustrated with an example in Sect. 6 and shown to be sound and relatively complete in Sect. 7. Section 8 considers translations from various other specification formalisms to development graphs. Section 9 contains conclusions and discusses related work.

2 Motivation

As a running example consider the following example of specifying and refining a sorting function *sorter*.

Given some specification of total orders and lists, an abstract specification of this sorting function may be denoted in CASL syntax as follows:

spec SORTING [TOTALORDER] =
  {
    LIST [**sort** Elem]
  then

```
      preds is_ordered    : List[Elem];
            permutation   : List[Elem] × List[Elem];
      forall x, y         : Elem;
             L, L1, L2    : List[Elem]
      • is_ordered([])
      • is_ordered([x])
      • is_ordered(x :: (y :: L)) ⇔ x ≤ y ∧ is_ordered(y :: L)
      • permutation(L1, L2) ⇔ (∀ x : Elem • x ∈ L1 ⇔ x ∈ L2)
  then
      op sorter : List[Elem] → List[Elem];
      forall L : List[Elem]
      • is_ordered(sorter(L))
      • permutation(L, sorter(L))
  }
  hide is_ordered, permutation
end
```

is_ordered and *permutation* are auxiliary predicates to specify *sorter*, and are hidden to the outside. A model of this specification is just an interpretation of the *sorter* function (together with a model of the imported specifications of total orders and lists) that can be extended to a model of the whole specification (including *is_ordered* and *permutation*).

During a development, we may refine Sorting into a design specification describing a particular sorting algorithm. For simplicity, we choose a sorting algorithm which recursively inserts the head element in the sorted tail list. In Casl we obtain the following specification:

```
spec InsertSort [TotalOrder] =
  {
      List [sort Elem]
  then
      ops insert : Elem × List[Elem] → List[Elem];
          sorter : List[Elem] → List[Elem];
      forall x, y : Elem;
             L    : List[Elem]
```

```
    • insert(x, [])  =  [x]
    • insert(x, y :: L)  =
          x :: insert(y, L) when x ≤ y else y :: insert(x, L)
    • sorter([])  =  []
    • sorter(x :: L)  =  insert(x, sorter(L))
  }
  hide insert
end
```

Now the interesting question arises whether InsertSort is actually a refinement of Sorting; i.e. whether each InsertSort-model is also a Sorting-model.

3 Preliminaries: institutions and logics

When studying development graphs with hiding, we want to focus on the structuring and want to abstract from the details of the underlying logical system. Therefore, we recall the abstract notion of logic from Meseguer [13]. Logics consist of model theory and proof theory. Model theory is captured by the notion of *institution*, providing an abstract framework for talking about signatures, models, sentences and satisfaction. Proof theory is captured by the notion of *entailment system*, providing an abstract framework for talking about signatures, sentences and entailment.

Let $\mathcal{CAT}$ be the category of categories and functors, [1] and **Set** the category of sets and functions.

Definition 1 An *institution* [10] $\mathcal{I} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ consists of

- a category **Sign** of *signatures*,
- a functor $\mathbf{Sen}\colon \mathbf{Sign} \longrightarrow \mathbf{Set}$ giving the set of *sentences* $\mathbf{Sen}(\Sigma)$ over each signature Σ, 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 $\sigma(\varphi)$,
- a functor $\mathbf{Mod}\colon \mathbf{Sign}^{op} \longrightarrow \mathcal{CAT}$ 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

[1] Strictly speaking, $\mathcal{CAT}$ is not a category but only a so-called quasicategory, which is a category that lives in a higher set-theoretic universe.

$M'|_\sigma$,

- a satisfaction relation $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in |\mathbf{Sign}|$, [2]

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*).

Within an arbitrary but fixed institution, we can easily define the usual notion of *logical consequence* or *semantical entailment*. Given a set of Σ-sentences Γ and a Σ-sentence φ, we say that φ *follows from* Γ, written $\Gamma \models_\Sigma \varphi$, iff for all Σ-models M, we have $M \models_\Sigma \Gamma$ implies $M \models_\Sigma \varphi$. (Here, $M \models_\Sigma \Gamma$ means that $M \models_\Sigma \psi$ for each $\psi \in \Gamma$.)

A *theory* (Σ, Γ) in an institution consists of a signature Σ together with a set of sentences $\Gamma \subseteq \mathbf{Sen}(\Sigma)$. [3] The model category $\mathbf{Mod}(\Sigma, \Gamma)$ of a theory is the full subcategory of $\mathbf{Mod}(\Sigma)$ consisting of those models satisfying all of Γ.

Definition 2 An *entailment system* $\mathcal{E} = (\mathbf{Sign}, \mathbf{Sen}, \vdash)$ consists of a category **Sign** of *signatures*, a functor $\mathbf{Sen} \colon \mathbf{Sign} \longrightarrow \mathbf{Set}$ giving the set of *sentences* over a given signature, and for each $\Sigma \in |\mathbf{Sign}|$, an entailment relation $\vdash_\Sigma \subseteq |\mathbf{Sen}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ such that the following properties are satisfied:

(1) *reflexivity:* for any $\varphi \in \mathbf{Sen}(\Sigma)$, $\{\varphi\} \vdash_\Sigma \varphi$,
(2) *monotonicity:* if $\Gamma \vdash_\Sigma \varphi$ and $\Gamma' \supseteq \Gamma$ then $\Gamma' \vdash_\Sigma \varphi$,
(3) *transitivity:* if $\Gamma \vdash_\Sigma \varphi_i$, for $i \in I$, and $\Gamma \cup \{\varphi_i \mid i \in I\} \vdash_\Sigma \psi$, then $\Gamma \vdash_\Sigma \psi$,
(4) $\vdash$-*translation:* if $\Gamma \vdash_\Sigma \varphi$, then for any $\sigma \colon \Sigma \longrightarrow \Sigma'$ in **Sign**, $\sigma(\Gamma) \vdash_{\Sigma'} \sigma(\varphi)$.

Definition 3 A *logic* is a 5-tuple $\mathcal{LOG} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \vdash, \models)$ such that:

(1) $(\mathbf{Sign}, \mathbf{Sen}, \vdash)$ is an entailment system (denoted by $ent(\mathcal{LOG})$),
(2) $(\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ is an institution (denoted by $inst(\mathcal{LOG})$), and
(3) the following *soundness condition* is satisfied: for any $\Sigma \in |\mathbf{Sign}|$, $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ and $\varphi \in \mathbf{Sen}(\Sigma)$,

$$\Gamma \vdash_\Sigma \varphi \text{ implies } \Gamma \models_\Sigma \varphi.$$

A logic is *complete* if, in addition, $\Gamma \models_\Sigma \varphi$ implies $\Gamma \vdash_\Sigma \varphi$.

Throughout the rest of the paper, we will work with an arbitrary but fixed logic $\mathcal{LOG} = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \vdash, \models)$ such that **Sign** has finite colimits, and $\mathcal{LOG}$ admits finite weak amalgamation, i.e. **Mod** maps finite colimits to weak limits. A weak limit is similar to a limit; the difference being that only one

[2] $|C|$ is the class of objects of a category C.
[3] Note that the theories introduced here are *presentations* of theories. We follow here the terminology of Meseguer's general logics [13] instead of Goguen and Burstall's original definition [10]. In what follows, when we talk about a theory (Σ, Γ) we shall mean a theory presentation.

half of the universal property is imposed, namely the existence of a mediating morphism, but not its uniqueness. In order to see what this means, we need some further notion:

Given a diagram $D: I \longrightarrow \mathbf{Sign}$, let us call a family of models $(m_i)_{i\in|I|}$ *consistent with* D, if for each $i \in |I|$, $m_i \in \mathbf{Mod}(D(i))$, and for each $l: i \longrightarrow j \in I$, $m_j|_{D(l)} = m_i$.

The finite weak amalgamation property can now be reformulated as follows: $\mathcal{LOG}$ admits finite weak amalgamation iff for each finite diagram $D: I \longrightarrow \mathbf{Sign}$ and each family of models $(m_i)_{i\in|I|}$ *consistent with* D, there exists a model $m \in \mathbf{Mod}(Colim\ D)$ with $m|_{\mu_i} = m_i$, where $\mu_i: D(i) \longrightarrow \mathbf{Mod}(Colim\ D)$ are the colimit injections.

For pushouts, this boils down to the following: given a pushout diagram in **Sign**

$$\begin{tikzcd}
 & \Sigma \arrow[dl, "\sigma_1"'] \arrow[dr, "\sigma_2"] & \\
\Sigma_1 \arrow[dr, "\theta_2"] & & \Sigma_2 \arrow[dl, "\theta_1"] \\
 & \Sigma' &
\end{tikzcd}$$

a Σ_1-model M_1 and a Σ_2-model M_2 with $M_1|_{\sigma_1} = M_2|_{\sigma_2}$, there exists some Σ'-model M' with $M'|_{\theta_2} = M_1$ and $M'|_{\theta_1} = M_2$.

There are plenty of logics satisfying the above requirements, e.g. many-sorted equational logic, many-sorted first-order logic, various temporal and object-oriented logics etc. The logic underlying CASL, subsorted partial first-order logic with sort generation constraints, does not admit weak amalgamation. However, the CASL logic can be embedded into a logic with a richer signature category and a model functor admitting (weak) amalgamation [23]. Hence, the results of this paper also are applicable for CASL, if colimits are taken in the richer signature category.

4 Development graphs with hiding

A development graph, as introduced in [3], represents the actual state of a formal program development. It is used to encode the structured specifications in various phases of the development. Roughly speaking, each node of the graph represents a theory like for instance the CASL-specifications LIST, SORTING or INSERTSORT. The links of the graph define how theories can make use of other theories.

Leaves in the graph correspond to basic specifications, which do not make use of other theories (e.g. Total_order). Inner nodes correspond to structured specifications which define theories using other theories (e.g. Sorting using Total_order). The corresponding links in the graph are called *definition links*. Arising proof obligations are attached as so-called *theorem links* to this graph. We here add a new type of definition links corresponding to *hiding*.

Definition 4 A *development graph* is an acyclic, directed graph $\mathcal{S} = \langle \mathcal{N}, \mathcal{L} \rangle$.

$\mathcal{N}$ is a set of nodes. Each node $N \in \mathcal{N}$ is a tuple (Σ^N, Γ^N) such that Σ^N is a signature and $\Gamma^N \subseteq \mathbf{Sen}(\Sigma^N)$ is the set of **local axioms** of N.

$\mathcal{L}$ is a set of directed links, so-called **definition links**, between elements of $\mathcal{N}$. Global definition links import the whole subgraph below a node, while local definition links import only its local axioms. Hiding definition links are like global definition links, with the possibility to hide some symbols of the signature. Formally, each definition link from a node M to a node N is either

- **global** (denoted $M \xRightarrow{\sigma} N$), annotated with a signature morphism $\sigma : \Sigma^M \to \Sigma^N$, or
- **local** (denoted $M \xrightarrow{\sigma} N$), again annotated with a signature morphism $\sigma : \Sigma^M \to \Sigma^N$, or
- **hiding** (denoted $M \xRightarrow[h]{\sigma} N$), annotated with a signature morphism $\sigma : \Sigma^N \to \Sigma^M$ *going against the direction of the link.*

To simplify matters, we write $M \xRightarrow{\sigma} N \in \mathcal{S}$ instead of $M \xRightarrow{\sigma} N \in \mathcal{L}$ when $\mathcal{L}$ are the links of $\mathcal{S}$.

Since development graphs are acyclic, we can use induction principles in definitions and proofs concerning development graphs.

In Fig. 1 we present the development graph for the running example: The left part of the graph represents the structured specification Sorting, and the the right part the structured specification InsertSort.

$$\begin{tikzcd}
\text{List+Order} \arrow[d, Rightarrow, "c"'] \arrow[dr, Rightarrow] & \\
\text{Perm+OrderDef} \arrow[d, Rightarrow] & \text{InsertSortDef} \arrow[dd, "h"'] \\
\text{SorterProps} \arrow[d, "h"'] & \\
\text{Sorting} \arrow[r, Rightarrow, dashed, "id"] & \text{InsertSort}
\end{tikzcd}$$

Fig. 1. Development graph for the sorting example

The next definition captures the existence of a path of local and global definition links between two nodes. Notice that such a path must not contain any hiding links.

Definition 5 Let $\mathcal{S}$ be a development graph. The notion of *global reachability* is defined inductively: A node N is globally reachable from a node M via a signature morphism σ, $M \mathrel{\blacktriangleright\!\!\xRightarrow{\sigma}} N$ for short, iff

- either $M = N$ and $\sigma = id$, or
- $M \xRightarrow{\sigma'} K \in \mathcal{S}$, and $K \mathrel{\blacktriangleright\!\!\xRightarrow{\sigma''}} N$, with $\sigma = \sigma'' \circ \sigma'$.

A node N is **locally reachable** from a node M via a signature morphism σ, $M \overset{\sigma}{\rightarrowtail} N$ for short, iff $M \mathrel{\blacktriangleright\!\!\xRightarrow{\sigma}} N$ or there is a node K with $M \xrightarrow{\sigma'} K \in \mathcal{S}$ and $K \mathrel{\blacktriangleright\!\!\xRightarrow{\sigma''}} N$, such that $\sigma = \sigma'' \circ \sigma'$.

Obviously global reachability implies local reachability.

Definition 6 Let $\mathcal{S} = \langle \mathcal{N}, \mathcal{L} \rangle$ be a development graph. A node $N \in \mathcal{N}$ is *flattenable* iff for all nodes $M \in \mathcal{N}$ with incoming hiding definition links, it holds that N is not globally reachable from M.

The models of flattenable nodes do not depend on existing hiding links. All the local axioms of ingoing links (while ignoring hiding links) are collected by the *theory* of a node:

Definition 7 Let $\mathcal{S} = \langle \mathcal{N}, \mathcal{L} \rangle$ be a development graph. For $N \in \mathcal{N}$, the **theory** $Th_{\mathcal{S}}(N)$ of N is defined by

$$\Gamma^N \cup \bigcup_{K \overset{\sigma}{\rightarrowtail} N} \sigma(\Gamma^K)$$

For flattenable nodes N, $Th_{\mathcal{S}}(N)$ captures N completely. However, this is not the case for nodes that are not flattenable. Therefore, we cannot define a theory semantics of development graphs as in [3]. Rather, we have to use a model-theoretic semantics, which is compatible with the theory semantics (see Prop. 10 below).

Definition 8 Given a node $N \in \mathcal{N}$, its associated class $\mathbf{Mod}_{\mathcal{S}}(N)$ of models (or N-models for short) consists of those Σ^N-models n for which

- n satisfies the local axioms Γ^N,
- for each $K \xRightarrow{\sigma} N \in \mathcal{S}$, $n|_\sigma$ is a K-model,
- for each $K \xrightarrow{\sigma} N \in \mathcal{S}$, $n|_\sigma$ satisfies the local axioms Γ^K, and
- for each $K \xRightarrow[h]{\sigma} N \in \mathcal{S}$, n has a σ-expansion k (i.e. $k|_\sigma = n$) that is a K-model.

This definition of model classes nicely interacts with reachability:

Proposition 9 (1) If $M \mathrel{\blacktriangleright\!\!\xRightarrow{\sigma}} N$ and $n \in \mathbf{Mod}(N)$, then $n|_\sigma \in \mathbf{Mod}(M)$.
(2) If $M \overset{\sigma}{\rightarrowtail} N$ and $n \in \mathbf{Mod}(N)$, then $n|_\sigma \models \Gamma^M$.

PROOF. 1. Easy induction over the definition of global reachability.

2. By 1 and Definition 8.

Proposition 10 (1) $\mathbf{Mod}(N) \subseteq \mathbf{Mod}(\mathit{Th}_{\mathcal{S}}(N))$.
(2) If N is flattenable, then $\mathbf{Mod}(N) = \mathbf{Mod}(\mathit{Th}_{\mathcal{S}}(N))$.

PROOF. 1. Let $n \in \mathbf{Mod}(N)$. By Proposition 9(2), for each $M \overset{\sigma}{\rightarrowtail} N$, $n|_\sigma \models \Gamma^M$, hence $n \models \sigma(\Gamma^M)$ by the satisfaction condition. Moreover, by Def. 8, $n \models \Gamma^N$. Hence, $n \in \mathbf{Mod}(\mathit{Th}_{\mathcal{S}}(N))$.

2. By 1, it suffices to prove the "$\supseteq$" direction. Let n be an $\mathit{Th}_{\mathcal{S}}(N)$-model. Let $\mathit{len}(p)$ be the length of a path p witnessing $M \overset{\tau}{\blacktriangleright\!\!\Longrightarrow} N$. Let maxp the maximal such length in $\mathcal{S}$ (for the given N). We show that for any $M \overset{\tau}{\blacktriangleright\!\!\Longrightarrow} N$, $n|_\tau$ is an M-model. We proceed by induction over $\mathit{maxp} - \mathit{len}(p)$ with p witnessing $M \overset{\tau}{\blacktriangleright\!\!\Longrightarrow} N$. Since N is flattenable, we only have to show clauses 1 to 3 of Definition 8:

(1) Since global implies local reachability, $M \overset{\tau}{\rightarrowtail} N$, and $\tau(\Gamma^M) \subseteq \mathit{Th}_{\mathcal{S}}(N)$; hence $n \models \tau(\Gamma^M)$. By the satisfaction condition for institutions, $n|_\tau \models \Gamma^M$.
(2) Let $K \overset{\theta}{\Longrightarrow} M$, hence $K \overset{\tau\circ\theta}{\blacktriangleright\!\!\Longrightarrow} N$. By the induction hypothesis, $n|_{\tau\circ\theta} = n|_\tau|_\theta$ is a K-model.
(3) Let $K \overset{\theta}{\longrightarrow} M$, hence $K \overset{\tau\circ\theta}{\rightarrowtail} N$. With a similar argument as for 1, we get $n|_{\tau\circ\theta} = n|_\tau|_\theta \models \Gamma^K$.

This completes the induction. Since $N \overset{id}{\blacktriangleright\!\!\Longrightarrow} N$, n is an N-model. $\square$

Definition 11 $\mathcal{DG}_1 = \langle \mathcal{N}_1, \mathcal{L}_1 \rangle$ is a *subgraph* of $\mathcal{DG}_2 = \langle \mathcal{N}_2, \mathcal{L}_2 \rangle$ if $\mathcal{N}_1 \subseteq \mathcal{N}_2$ and $\mathcal{L}_1 \subseteq \mathcal{L}_2$. It is a *faithful subgraph*, if all links in $\mathcal{L}_2 \setminus \mathcal{L}_1$ have target nodes in $\mathcal{N}_2 \setminus \mathcal{N}_1$. Also, in this case $\mathcal{DG}_2$ is called a *faithful supergraph* of $\mathcal{DG}_1$.

Model classes do not change when passing to faithful supergraphs:

Proposition 12 If $\mathcal{DG}_1$ is a faithful subgraph of $\mathcal{DG}_2$ and N a node in $\mathcal{DG}_1$, then

$$\mathbf{Mod}_{\mathcal{DG}_1}(N) = \mathbf{Mod}_{\mathcal{DG}_2}(N).$$

PROOF. The notion of N-model only depends on the local axioms of N and definition links going into N. Both do not change when passing to a faithful supergraph. $\square$

Complementary to definition links, which *define* the theories of related nodes, we introduce the notion of a *theorem link* with the help of which we are able to *postulate* relations between different theories. Theorem links are the central data structure to represent proof obligations arising in formal developments. Again we distinguish between local and global theorem links (denoted by $N \overset{\sigma}{\dashrightarrow\!\!\!\blacktriangleright} M$ and $N \overset{\sigma}{\dashrightarrow} M$ respectively). Moreover, we introduce *local implications* of form $N \Rightarrow \Gamma$, where Γ is a set of Σ^N-sentences. $N \Rightarrow \{\varphi\}$ also is written $N \Rightarrow \varphi$. Finally, we also need theorem links $N \xRightarrow[\theta\ h]{\sigma} M$ (where for some Σ, $\theta\colon \Sigma \longrightarrow \Sigma^N$ and $\sigma\colon \Sigma \longrightarrow \Sigma^M$) involving hiding.

The semantics of theorem links is given by the next definition.

Definition 13 Let $\mathcal{S}$ be a development graph and N, M nodes in $\mathcal{S}$.

$\mathcal{S}$ **satisfies** a global theorem link $N \overset{\sigma}{\dashrightarrow\!\!\!\blacktriangleright} M$ (denoted $\mathcal{S} \models N \overset{\sigma}{\dashrightarrow\!\!\!\blacktriangleright} M$) iff for all $m \in \mathbf{Mod}_{\mathcal{S}}(M)$, $m|_\sigma \in \mathbf{Mod}_{\mathcal{S}}(N)$.

$\mathcal{S}$ **satisfies** a local theorem link $N \overset{\sigma}{\dashrightarrow} M$ (denoted $\mathcal{S} \models N \overset{\sigma}{\dashrightarrow} M$) iff for all $m \in \mathbf{Mod}_{\mathcal{S}}(M)$, $m|_\sigma \models \Gamma^N$.

$\mathcal{S}$ **satisfies** a local implication $N \Rightarrow \Gamma$, written $\mathcal{S} \models N \Rightarrow \Gamma$, if for all $n \in \mathbf{Mod}_{\mathcal{S}}(N)$, $n \models \Gamma$.

$\mathcal{S}$ **satisfies** a hiding theorem link $N \xRightarrow[\theta\ h]{\sigma} M$ (denoted $\mathcal{S} \models N \xRightarrow[\theta\ h]{\sigma} M$) iff for all $m \in \mathbf{Mod}_{\mathcal{S}}(M)$, $m|_\sigma$ has a θ-expansion to some N-model.

E.g. consider the development graph of the running example (cf. Fig. 1): The theorem link from SORTING to INSERTSORT expresses the postulation that the latter is a refinement of the former. Furthermore, common proof obligations in a formal development can be encoded into properties that specific global theorem links are implied by the actual development graph.

A global definition link $M \xRightarrow{\sigma} N$ in a development graph is a *conservative extension*, if every M-model can be expanded along σ to an N-model. We will allow to annotate a global definition link as $M \xRightarrow[c]{\sigma} N$, which shall express that it is a conservative extension. These annotations can be seen as another kind of proof obligations.

5 Rules for development graphs with hiding

The rules for theorem proving in development graphs given in [3] allow to decompose a global theorem link into local theorem links. Unfortunately, it is not possible to decompose global theorem links starting from nodes with hiding definition links going (directly or indirectly) into them. This is because hiding is some kind of existential quantification, and in general it is not possible to decompose an existential quantification of a conjunction into existential quantifications of the conjuncts.

We therefore have to extend the set of rules from [3] to deal with hiding. We have two kinds of rules:

(1) *Rules for hiding and conservative extension.* These rules are suited to push theorem links along the hidings inside the development graph, such that they eventually can be decomposed into local theorem links.
(2) *Decomposition rules* from [3]. They allow to split global theorem links into local and hiding theorem links.

5.1 Rules for hiding and conservative extension

We now come to the rules for hiding and conservative extension. We introduce two rules to shift theorem links over hiding, one dealing with hiding links on the left hand side of a theorem link, and the other one with hiding links on the right hand side of a theorem link.

Since the first rule is quite powerful, we need some preliminary notions. Given a node N in a development graph $\mathcal{S} = \langle \mathcal{N}, \mathcal{L} \rangle$, the idea is that we unfold the subgraph below N into a tree and form a diagram with this tree. More formally, define the *diagram* $D\colon I \longrightarrow \mathbf{Sign}$ *associated with* N together with a map $G\colon |I| \longrightarrow \mathcal{N}$ inductively as follows:

- $\langle N \rangle$ is an object in I, with $D(\langle N \rangle) = \Sigma^N$. Let $G(\langle N \rangle)$ be just N.
- if $i = \langle\, M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N \,\rangle$ is an object in I with $l_1, \ldots, l_n$ non-local links in $\mathcal{L}$, and $l = K \overset{\sigma}{\Longrightarrow} M$ is a (global or local) definition link in $\mathcal{L}$, then
$$j = \langle\, K \xrightarrow{l} M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N \,\rangle$$
is an object in I with $D(j) = \Sigma^K$, and l is a morphism from j to i in I with $D(l) = \sigma$. We set $G(j) = K$.
- if $i = \langle\, M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N \,\rangle$ is an object in I with $l_1, \ldots, l_n$ non-local links in $\mathcal{L}$, and $l = K \underset{h}{\overset{\sigma}{\Longrightarrow}} M$ is a hiding link in $\mathcal{L}$, then
$$j = \langle\, K \xrightarrow{l} M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N \,\rangle$$
is an object in I with $D(j) = \Sigma^K$, and l is a morphism from i to j in I with $D(l) = \sigma$. We set $G(j) = K$.

This means that the graph is just unfolded to the diagram. The unfolding is necessary to achieve that in the diagram there is a distinction between instances of the same node that are imported via different paths into another node.

5.1.0.1 Theorem-Hide-Shift.

This rule (cf. Fig. 2) is used if a hiding link occurs on the right-hand side of a theorem link. For this rule $D: I \longrightarrow$ **Sign** is the diagram associated with N, $\mu_i: D(i) \longrightarrow Colim\ D$ are the colimit injections ($i \in |I|$), C is a new isolated node with signature $Colim\ D$, and with ingoing local definition links $G(i) \xRightarrow{\mu_i} C$ for $i \in |I|$. Here, an isolated node is one with no local axioms and no ingoing definition links other than those shown in the rule

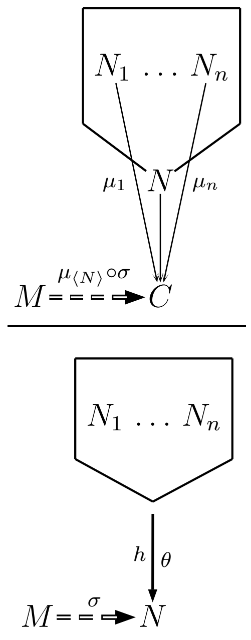

Fig. 2. Rule *Theorem-Hide-Shift*

We now illustrate why the unfolding of the subgraph under N in the rule *Theorem-Hide-Shift* is needed. Consider the development graph defining groups with the help of groups with inverse (by hiding the inverse) and then defining fields using groups twice: both for the additive and the multiplicative group.

$$\begin{tikzcd}
\text{GroupInv} \arrow[dr, "\iota", "h"'] & \text{Field} \\
& \text{Group} \arrow[u, Rightarrow, shift right, "\sigma_1"] \arrow[u, Rightarrow, shift left, "\sigma_2"']
\end{tikzcd}$$

If we would take the colimit of this graph, we would identify the additive with the multiplicative group in *Field*.

This is not what we want. The unfolding of the rule *Theorem-Hide-Shift* now doubles the signature of groups and groups with inverse, leading to a signature *Colim D* containing the additive and the multiplicative group, and an additive and a multiplicative inverse. The graph for the premise of the rule is then

$$\begin{tikzcd}
\text{GroupInv} \arrow[r, Rightarrow, shift left, "\iota_1"] \arrow[r, Rightarrow, shift right, "\iota_2"'] \arrow[dr, "\iota", "h"'] & \text{C} & \\
& \text{Group} \arrow[u, Rightarrow, shift right, "\theta_1"] \arrow[u, Rightarrow, shift left, "\theta_2"'] \arrow[r, Rightarrow, shift left, "\sigma_1"] \arrow[r, Rightarrow, shift right, "\sigma_2"'] & \text{Field} \arrow[ul, Rightarrow, "\theta"']
\end{tikzcd}$$

In the node C, one can reason about both inverses in parallel, while the theory of groups with inverse is not doubled (as it would be the case with approaches that flatten specifications).

5.1.0.2 Hide-Theorem-Shift.

This rule (cf. Fig. 3) replaces hiding theorem links by normal theorem links. This is only possible if on the right-hand side of the hiding theorem link, a conservative definition link occurs, and furthermore $\sigma' \circ \theta = \theta' \circ \sigma$.

$$\begin{tikzcd} & N' \\ M' \arrow[ur, Rightarrow, dashed, "\sigma'"] & N \arrow[u, Rightarrow, "\theta'", "c"'] \end{tikzcd}$$

$$\begin{tikzcd} & N' \\ M' \arrow[r, dashed, "\sigma", "h\ \theta"'] & N \arrow[u, Rightarrow, "\theta'", "c"'] \end{tikzcd}$$

Fig. 3. Rule *Hide-Theorem-Shift*

5.1.0.3 Cons-Shift.

The previous rule that allows for replacing hiding theorem links requires conservative definition links. In order to be able to derive new conservative definition links from existing ones, we introduce a rule which allows their derivation. For this rule (cf. Fig. 4) we must require that

$$\begin{tikzcd} \Sigma^M \arrow[r, "\sigma"] \arrow[d, "\theta"] & \Sigma^N \arrow[d, "\theta'"] \\ \Sigma^{M'} \arrow[r, "\sigma'"] & \Sigma^{N'} \end{tikzcd}$$

is a pushout, and moreover, that N' is isolated.

$$\begin{tikzcd} M \arrow[r, Rightarrow, "\sigma"] \arrow[d, Rightarrow, "c"', "\theta"] & N \\ M' & \end{tikzcd}$$

$$\begin{tikzcd} M \arrow[r, Rightarrow, "\sigma"] \arrow[d, Rightarrow, "c"', "\theta"] & N \arrow[d, Rightarrow, "c"', "\theta'"] \\ M' \arrow[r, Rightarrow, "\sigma'"] & N' \end{tikzcd}$$

Fig. 4. Rule *Cons-shift*

In addition to the above rules, one would use logic-specific rules for syntactically determining conservative extensions (e.g. extensions by definitions).

Proposition 14 The rules for hiding and conservative extension are sound, i.e. for any theorem link or conservativity annotation L, if $\mathcal{S} \vdash L$, then $\mathcal{S} \models L$.

PROOF. By induction over the proof of $\mathcal{S} \vdash L$. That is, for a rule with premises $L_1, \dots L_n$ and conclusion L, we need to show that $\mathcal{S} \models L_1$ and ... and $\mathcal{S} \models L_n$ implies $\mathcal{S} \models L$.

Theorem-Hide-Shift: Assume that $\mathcal{S} \models M \overset{\mu_{\langle N\rangle}\circ\sigma}{\Rightarrow} C$. Let n be an N-model. We have to show $n|_\sigma$ to be an M-model in order to establish the holding of $M \overset{\sigma}{\Rightarrow} N$. We inductively define a family $(m_i)_{i\in|I|}$ of models $m_i \in \mathbf{Mod}(G(i))$ by putting

- $m_{\langle N\rangle} := n$,

- $m_{\langle K \xrightarrow{l} M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N\rangle} := m|_\sigma$, where $l = K \xRightarrow{\sigma} M$ and $m = m_{\langle M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N\rangle}$, and
- $m_{\langle K \xrightarrow{l} M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N\rangle}$ is a σ-expansion of m to a K-model, where $l = K \xRightarrow[h]{\sigma} M$ and $m = m_{\langle M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N\rangle}$.

It is easy to show that this family is consistent with D. By weak amalgamation, there is a $\Sigma^C = \mathit{Colim}\ D$-model c with $c|_{\mu_i} = m_i$. The latter implies that c is a C-model. By the assumption, $c|_{\mu_{\langle N\rangle}\circ\sigma} = m_{\langle N\rangle}|_\sigma = n|_\sigma$ is an M-model.

Hide-Theorem-shift: Assume that $\mathcal{S} \models M' \overset{\sigma'}{=\!=\!\Rightarrow} N'$ and $N \xRightarrow[c]{\theta'} N'$ is conservative. We have to show that $\mathcal{S} \models M' \xRightarrow[\theta\ h]{\sigma} N$. Let n be an N-model. Since $N \xRightarrow[c]{\theta'} N'$ is conservative, n can be expanded to an N'-model n' with $n'|_{\theta'} = n$. By the assumption, $n'|_{\sigma'}$ is an M'-model. Thus, $n'|_{\sigma'\circ\theta} = n'|_{\theta'\circ\sigma} = n|_\sigma$ has a θ-expansion to an M'-model.

Cons-shift: Assume that $M \xRightarrow[c]{\theta} M'$ is conservative. We have to prove that $N \xRightarrow[c]{\theta'} N'$ is conservative as well. Let n be an N-model. Since $M \xRightarrow{\theta} M'$ is conservative, $n|_\sigma$ has a θ-expansion m' being an M'-model. By weak amalgamation, there is some $\Sigma^{N'}$-model n' with $n'|_{\sigma'} = m'$ and $n'|_{\theta'} = n$. Since N' is isolated, n' is an N'-model.

5.2 *Rules for decomposition*

The rules for decomposition are taken mostly from [3]. The rule *Glob-Decomposition* has to be changed. It now decomposes a global theorem link from N to M into local theorem links into M from those nodes from which N is reachable and into hiding theorem links into M from those nodes which are the source of a hiding link going into some node from which N is reachable.

Glob-Decomposition:

$$\frac{\bigcup_{K \overset{\sigma'}{\rightarrowtail} N}\{K \overset{\sigma\circ\sigma'}{\dashrightarrow} M\} \cup \bigcup_{L \xRightarrow[h]{\theta} K \text{ and } K \blacktriangleright\!\!\xRightarrow{\sigma'} N}\{L \xRightarrow[\theta\ h]{\sigma\circ\sigma'} M\}}{N \overset{\sigma}{=\!=\!\Rightarrow} M}$$

Subsumption:

$$\frac{N \blacktriangleright\!\!\xRightarrow{\sigma} M}{N \overset{\sigma}{=\!=\!\Rightarrow} M}$$

Loc-Decomposition I:

$$\frac{K \overset{\sigma}{\dashrightarrow} L}{K \overset{\sigma''}{\dashrightarrow} M} \quad \text{if } L \blacktriangleright\!\!\xRightarrow{\sigma'} M \text{ and } \sigma''(\Gamma^K) = \sigma'(\sigma(\Gamma^K))$$

Loc-Decomposition II:

$$\frac{}{N \overset{\sigma}{\dashrightarrow} M} \text{ if } N \overset{\sigma'}{\rightarrowtail} M \text{ and } \sigma(\Gamma^N) = \sigma'(\Gamma^N)$$

Local Inference:

$$\frac{N \Rightarrow \sigma(\Gamma^M)}{M \overset{\sigma}{\dashrightarrow} N}$$

Basic Inference:

$$\frac{Th_{\mathcal{S}}(N) \vdash_{\Sigma^N} \varphi \text{ for each } \varphi \in \Gamma}{N \Rightarrow \Gamma}$$

Given a development graph containing some theorem links that have to be proved, the intention is that the above rules will be applied in a backwards manner as long as possible. The rules for hiding allow to shift the (global) theorem links to nodes without hiding involved, while the rules for decomposition allow to decompose the global theorem links into local ones. Then, with *Local Inference*, local theorem links can be reduced to local implications, which in turn can be proved with *Basic Inference* using the entailment relation of the base logic $\mathcal{LOG}$.

Proposition 15 The above rules for decomposition are sound.

PROOF.

Glob-Decomposition: Assume that

(1) $\mathcal{S} \models K \overset{\sigma\circ\tau}{\dashrightarrow} M$ for each $K \overset{\tau}{\rightarrowtail} N$,

(2) $\mathcal{S} \models L \overset{\sigma\circ\tau}{\underset{\theta\ h}{\Longrightarrow}} M$ for each $L \overset{\theta}{\underset{h}{\Longrightarrow}} K$ and $K \overset{\tau}{\blacktriangleright\!\!\Longrightarrow} N$.

In order to show $\mathcal{S} \models N \overset{\sigma}{\Longrightarrow} M$, let m be an M-model. Let *len*(p) be the length of a path p witnessing $K \overset{\tau}{\blacktriangleright\!\!\Longrightarrow} N$. Let *maxp* the maximal such length in S. We show that for any $K \overset{\tau}{\blacktriangleright\!\!\Longrightarrow} N$, $m|_{\sigma\circ\tau}$ is a K-model. We proceed by induction over $maxp - len(p)$ for p witnessing $K \overset{\tau}{\blacktriangleright\!\!\Longrightarrow} N$. We have to show clauses 1 to 4 of Definition 8:

(1) By the first assumption, $m|_{\sigma\circ\tau} \models \Gamma^K$.

(2) By the induction hypothesis, $m|_{\sigma\circ\tau}$ satisfies any global definition link going into K.

(3) By the first assumption, $m|_{\sigma\circ\tau}$ satisfies any local definition link going into K.

(4) By the second assumption, $m|_{\sigma\circ\tau}$ satisfies any hiding definition link going into K.

This completes the induction. Since $N \overset{id}{\blacktriangleright\!\!\Longrightarrow} N$, $m|_\sigma$ is an N-model.

Subsumption: Obvious.

Loc-Decomposition I: Assume $\mathcal{S} \models K \overset{\sigma}{\dashrightarrow} L$ and $L \overset{\theta}{\Longrightarrow} M$ and $\tau(\Gamma^K) = \theta(\sigma(\Gamma^K))$. In order to show $\mathcal{S} \models K \overset{\tau}{\dashrightarrow} M$, let m be an M-model. By Proposition 9, $m|_\theta$ is an L-model, and by the assumption, $m|_{\theta\circ\sigma} \models \Gamma^K$. By the satisfaction condition for institutions, $m \models \theta \circ \sigma(\Gamma^K) = \tau(\Gamma^K)$. Again by the satisfaction condition, $m|_\tau \models \Gamma^K$.

Loc-Decomposition II: Assume that $M \overset{\theta}{\rightarrowtail} N$ and $\sigma(\Gamma^M) = \theta(\Gamma^M)$. Let n be an N-model. By Proposition 9, $n|_\theta \models \Gamma^M$. By the satisfaction condition for institutions, $n \models \theta(\Gamma^M) = \sigma(\Gamma^M)$. Again by the satisfaction condition, $n|_\sigma \models \Gamma^M$.

Local Inference: Assume that $n \models \sigma(\Gamma^M)$ for each N-model n. In order to show $\mathcal{S} \models M \overset{\sigma}{\dashrightarrow} N$, let n be an N-model. By assumption, $n \models \sigma(\Gamma^M)$. By the satisfaction condition for institutions, $n|_\sigma \models \Gamma^M$.

Basic Inference: Assume that $Th_{\mathcal{S}}(N) \vdash_{\Sigma^N} \varphi$ for each $\varphi \in \Gamma$. By soundness of $\vdash_{\Sigma^N}$, we get $Th_{\mathcal{S}}(N) \models_{\Sigma^N} \Gamma$. In order to show $\mathcal{S} \models N \Rightarrow \Gamma$, let n be an N-model. By Proposition 10, $n \models Th_{\mathcal{S}}(N)$. Since $Th_{\mathcal{S}}(N) \models_{\Sigma^N} \Gamma$, also $n \models \Gamma$.

6 Example

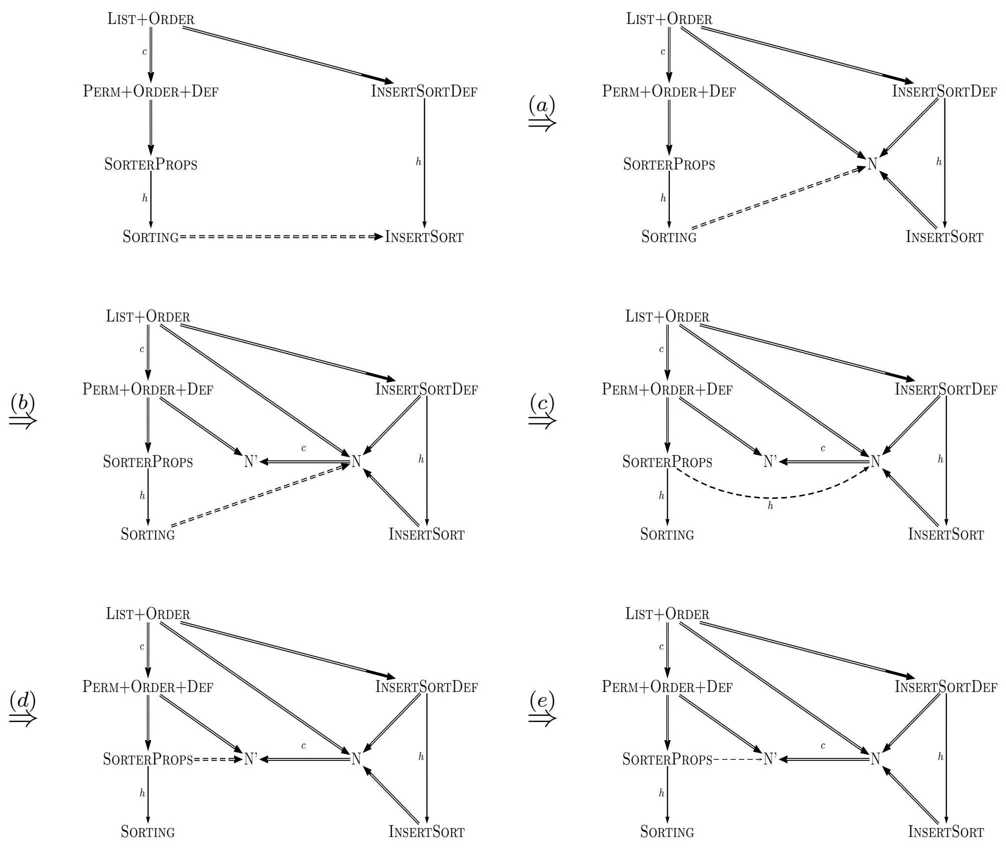

Fig. 5. Reduction of theorem links in the running example.

We now demonstrate the (backward manner) use of the rules with the example development graph from Sect. 4. The goal is to reduce the theorem link between Sorting and InsertSort to theorem links between flattenable nodes. The derivation is shown in Fig. 5. In the first step (a) the *Theorem-Hide-Shift* rule is applied, which introduces the new node N and the new global definition links. In the second step (b), we infer conservative relationships by applying the rule *Cons-Shift*. This introduces the new node N' and the respective global definition links. Now the theorem link can be reduced to a hiding theorem link from SorterProps to N by *Glob-Decomposition* (step (c)). Now, this hiding theorem link can be reduced to the theorem link between SorterProps and N' using the rule *Hide-Theorem-Shift* (step (d)). Finally, using *Glob-Decomposition* again, we get three local theorem links, two of which can be immediately discarded with *Subsumption* (step (e)). The remaining local theorem link can then be proved by reasoning in the logic (via *Local Inference* and *Basic Inference*).

7 Results about completeness

The soundness of our rules is established by Prop. 14 and 15. Another question is the completeness of our rules. We have the following counterexample:

Proposition 16 Let FOL be the usual first-order logic with a recursively axiomatized complete entailment system. Solving the question whether a global theorem link holds in a development graph with hiding over FOL is not recursively enumerable. Thus, any recursively axiomatized calculus for development graphs with hiding is incomplete.

PROOF. This can be seen as follows. Let Σ be the FOL-signature with a sort *nat* and operations for zero and successor, addition and multiplication and take the usual second-order Peano axioms characterizing the natural numbers uniquely up to isomorphism,

plus the defining axioms for addition and multiplication.

Without loss of generality, we can assume that these axioms are combined into a single axiom of the form

$$\forall P\colon pred(nat)\ .\ \varphi$$

where φ is a first-order formula. Let ψ be any sentence over Σ. Let $\theta\colon \Sigma \longrightarrow \Sigma'$ add a predicate $P : \mathit{pred}(\mathit{nat})$ to Σ. Consider the development graph

$$\begin{tikzcd}
\textsc{Peano} \arrow[d, Rightarrow, dashed, "id"] & \textsc{PeanoDef} \arrow[l, "h"', "\theta"] \\
\Sigma &
\end{tikzcd}$$

where Σ and $\textsc{Peano}$ are nodes with signature Σ and no local axioms, whereas $\textsc{PeanoDef}$ is a node with signature Σ' and local axiom $\varphi \Rightarrow \psi$.

Now we have that $\textsc{Peano} \xRightarrow{id} \Sigma$ holds iff each Σ-model has a $\textsc{PeanoDef}$-expansion. It is easy to see that this holds iff the second-order formula $\exists P : \mathit{pred}(\mathit{nat})\,.\,\varphi \Rightarrow \psi$ is valid. By the quantifier rules for prenex normal form, this is equivalent to $\forall P : \mathit{pred}(\mathit{nat})\,.\,\varphi \models \psi$, i.e. equivalent to the fact that ψ holds in the second-order axiomatization of Peano arithmetic. By Gödels incompleteness theorem, this question is not recursively enumerable. $\square$

In spite of this negative result, there is still the question of a relative completeness w.r.t. a given oracle deciding conservative extensions. Such a completeness result has been proved by Borzyszkowski [6] in a similar setting. We are going to prove an analogous result here. We first need a preparatory lemma:

Lemma 17 If C and N are as in the rule *Theorem-Hide-Shift*, then $\mathcal{S} \models N \xRightarrow{\mu_{\langle N \rangle}} C$.

PROOF. We use the notation introduced in connection with the construction of C in the rule *Theorem-Hide-Shift*. For $i \in |I|$, let $\mathit{len}(i)$ be the length of the path i, and let p be the maximum of all $\mathit{len}(i)$, $i \in |I|$. We prove by induction over $p - \mathit{len}(i)$ that for all C-models c and all paths $i \in |I|$ containing no local definition link, $c|_{\mu_i}$ is a $G(i)$-model. Since $G(\langle N \rangle) = N$, the result then follows. Let $i = \langle\, M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N \,\rangle \in |I|$ be a path containing no local definition link. By induction hypothesis, for each ingoing non-local link $K \xRightarrow{l} M$, $c|_{\mu_j}$ is a K-model for $j = \langle\, K \xrightarrow{l} M \xrightarrow{l_1} \cdots \xrightarrow{l_n} N \,\rangle$. Now

- if $l = K \xrightarrow{\sigma} M$, $c|_{\mu_i \circ \sigma} = c|_{\mu_j}$, and since $K = G(j) \xrightarrow{\mu_j} C \in S$, $c|_{\mu_j} \models \Sigma^K$;
- if $l = K \xRightarrow{\sigma} M$, $c|_{\mu_i \circ \sigma} = c|_{\mu_j}$, and $c|_{\mu_i}$ σ-reduces to a K-model;
- if $l = K \xRightarrow[h]{\sigma} M$, $c|_{\mu_j \circ \sigma} = c|_{\mu_i}$, and $c|_{\mu_i}$ σ-expands to a K-model.

Hence in all cases, the link l is satisfied by $c|_{\mu_i}$. Since $M = G(i) \xrightarrow{\mu_i} C$, $c|_{\mu_i}$ also satisfies the local axioms in M. Hence, $c|_{\mu_i}$ is a model of $M = G(i)$. $\square$

Theorem 18 Assume that the underlying logic $\mathcal{LOG}$ is complete. Then the rule system for development graphs with hiding is complete relative to an oracle for conservative extensions.

PROOF. Assume $\mathcal{S} \models M \overset{\sigma}{\dashrightarrow} N$. We show that there is some faithful extension $\mathcal{S}_1$ of $\mathcal{S}$ (i.e. new nodes and new definition links are added, but the latter go only into new nodes) such that $\mathcal{S}_1 \vdash M \overset{\sigma}{\dashrightarrow} N$.

Let $D\colon I \longrightarrow \mathbf{Sign}$ and C be as in the rule *Theorem-Hide-Shift*, and let c be a Σ^C-model satisfying $Th_{\mathcal{S}}(C)$. By the construction of C, C is flattenable. By Proposition 10, c is a C-model. By Lemma 17, $c|_{\mu_{\langle N\rangle}}$ is an N-model, and by the assumption $\mathcal{S} \models M \overset{\sigma}{\dashrightarrow} N$, $c|_{\mu_{\langle N\rangle}\circ\sigma}$ is an M-model. We now have:

(1) for any $K \overset{\theta}{\rightarrowtail} M$, $c|_{\mu_{\langle N\rangle}\circ\sigma\circ\theta} \models \Gamma^K$ by Proposition 9. By the satisfaction condition for institutions, we get $c \models \mu_{\langle N\rangle}(\sigma(\theta(\Gamma^K)))$. Since c has been an arbitrary $Th_{\mathcal{S}}(C)$-model, $Th_{\mathcal{S}}(C) \models \mu_{\langle N\rangle}(\sigma(\theta(\Gamma^K)))$. By completeness of the underlying logic, we get $Th_{\mathcal{S}}(C) \vdash \mu_{\langle N\rangle}(\sigma(\theta(\Gamma^K)))$. By *Basic Inference*, $\mathcal{S} \vdash C \Rightarrow \mu_{\langle N\rangle}(\sigma(\theta(\Gamma^K)))$. By *Local Inference*, $\mathcal{S} \vdash K \overset{\mu_{\langle N\rangle}\circ\sigma\circ\theta}{\dashrightarrow} C$.

(2) for any $K \overset{\theta}{\Rightarrow} M$ and $L \underset{h}{\overset{\tau}{\Rightarrow}} K$, take a pushout

$$\begin{tikzcd}
\Sigma^K \arrow[r, "\mu_{\langle N\rangle}\circ\sigma\circ\theta"] \arrow[d, "\tau"] & \Sigma^C \arrow[d, "\tau'"] \\
\Sigma^L \arrow[r, "\mu'"] & \Sigma'
\end{tikzcd}$$

and obtain a new development graph $\mathcal{S}'$ from $\mathcal{S}$ by introducing a new node L' with signature Σ' and two ingoing definition links $L \overset{\mu'}{\Rightarrow} L'$ and $C \overset{\tau'}{\Rightarrow} L'$. The latter link is conservative: for any C-model c_1, with an argument as above, $c_1|_{\mu_{\langle N\rangle}\circ\sigma\circ\theta}$ is a K-model, and hence has a τ-expansion to an L-model c_2, and by weak amalgamation, there is some Σ'-model c_3 with $c_3|_{\tau'} = c_1$ and $c_3|_{\mu'} = c_2$, which is hence an L'-model. By the oracle for conservativity, we get $C \underset{c}{\overset{\tau'}{\Rightarrow}} L'$. Now $\mathcal{S}' \vdash L \overset{\mu'}{\dashrightarrow} L'$ by *Subsumption.* By *Hide-Theorem-Shift*, we get $\mathcal{S}' \vdash L \underset{\tau\ h}{\overset{\mu_{\langle N\rangle}\circ\sigma\circ\theta}{\Rightarrow}} C$.

Let $\mathcal{S}_1$ be the union of all the $\mathcal{S}'$ constructed in step 2 above (assuming that all the added nodes are distinct). By *Glob-Decomposition*, we get $\mathcal{S}_1 \vdash M \overset{\mu_{\langle N\rangle}\circ\sigma}{\dashrightarrow} C$. By *Theorem-Hide-Shift*, we get $\mathcal{S}_1 \vdash M \overset{\sigma}{\dashrightarrow} N$. □

Corollary 19 If $\mathcal{LOG}$ is complete, the decomposition rules are complete for proving theorem links between flattenable nodes.

PROOF. By inspecting the proof of Theorem 18, one can see that for theorem links between flattenable nodes, case 2 is never entered, and thus neither the rules *Cons-Shift* and *Hide-Theorem-Shift* nor the oracle for conservativeness are needed. Moreover, one can replace the node C in the proof of Theorem 18 with the node N, thus also avoiding the use of *Theorem-Hide-Shift*. □

8 Translation from specification languages to development graphs

In this section, we show that several institution independent specification formalisms that have been studied in the literature [21,19,20] can be translated to development graphs. This underpins the intended role of development graphs as a kernel formalism for management of structured proofs and of change. Since we provide an institution independent formalism, we do not treat institution dependent systems such a the module algebra of [5] here (however, see [20] for the relation of module algebra to institution independent structuring).

8.1 Structured specifications

We start our investigations by translating the kernel language for structured specifications to development graphs in an arbitrary institution proposed in [21] into our notion of development graphs. The language defines, simultaneously with the notion of specification, functions **Sig** and **Mod** yielding the signature and the model class of a specification.

***presentations*:** For any signature $\Sigma \in |\mathbf{Sign}|$ and finite set $\Gamma \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences, the *presentation* $\langle\Sigma, \Gamma\rangle$ is a specification with:

$$\mathbf{Sig}(\langle\Sigma,\Gamma\rangle) := \Sigma$$

$$\mathbf{Mod}(\langle\Sigma,\Gamma\rangle) := \{M \in \mathbf{Mod}(\Sigma) \mid M \models \Gamma\}$$

***union*:** For any signature $\Sigma \in |\mathbf{Sign}|$, given Σ-specifications SP_1 and SP_2, their *union* $SP_1 \cup SP_2$ is a specification with:

$$\mathbf{Sig}(SP_1 \cup SP_2) := \Sigma$$

$$\mathbf{Mod}(SP_1 \cup SP_2) := \mathbf{Mod}(SP_1) \cap \mathbf{Mod}(SP_2)$$

***translation*:** For any signature morphism $\sigma\colon \Sigma \longrightarrow \Sigma'$ and Σ-specification SP, **translate** SP **by** σ is a specification with:

$$\mathbf{Sig}(\textbf{translate } SP \textbf{ by } \sigma) := \Sigma'$$

$$\mathbf{Mod}(\textbf{translate } SP \textbf{ by } \sigma) := \{M' \in \mathbf{Mod}(\Sigma') \mid M'|_\sigma \in \mathbf{Mod}(SP)\}$$

***hiding*:** For any signature morphism $\sigma\colon \Sigma \longrightarrow \Sigma'$ and Σ'-specification SP', **derive from** SP' **by** σ is a specification with:

$$\begin{aligned}\mathbf{Sig}(\textbf{derive from } SP' \textbf{ by } \sigma) &:= \Sigma \\ \mathbf{Mod}(\textbf{derive from } SP' \textbf{ by } \sigma) &:= \{M'|_\sigma \mid M' \in \mathbf{Mod}(SP')\}\end{aligned}$$ [4]

To define the translation of structured specification in [21] we use a judgement $SP \rhd_S \mathcal{S}, O$, where SP is a structured specification, $\mathcal{S}$ is a development graph and O a node corresponding to SP (note that $\mathcal{S}$ may contain further nodes corresponding to parts of SP). Simultaneously with the definition, by induction we show that

$$\mathbf{Mod}(SP) = \mathbf{Mod}_{\mathcal{S}}(O)$$

8.1.1 Basic Specifications

According to Def. 8 we can represent a basic specifications as a single node consisting of Γ as local axioms:

$$\frac{}{\langle \Sigma, \Gamma \rangle \rhd_S \langle \{N\}, \emptyset \rangle, N}$$

where N is the theory with local signature Σ and local axioms Γ.

8.1.2 Union:

In order to represent unions of specifications, we add a new node with empty set of axioms and import the translations of the united specifications via global definition links annotated with the identity morphism *id*.

$$\frac{\begin{array}{c} SP_1 \rhd_S \mathcal{S}_1, O_1 \\ SP_2 \rhd_S \mathcal{S}_2, O_2 \end{array}}{SP_1 \cup SP_2 \rhd_S \mathcal{S}_1 \cup \mathcal{S}_2 \uplus \langle \{N\}, \{O_i \xRightarrow{id} N | i = 1, 2\} \rangle, N}$$ [5]

where N is a new node having empty axiom set. Let $\mathcal{S}$ denote the development graph obtained in the conclusion. By induction hypothesis, $\mathbf{Mod}(SP_i) = \mathbf{Mod}_{\mathcal{S}_i}(O_i) = \mathbf{Mod}_{\mathcal{S}}(O_i)$. By Def. 8, the models of N are all models n such

[4] Additionally, [21] provide also further operations such as e.g. data (=freeness) constraints which are not supported in our framework but could be added if needed, as in [18].

[5] Thereby $\mathcal{S} \uplus \mathcal{S}'$ denotes the development graph resulting from the disjoint union of the nodes and the links of both $\mathcal{S}$ and $\mathcal{S}'$.

that $n|_{id} = n$ is a model of O_i for $i = 1, 2$. Hence, $\mathbf{Mod}(SP_1 \cup SP_2) = \mathbf{Mod}_{\mathcal{S}}(N)$.

8.1.3 Translate

Translations are translated to global definition links:

$$\frac{SP \rhd_S \mathcal{S}', O}{\textbf{translate } SP \textbf{ by } \sigma \rhd_S \mathcal{S}' \uplus \langle \{N\}, \{O \xrightarrow{\sigma} N\} \rangle, N}$$

where N is a new node having empty axiom set. Let $\mathcal{S}$ denote the development graph obtained in the conclusion. By induction hypothesis, $\mathbf{Mod}_{\mathcal{S}}(O) = \mathbf{Mod}(SP)$. Since N does not contain any local axioms, the models of N are only restricted by the fact that they have to be σ-reducts of the models of O according to Def. 8, and thus $\mathbf{Mod}(\textbf{translate } SP \textbf{ by } \sigma) = \mathbf{Mod}_{\mathcal{S}}(N)$ holds.

8.1.4 Derive

Deriving specifications corresponds to the use of a hiding link:

$$\frac{SP \rhd_S \mathcal{S}', O}{\textbf{derive from } SP \textbf{ by } \sigma \rhd_S \mathcal{S}' \uplus \langle \{N\}, \{O \xrightarrow[h]{\sigma} N\} \rangle, N}$$

where N is a new node having empty axiom set and Let $\mathcal{S}$ denote the development graph obtained in the conclusion. By induction hypothesis, $\mathbf{Mod}_{\mathcal{S}}(O) = \mathbf{Mod}(SP)$. According to Def. 8, the models of N are just the σ-reducts of the models of O. Hence, $\mathbf{Mod}(\textbf{derive from } SP \textbf{ by } \sigma) = \mathbf{Mod}_{\mathcal{S}}(N)$.

8.2 CASL

We briefly recapitulate the basic notions about CASL specifications that are relevant for the definition of the translation function into development graphs. The nucleus of CASL specifications are *basic specifications*, which consist of a (local) signature and a set of axioms. Therefore, basic specifications are simply translated into a new theory node with exactly this local signature and axioms. CASL *structured specifications* are formed by combining specifications in various ways, starting from basic specifications. For instance, specifications may be *united*; a specification may be *extended* with further signature items and/or sentences; parts of a signature may be *hidden*; the signature may be *translated* to use different symbols (with corresponding translation of the sentences) by a signature morphism; and models may be restricted to *initial* or *free* models. A CASL *library* of specifications consists of a list of *named specifications* and *named views*, which may be possibly generic, for later *instantiation* with possibly different specifications. The structuring concepts and constructs and their semantics do not depend on a specific institution; in particular, they can be used with institutions that are completely different from the CASL institution.

For the translation of CASL specifications into development graphs we restrict ourselves to a subset of the CASL language, namely CASL specifications without the structuring operation freeness (cf. [19]), without the semantic annotations `%mono` and `%def`, and without architectural specifications. Supporting the annotations `%mono` and `%def` and freeness would require further extensions of the development graph formalism, which are beyond the scope of this article. We refer the interested reader to [18]. Architectural specifications are in a sense orthogonal (and built on top of) the structured specifications that we consider here.

Roughly speaking, the translation of a CASL-specification to a development graph works as follows:

- it maps basic (unstructured) parts of the specification, like the specification of simple abstract datatypes, into a (local) signature and collection of (local) axioms of a new theory node,
- it translates the structuring operations of CASL into the various notions of definition link, and
- it reformulates proof obligations given in the specification either into theorem links connecting corresponding theories or into conjectures considered as local implications of a specific node in the development graph.

We now first informally explain how the transformation works, using a set of rewrite rules shown in Figure 6. To translate a CASL construct one starts with a *pre-development graph* consisting of a node which contains the (not-yet translated) CASL construct. In the figures, nodes which are not yet translated are represented as shaded boxes, translated nodes as circles.

A pre-development graph for a specification occurring in the pre-development graphs is translated by successive applications of rewrite rules (Fig. 6) to boxes. These boxes are decomposed until one eventually arrives at a graph in which there are no boxes left, i.e. a development graph representation. During this process, boxes may be created which have in-going or out-going links. The thick arrows indicate how these links are inherited when such a box is rewritten.

In the following we present the formal definition of the translation of CASL-

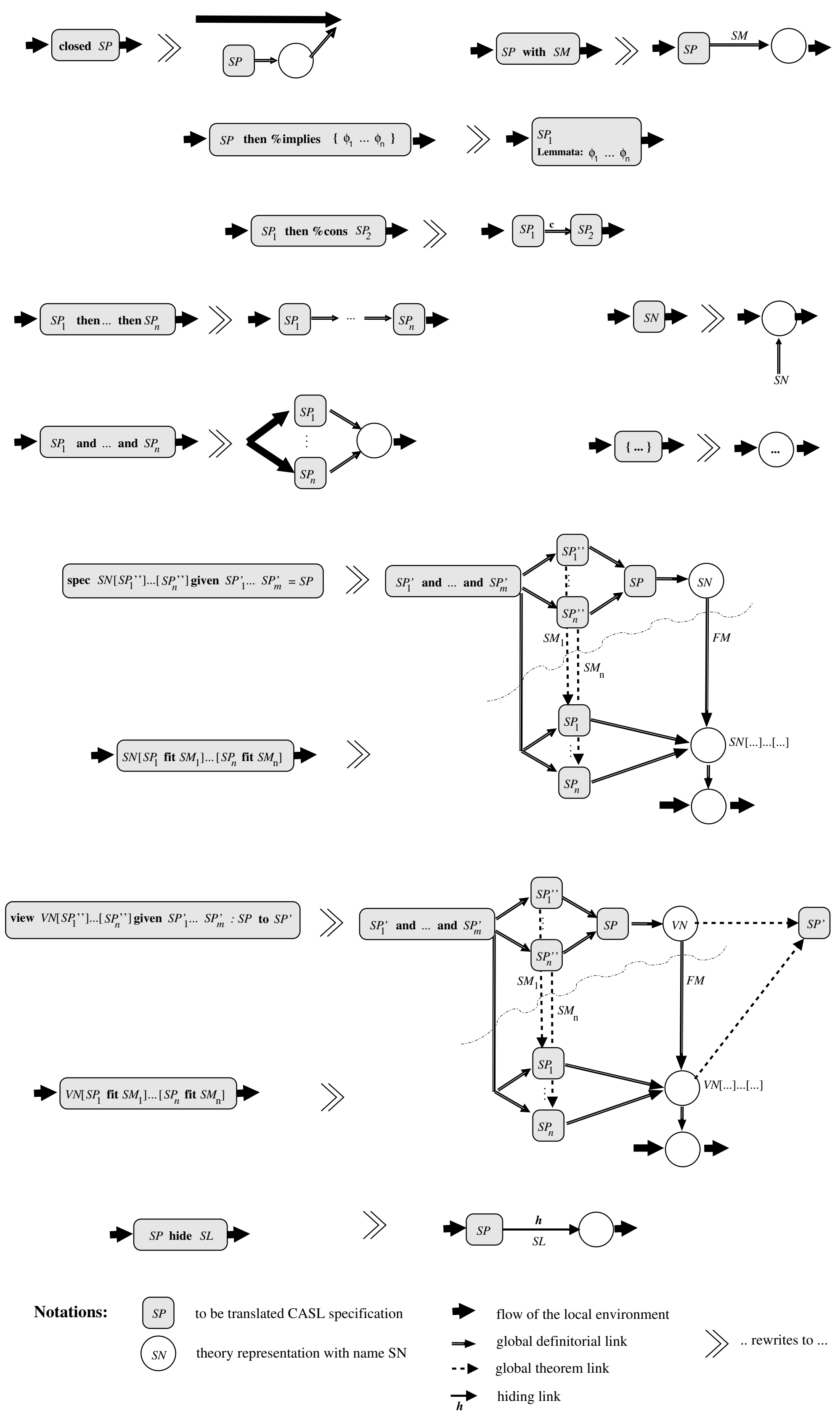

Fig. 6. Translating CASL specifications into development graphs with hiding

specifications into development graphs. Proving the adequacy of the translation would require to define the CASL semantics, which is out of scope for this paper. For the adequacy proof with refer the interested reader also to [18].

8.2.1 Translation of CASL into Development Graphs

The major difficulty of this translation is the encoding of the so-called *linear visibility constraint*, which is implicitly given by the CASL-semantics: the semantics of a specification part depends on its *local environment* which consists of previously parsed specification parts. Furthermore, also the *global environment* providing information about already translated named specifications and views (and their representations as nodes in a development graph), is needed. We make use of this information to translate references to named specifications and instantiations of generic specifications (and views).

For the definition of the translation of a CASL-specifications we introduce the following three judgement types:

(1) $\mathcal{S}, \mathcal{P}, \textit{lib-item-list} \rhd_L \mathcal{S}', \mathcal{P}'$ expressing that given a development graph $\mathcal{S}$ and a global environment $\mathcal{P}$, the translation of the CASL list *lib-item-list* of named specifications and named views in a CASL library results in the development graph $\mathcal{S}'$ and global environment $\mathcal{P}'$.

(2) $\mathcal{S}, \mathcal{P}, \textit{lib-item} \rhd_P \mathcal{S}', \mathcal{P}'$ for the translation of single named specifications and named views in a CASL library. Aside from the new development graph $\mathcal{S}'$, it yields the updated global environment $\mathcal{P}'$.

(3) $\mathcal{S}, \mathcal{P}, \textit{spec} \rhd_S \mathcal{I}', \mathcal{S}', O'$ for the translation of structured specifications. It yields $\mathcal{I}', \mathcal{S}', O'$ with $\mathcal{S}'$ being the development graph updated with the translated *spec*. The linear visibility constraints for structured CASL specifications defines *how* specification parts are visible when parsing the next specification. Translating this requirement in terms of development graphs, the development graphs of previous specifications have to be imported to the graph of the specification part under consideration. Therefore the translation of the structured specification also returns a set $\mathcal{I}'$ of nodes denoting the interface for the incoming local environment and a node O' which represents the outgoing local environment.

Formally, we define the translation top-level CASL libraries of named specifications and views as follows:

$$\frac{}{\mathcal{S}, \mathcal{P}, \langle\rangle \rhd_L \mathcal{S}, \mathcal{P}} \qquad \frac{\mathcal{S}, \mathcal{P}, \textit{lib-item} \rhd_P \mathcal{S}', \mathcal{P}' \qquad \mathcal{S}', \mathcal{P}', \textit{restlist} \rhd_L \mathcal{S}'', \mathcal{P}''}{\mathcal{S}, \mathcal{P}, \langle \textit{lib-item}, \textit{restlist}\rangle \rhd_L \mathcal{S}'', \mathcal{P}''}$$

In the next two paragraphs we define the translation of named specifications and named views. For presentational reasons the translation of fitting views is given after the structured specifications.

8.2.2 Named Specifications.

A definition of a (possibly generic) named specification in CASL is of the form

$$\textbf{spec } SN[SP_1]\ldots[SP_n] \textbf{ given } SP'_1,\ldots,SP'_m = SP \textbf{ end}$$

where SN is the name of the specification, $SP_1,\ldots,SP_n$ are the parameter specifications, $SP'_1,\ldots,SP'_m$ are so-called import specifications that are visible inside the parameter specifications, and SP is the body of the named specification. In order to satisfy the visibility rules from CASL the translation is done according to the following steps

(1) The translation of the union of the import specifications $SP'_1,\ldots,SP'_m$ is obtained by:

$$\mathcal{S},\mathcal{P},SP'_1 \textbf{ and} \ldots \textbf{and } SP'_m \rhd_S \mathcal{I}_0,\mathcal{S}_0,O_0$$

(2) The translation of the parameter specifications SP_i, $1 \le i \le n$, is obtained by:

$$\mathcal{S}_{i-1},\mathcal{P},SP_i \rhd_S \mathcal{I}_i,\mathcal{S}_i,O_i$$

and new definition links annotated with the identity morphism id are inserted to include the outgoing theory node O_0 into all elements of all $\mathcal{I}_i$ with $1 \le i \le n$:

$$\mathcal{S}_{n+1} := \mathcal{S}_n \uplus \langle \emptyset, \{O_0 \xrightarrow{id} N \mid N \in \mathcal{I}_i \wedge 1 \le i \le n\}\rangle$$

(3) Next, the body SP of the named specification is translated by:

$$\mathcal{S}_{n+1},\mathcal{P},SP \rhd_S \mathcal{I}_{n+2},\mathcal{S}_{n+2},O_{n+2}$$

and new definition links are added to include the parameters' outgoing nodes $O_1,\ldots,O_n$ into each element N of $\mathcal{I}_{n+2}$.

$$\begin{aligned}&\langle \mathcal{I}_{name},\mathcal{S}_{name},O_{name}\rangle :=\\ &\quad\langle \mathcal{I}_0, \mathcal{S}_{n+2} \uplus \langle \emptyset, \{O_i \xrightarrow{id} N \mid N \in \mathcal{I}_{n+2} \wedge 1 \le i \le n\}\rangle, O_{n+2}\rangle\end{aligned}$$

As global environment we store that the CASL-specification of name SN has the top-level output node O_{name}, and add the information about the translation of the parameter specifications $(\langle \mathcal{I}_i, O_i\rangle)$ with $1 \le i \le n$ as well as the outgoing node O_0 of the given part. This is used for the translation of instantiations of SN.

The final result of the translation of the named specification definition is then

$$\dfrac{\begin{array}{ll} \textit{1.} & \mathcal{S}, \mathcal{P}, SP'_1 \textbf{ and} \ldots \textbf{and } SP'_m \rhd_S \mathcal{I}_0, \mathcal{S}_0, O_0 \\ \textit{2.} & \mathcal{S}_0, \mathcal{P}, SP_1 \rhd_S \mathcal{I}_1, \mathcal{S}_1, O_1 \\ \vdots & \vdots \\ \textit{n+1.} & \mathcal{S}_{n-1}, \mathcal{P}, SP_n \rhd_S \mathcal{I}_n, \mathcal{S}_n, O_n \\ \textit{n+2.} & \mathcal{S}_n \uplus \langle \emptyset, \{O_0 \xRightarrow{id} N \mid N \in \mathcal{I}_i \wedge 1 \leq i \leq n\}\rangle, \mathcal{P}, SP \\ & \rhd_S \mathcal{I}_{n+2}, \mathcal{S}_{n+2}, O_{n+2} \end{array}}{\begin{array}{l} \mathcal{S}, \mathcal{P}, \textbf{spec } SN[SP_1] \ldots [SP_n] \textbf{ given } SP'_1, \ldots, SP'_m = SP \textbf{ end} \\ \rhd_S \; \mathcal{S}_{n+2} \uplus \langle \emptyset, \{O_i \xRightarrow{id} N \mid N \in \mathcal{I}_{n+2} \wedge 1 \leq i \leq n\}\rangle, \\ \quad \mathcal{P} \cup [SN, O_{n+2}, (\langle \mathcal{I}_1, O_1\rangle, \ldots, \langle \mathcal{I}_n, O_n\rangle), O_0] \end{array}}$$

8.2.3 Views.

A named view in CASL is of the general form

$$\begin{array}{l} \textbf{view } VN \; [SP_1] \ldots [SP_n] \textbf{ given } SP_1, \ldots, SP_m \; : \; SP \textbf{ to } SP' \\ \qquad = \; SM \textbf{ end}. \end{array} \tag{1}$$

$[SP_1] \ldots [SP_n]$ **given** $SP_1, \ldots, SP_m \; : \; SP$ represents a parameter information similar as in the definition of named specifications. The view constitutes the proof obligation that the models of SP' can be mapped to models of SP using the signature morphism induced by the symbol map SM, which we denote by $\mu(SM)$.

To translate a named view we translate a dummy named specification

$$\textbf{spec } SN \; [SP_1] \ldots [SP_n] \textbf{ given } SP_1, \ldots, SP_m \; = \; SP$$

which results as described in the previous paragraph in

$$\begin{array}{l} \mathcal{S}, \mathcal{P}, \textbf{spec } SN \; [SP_1] \ldots [SP_n] \textbf{ given } SP_1, \ldots, SP_m \; = \; SP \\ \rhd_P \mathcal{S}_{name}, \mathcal{P} \cup [SN, O_{name}, (\langle \mathcal{I}_1, O_1\rangle, \ldots, \langle \mathcal{I}_n, O_n\rangle), O_0]. \end{array}$$

Next we translate the structured specification SP' by

$$\mathcal{S}_{name}, \mathcal{P}, SP' \rhd_S \mathcal{I}', \mathcal{S}', O'$$

and add a global theorem link from O_{name} to O' with the morphism $\mu(SM)$ induced by SM. The final result of the translation of (1) consists of this new development graph and the parameter information for the named view. Let $= \langle \mathcal{N}', \Psi'_D \uplus \Psi'_T \rangle$ then

$$\frac{\begin{array}{l} \textit{1. } \mathcal{S}, \mathcal{P}, \textbf{spec } SN\ [SP_1] \ldots [SP_n] \textbf{ given } SP_1, \ldots, SP_m \ = \ SP \\ \quad \rhd_P \mathcal{S}_{name}, \mathcal{P} \cup [SN, O_{name}, (\langle \mathcal{I}_1, O_1 \rangle, \ldots, \langle \mathcal{I}_n, O_n \rangle), O_0] \\ \textit{2. } \mathcal{S}_{name}, \mathcal{P}, SP' \rhd_S \mathcal{I}', \mathcal{S}', O' \end{array}}{\begin{array}{l} \mathcal{S}, \mathcal{P}, \textbf{ view } VN\ [SP_1] \ldots [SP_n] \textbf{ given } SP_1, \ldots, SP_m \ : \ SP \textbf{ to } SP' \\ \qquad\qquad = \ SM \textbf{ end} \\ \rhd_P \mathcal{S}' \uplus \langle \emptyset, \{ O_{name} \xRightarrow{\mu(SM)} O' \} \rangle, \\ \quad \mathcal{P} \cup \{ [VN, (O_{name}, O'), (\langle \mathcal{I}_1, O_1 \rangle, \ldots, \langle \langle \mathcal{I}_n, O_n \rangle), O_0] \} \end{array}}$$

8.2.4 Structured specifications.

We now define the translation of basic specifications and the structuring operations in CASL.

8.2.4.1 Basic Specifications.

A basic specification is a pair (Σ, Γ) composed of a signature Σ and a collection of axioms Γ. We create a new node N with local signature $\Sigma^N := \Sigma$ and local axioms $\Gamma^N := \Gamma$ and add it to the development graph. The node N is both the node where the current local environment shall be included into N as well as the outgoing node, i.e. the node that contains the local environment "after" parsing the basic specification.

$$\mathcal{S}, \mathcal{P}, (\Sigma, \Gamma) \rhd_S \{N\}, \mathcal{S} \uplus \langle \{N\}, \emptyset \rangle, N$$

8.2.4.2 Translations.

A translation is of the form SP **with** SM, where SP is a structured specification and SM a symbol map. Let

$$\mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}', \mathcal{S}', O'$$

be the translation of SP and let N be a new node in $\mathcal{S}'$ with signature as determined by the target of $\mu(SM)$ and no local axioms. We add this new node to the new development graph and include SP's outgoing node O' into

N via a global definition link with morphism $\mu(SM)$.

$$\frac{\mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}', \mathcal{S}', O'}{\mathcal{S}, \mathcal{P}, SP \textbf{ with } SM \rhd_S \mathcal{I}', \mathcal{S}' \uplus \langle \{N\}, \{O' \xRightarrow{\mu(SM)} N\} \rangle, N}$$

8.2.4.3 Extensions. There are three kinds of extensions in CASL, namely SP **then** SP', SP **then %implies** SP', and SP **then %cons** SP'. The first is a *regular* extension of SP by SP', while the second requires is an *implied* extension, i.e. the added axioms are only implied and the third denotes a *conservative* extension.

- *Regular Extensions* are translated as follows:

$$\frac{\mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}', \mathcal{S}', O' \qquad \mathcal{S}', \mathcal{P}, SP' \rhd_S \mathcal{I}'', \mathcal{S}'', O''}{\mathcal{S}, \mathcal{P}, SP \textbf{ then } SP' \rhd_S \mathcal{I}', \mathcal{S}'' \uplus \langle \emptyset, \{O' \xRightarrow{id} I'' \mid I'' \in \mathcal{I}''\} \rangle, O''}$$

- *Conservative Extensions* are translated analogously to regular extensions, except that a conservative definition link is added instead of a normal definition link:

$$\frac{\mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}', \mathcal{S}', O' \qquad \mathcal{S}', \mathcal{P}, SP' \rhd_S \mathcal{I}'', \mathcal{S}'', O''}{\mathcal{S}, \mathcal{P}, SP \textbf{ then } SP' \rhd_S \mathcal{I}', \mathcal{S}'' \uplus \langle \emptyset, \{O' \xRightarrow[c]{id} I'' \mid I'' \in \mathcal{I}''\} \rangle, O''}$$

- *Implied Extensions:* The CASL-semantics requires from an implied extension SP **then %implies** SP' that SP' is a *basic specification* with the same signature as that of SP, and the local axioms of SP' must be implied properties with respect to SP. Therefore we translate an implied extension like a regular extension and add a local implication between the top-level node for SP' to the one obtained for SP:

$$\frac{\mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}', \mathcal{S}', O' \qquad \mathcal{S}', \mathcal{P}, SP' \rhd_S \mathcal{I}'', \mathcal{S}'', O''}{\begin{array}{l}\mathcal{S}, \mathcal{P}, SP \textbf{ then } SP' \\ \rhd_S \mathcal{I}', \mathcal{S}'' \uplus \langle \emptyset, \{O' \xRightarrow[c]{id} I'' \mid I'' \in \mathcal{I}''\} \cup \{O' \Rightarrow Th_{\mathcal{S}''}(O'')\} \rangle, O''\end{array}}$$

8.2.4.4 Union. A union of specifications in CASL is of the form SP **and** SP'. In order to adequately represent the union of the specifications, we add a new node without local axioms and include both SP's and SP''s outgoing theory nodes via global definition links. Thus:

$$\frac{\begin{array}{c}\textit{1. } \mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}, \mathcal{S}', O \\ \textit{2. } \mathcal{S}', \mathcal{P}, SP' \rhd_S \mathcal{I}', \mathcal{S}'', O'\end{array}}{\mathcal{S}, \mathcal{P}, SP \ \mathbf{and}\ SP' \rhd_S \mathcal{I} \cup \mathcal{I}', \mathcal{S}'' \uplus \langle \{N\}, \{O \xRightarrow{id} N, O' \xRightarrow{id} N\}\rangle, N}$$

where N is a new node without local axioms.

The theory nodes to include the local environment into are the union of both $\mathcal{I}$ and $\mathcal{I}'$, while the local environment "after" the union is the global signature of the new, outgoing node N.

8.2.4.5 Closed specifications. They are of the form **closed**$\{SP\}$. The semantics is that the local environment is not visible inside SP, but shall still be visible "after" **closed**$\{SP\}$ together with the environment generated from SP. Thus, the translation of the closed specification consists in creating a new empty node N, include the environment from SP into N via a global definition link, and returning N as both the incoming and outgoing theory node for the local environment. Thus:

$$\frac{\mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}, \mathcal{S}', O}{\mathcal{S}, \mathcal{P}, \mathbf{closed}\{SP\} \rhd_S N, \mathcal{S}' \uplus \langle \{N\}, \{O \xRightarrow{id} N\}\rangle, N}$$

8.2.4.6 Instantiation. An instantiation in Casl is of the general form

$SN\ [SP_1\ \mathbf{fit}\ SM_1] \ldots [SP_n\ \mathbf{fit}\ SM_n]$.

Its semantics is that the formal parameter of the formerly declared named specification SN are instantiated with the SP_i and the "**given**"-specifications of SN are imported into the actual parameters SP_i. This is only well-formed if the actual parameters fit the formal parameters theories modulo the morphisms induced by the SM_i. A parameter information for SN is

$$[SN, O, (\langle \mathcal{I}'_1, O'_1\rangle, \ldots, \langle \mathcal{I}'_n, O'_n\rangle), O_I],$$

where O is the node representing the outgoing local environment for SN, $\langle \mathcal{I}'_i, O'_i\rangle$ the information about incoming and outgoing local environments of the parameter theories, and O_I the top-level theory node that is inluded into the parameters. Given this parameter information for the named specification

SN, let $\mathcal{S}_0$ be the development graph before translating the instantiation. Then the translations of the SP_i, $1 \leq i \leq n$, are given by:

$$\mathcal{S}_{i-1}, \mathcal{P}, SP_i \rhd_S \mathcal{I}_i, \mathcal{S}_i, O_i$$

To construct an adequate development graph for the instantiation

(1) we include the import ("**given**") local environment by including the theory O_I into each theory in $\mathcal{I}_i$, for all $1 \leq i \leq n$:

$$\mathcal{S}_n \uplus \langle \emptyset, \{O_I \xRightarrow{id} I \mid I \in \bigcup_{i=1}^{n} \mathcal{I}_i\}\rangle$$

(2) We further encode the well-formedness condition required by **fit** by introducing global theorem links from each O'_i to O_i with morphisms $\mu(SM_i)$:

$$\mathcal{S}_n \uplus \langle \emptyset, \{O_I \xRightarrow{id} I \mid I \in \bigcup_{i=1}^{n} \mathcal{I}_i\} \cup \{O'_i \xRightarrow{\mu(SM_i)} O_i \mid 1 \leq i \leq n\}\rangle$$

(3) Finally, we create the node N_I to encode the instantiated theory: This node includes the SN's local environment via the top-level node O, as well as local environments of the actual parameter theories via the the top-level nodes O_i.

$$\begin{aligned}\mathcal{S}_n \uplus \langle \{N_I\},\ &\{O_I \xRightarrow{id} I \mid I \in \textstyle\bigcup_{i=1}^{n} \mathcal{I}_i\} \rangle\\ &\cup\{O \xRightarrow{\mu(SM)} N_I, O_1 \xRightarrow{id} N_I, \ldots, O_n \xRightarrow{id} N_I\}\\ &\cup\{O'_i \xRightarrow{\mu(SM_i)} O_i \mid 1 \leq i \leq n\}\end{aligned}$$

where $SM := \bigcup_{i=1}^{n} SM_i$.

From there we define the final translation rule for instantiations by:

$$\frac{\begin{array}{l} \mathit{1.}\ \mathcal{S}_0, \mathcal{P}, SP_1 \rhd_S \mathcal{I}_1, \mathcal{S}_1, O_1 \\ \vdots \\ \mathit{n.}\ \mathcal{S}_{n-1}, \mathcal{P}, SP_n \rhd_S \mathcal{I}_n, \mathcal{S}_n, O_n \end{array}}{\begin{array}{l} \mathcal{S}_0, \mathcal{P} \cup \{[SN, O, (\langle \mathcal{I}'_1, O'_1\rangle, \ldots, \langle \mathcal{I}'_n, O'_n\rangle), O_I]\}, \\ SN\ [SP_1\ \mathbf{fit}\ SM_1] \ldots [SP_n\ \mathbf{fit}\ SM_n] \\ \rhd_S \{N_I\}, \mathcal{S}_n \uplus \langle \{N_I\},\ \{O_I \xRightarrow{id} I \mid I \in \bigcup_{i=1}^{n} \mathcal{I}_i\} \qquad \rangle, N_I \\ \qquad \cup\{O \xRightarrow{\mu(SM)} N_I, O_1 \xRightarrow{id} N_I, \ldots, O_n \xRightarrow{id} N_I\} \\ \qquad \cup\{O'_i \xRightarrow{\mu(SM_i)} O_i \mid 1 \leq i \leq n\} \end{array}}$$

8.2.4.7 Hiding. Hiding specifications in CASL are of the form SP **hide** SL, where SP is a specification and SL a list of symbols. The translation of these specifications is by first translating SP and then add a new node without local axioms N, which imports the top-level theory for SP by a hiding definition link. The morphism on that link is the inclusion ι induced by the symbol list SL.

$$\frac{\mathcal{S}, \mathcal{P}, SP \rhd_S \mathcal{I}', \mathcal{S}', \mathcal{P}, O'}{\mathcal{S}, \mathcal{P}, SP \textbf{ hide } SL \rhd_S \mathcal{I}', \mathcal{S}' \uplus \langle \{N\}, \{O' \xRightarrow[h]{\iota} N\}\rangle, N}$$

This completes the definition of the translation of structured specification. It remains to define the translation of fitting views, which we postponed until here.

8.2.5 Fitting Views

A fitting view is of the form $VN\ [SP_1 \textbf{ fit } SM_1] \ldots [SP_n \textbf{ fit } SM_n]$, where VN is the name of a view. The translation of this fitting view is analogously to the translation of an instantiation of a named specification, except that an additional global theorem link from the instantiated theory N_I to the top-level theory node O' obtained for the target SP of the view VN is inserted. Note that the CASL language definition requires the morphism on the theorem link from N_I to O' to be the identity.

$$\frac{\begin{array}{l} \mathit{1.}\ \mathcal{S}_0, \mathcal{P}, SP_1 \rhd_S \mathcal{I}_1, \mathcal{S}_1, O_1 \\ \vdots \\ n.\ \mathcal{S}_{n-1}, \mathcal{P}, SP_n \rhd_S \mathcal{I}_n, \mathcal{S}_n, O_n \end{array}}{\begin{array}{l} \mathcal{S}_0, \mathcal{P} \cup \{[VN, (O_{name}, O'), (\langle \mathcal{I}_1, O_1\rangle, \ldots, \langle\langle \mathcal{I}_n, O_n\rangle), O_0]\}, \\ VN\ [SP_1] \ldots [SP_n] \\ \rhd_P \mathcal{S}_n \uplus \langle \{N_I\},\ \{O_I \xRightarrow{id} I \mid I \in \bigcup_{i=1}^n \mathcal{I}_i\} \qquad\qquad \rangle \\ \qquad\qquad \cup \{O_{name} \xRightarrow{\mu(SM)} N_I, O_1 \xRightarrow{id} N_I, \ldots, O_n \xRightarrow{id} N_I\} \\ \qquad\qquad \cup \{O_i' \xRightarrow{\mu(SM_i)} O_i \mid 1 \leq i \leq n\} \cup \{N_I \xRightarrow{id} O'\}, \\ \quad \mathcal{P} \cup \{[VN, (O_{name}, O'), (\langle \mathcal{I}_1, O_1\rangle, \ldots, \langle\langle \mathcal{I}_n, O_n\rangle), O_0]\} \end{array}}$$

Goguen and Rosu [20] define a system of hidden information modules, which follows the OBJ tradition of giving a theory-level semantics to modules, but adds the possibility to use hidden symbols.

Their module system is defined over an arbitrary inclusive institution. We refrain here from repeating the details of the definition of an inclusive institution; it suffices to know that they come with a notion of inclusion signature morphism (usually denoted by $\Sigma_1 \hookrightarrow \Sigma_2$), and notions of intersection and of union of signatures (denoted by $\Sigma_1 \cap \Sigma_2$ and $\Sigma_1 \cup \Sigma_2$). Sentence translations along inclusions are inclusions (and hence often omitted). Moreover, the signature category is assumed to have pushouts, which are translated to pullbacks of model categories (i.e. the amalgamation property holds). There are selected pushouts of signature morphisms along inclusions, written

$$\begin{tikzcd}
\Phi \arrow[r] \arrow[d, "h"] & \Sigma \arrow[d, "h_\Sigma"] \\
\Phi' \arrow[r] & \Sigma_h
\end{tikzcd}$$

and the union of signatures is a pushout w.r.t. their intersection

$$\begin{tikzcd}
\Sigma_1 \cap \Sigma_2 \arrow[r] \arrow[d] & \Sigma_1 \arrow[d] \\
\Sigma_2 \arrow[r] & \Sigma_1 \cup \Sigma_2
\end{tikzcd}$$

A (basic) *module* is a triple (Φ, Σ, A), where $\Phi \hookrightarrow \Sigma$ is a signature inclusion and A is a set of Σ-sentences. Φ is called the *visible signature*, Σ the *working signature*. The *visible theorems* of a module are given by

$$Vth(M) = \iota^{-1}(A^\bullet)$$

where $\iota : \Phi \hookrightarrow \Sigma$ is the inclusion, while the *working theorems* are given by

$$Th(M) = A^\bullet$$

Here, given a set of sentences A, we denote by $A^\bullet$ its set of consequences

$$\{\varphi | A \models \varphi\}.$$

A module M is called *conservative* if the theory inclusion $(\Phi, Vth(M)) \hookrightarrow (\Sigma, Th(M))$ is conservative, i.e. each $(\Phi, Vth(M))$-model has some expansion to a $(\Sigma, Th(M))$-model.

While Goguen and Rosu define a module semantics in terms of the visible theorems, for development graph compatibility reasons we here prefer a different semantics, namely

$$\mathbf{Mod}(M) = \mathbf{Mod}(\Phi, \Sigma, A) = \mathbf{Mod}(\Phi \hookrightarrow \Sigma)(\mathbf{Mod}(\Sigma, A)),$$

i.e. all reducts of (Σ, A)-models. It is easy to see that

Proposition 20 For conservative modules M, the above semantics agrees with the original one of [20]:

$$\mathbf{Mod}(M) = \mathbf{Mod}(Vth(M))$$

PROOF. $M \in \mathbf{Mod}(Vth(M))$, iff (by conservativity) M can be expanded to $M' \in \mathbf{Mod}(Th(M))$ iff $M \in \mathbf{Mod}(\Phi \hookrightarrow \Sigma)(\mathbf{Mod}(\Sigma, A)) = \mathbf{Mod}(M)$. $\square$

In [20], criteria for conservativity of modules are studied.

We now come to the module language. While [20] give a semantics to the language by directly passing over to the visible theorems, we here prefer to construct a basic module first, and then take our above semantics of this module (which, as said, for conservative modules, coincides with the visible theorem semantics).

Aggregation Given modules M_1 and M_2 with $[\![M_1]\!] = (\Phi_1, \Sigma_1, A_1)$ and $[\![M_2]\!] = (\Phi_2, \Sigma_2, A_2)$, their aggregation is written as $M_1 + M_2$ and denotes the module

$$[\![M_1 + M_2]\!] = (\Phi_1 \cup \Phi_2, \Sigma_1 \cup \Sigma_2, A_1 \cup A_2).$$

Call an aggregation *private*, if $\Phi_1 \cap \Phi_2 = \Sigma_1 \cap \Sigma_2$.

Renaming Given a module M with $[\![M]\!] = (\Phi, \Sigma, A)$ and a signature morphism $h: \Sigma \longrightarrow \Sigma'$, the renaming of M by h is written $M \star h$ and denotes the module (recall the above pushout notation)

$$[\![M \star h]\!] = (\Phi', \Sigma_h, h_\Sigma(A)).$$

Enrichment Given a module M with $[\![M]\!] = (\Phi, \Sigma, A)$ and a basic module (Φ', Σ', A') with $\Phi \hookrightarrow \Phi'$ and $\Sigma \hookrightarrow \Sigma'$, the enrichment of M by (Φ', Σ', A') is written $M@(\Phi', \Sigma', A')$ and denotes the module

$$[\![M@(\Phi', \Sigma', A')]\!] = (\Phi', \Sigma', A \cup A')$$

Enrichment is a special case for aggregation; however, aggregations often are required to be private (e.g. in the results below and in those of [20]), while enrichments are not.

Hiding Given a module M with $[\![M]\!] = (\Phi, \Sigma, A)$ and a subsignature Ψ of Φ, then the hiding restriction of M to Ψ is written $\Psi \Box M$ and denotes the module

$$[\![\Psi \Box M]\!] = (\Psi, \Sigma, A).$$

We now come to the translation of hidden information modules to development graphs. In the rules, inclusion signature morphisms are denoted by ι.

8.3.1 Basic modules

$$(\Phi, \Sigma, A) \rhd_S \langle\{N_1 = (\Sigma, A); N_2 = (\Phi, \emptyset)\}, \{N_1 \xRightarrow[h]{\iota} N_2\}\rangle, N_2$$

8.3.2 Aggregation

$$\frac{\begin{array}{c} M_1 \rhd_S \mathcal{S}_1, N_1 \\ M_2 \rhd_S \mathcal{S}_2, N_2 \end{array}}{M_1 + M_2 \rhd_S \mathcal{S}_1 \uplus \mathcal{S}_2 \uplus (\{N = (\Sigma^{N_1} \cup \Sigma^{N_2}, \emptyset)\}, \{N_1 \xRightarrow{\iota_1} N, N_2 \xRightarrow{\iota_1} N\}), N}$$

8.3.3 Renaming

$$\frac{M \rhd_S \mathcal{S}, N_1}{M \star \sigma \colon \Sigma \longrightarrow \Sigma' \rhd_S \mathcal{S} \uplus (\{N_2 = (\Sigma', \emptyset)\}, \{N_1 \xRightarrow{\sigma} N_2\}), N_2}$$

8.3.4 Enrichment

$$\frac{M \rhd_S \mathcal{S}, N_1}{M @ (\Phi', \Sigma', A') \rhd_S \mathcal{S} \uplus (\{N_2 = (\Sigma', A'); N_3 = (\Phi', \emptyset)\}, \{N_1 \xRightarrow{\iota} N_2; N_2 \xRightarrow[h]{\iota} N_3\}), N_3}$$

8.3.5 Hiding

$$\frac{M \rhd_S \mathcal{S}, N_1}{\Psi \Box M \rhd_S \mathcal{S} \uplus (\{N_2 = (\Psi, \emptyset)\}, \{N_1 \xRightarrow[h]{\iota} N_2\}), N_2}$$

We now prove the adequacy of the translation.

Theorem 21 Given a module M containing only private aggregations, if

$$M \rhd_S \mathcal{S}, N,$$

then

$$\mathbf{Mod}([\![M]\!]) = \mathbf{Mod}_{\mathcal{S}}(N).$$

Note that by Prop. 20, this in turn equal to $\mathbf{Mod}(Vth([\![M]\!]))$ in case that M is conservative.

PROOF. By induction over the structure of M.

Basic modules

$$\mathbf{Mod}([\![(\Phi, \Sigma, A)]\!]) = \mathbf{Mod}(\Phi, \Sigma, A) = \mathbf{Mod}(\Phi \hookrightarrow \Sigma)(\mathbf{Mod}(\Sigma, A)) = \mathbf{Mod}_{\mathcal{S}}(N).$$

Aggregation Let $[\![M_i]\!] = (\Phi_i, \Sigma_i, A_i)$ and $M_i \rhd_S \mathcal{S}, N_i$ $(i = 1, 2)$. Then

$\mathbf{Mod}([\![M_1 + M_1]\!]) =$

$\mathbf{Mod}(\Phi_1 \cup \Phi_2, \Sigma_1 \cup \Sigma_2, A_1 \cup A_2) =$

$\mathbf{Mod}(\iota)(\mathbf{Mod}(\Sigma_1 \cup \Sigma_2, A_1 \cup A_2)) = (*)$

$\mathbf{Mod}(\iota_1^{-1}(\mathbf{Mod}(\theta_1)(\mathbf{Mod}(\Sigma_1, A_1)) \cap \mathbf{Mod}(\iota_2^{-1}(\mathbf{Mod}(\theta_2)(\mathbf{Mod}(\Sigma_2, A_2)) =$

$\mathbf{Mod}(\iota_1^{-1}(\mathbf{Mod}(\Phi_1, \Sigma_1, A_1)) \cap \mathbf{Mod}(\iota_2^{-1}(\mathbf{Mod}(\Phi_2, \Sigma_2, A_2)) =$

$\mathbf{Mod}(\iota_1^{-1}(\mathbf{Mod}(M_1)) \cap \mathbf{Mod}(\iota_2^{-1}(\mathbf{Mod}(M_2)) =$ (induction hypothesis)

$\mathbf{Mod}(\iota_1^{-1}(\mathbf{Mod}_{\mathcal{S}}(N_1)) \cap \mathbf{Mod}(\iota_2^{-1}(\mathbf{Mod}_{\mathcal{S}}(N_2)) =$

$\mathbf{Mod}_{\mathcal{S}}(N)$

Equation $(*)$ follows from the aggregation being private as follows. By privacy (i.e. $\Phi_1 \cap \Phi_2 = \Sigma_1 \cap \Sigma_2$), the outer diagram is a pushout.

$$\begin{tikzcd}
 & \Phi_1 \arrow[rr, "\theta_1"] \arrow[dr, "\iota_1"] & & \Sigma_1 \arrow[dr, "\tau_1"] & \\
\Phi_1 \cap \Phi_2 \arrow[ur, "\kappa_1"] \arrow[rr, "\kappa"] \arrow[dr, "\kappa_2"'] & & \Phi_1 \cup \Phi_2 \arrow[rr, "\iota"] & & \Sigma_1 \cup \Sigma_2 \\
 & \Phi_2 \arrow[rr, "\theta_2"] \arrow[ur, "\iota_2"'] & & \Sigma_2 \arrow[ur, "\tau_2"'] &
\end{tikzcd}$$

To prove the upwards inclusion of $(*)$, assume that

$$\begin{aligned} M \in\ & \mathbf{Mod}(\iota_1^{-1}(\mathbf{Mod}(\theta_1)(\mathbf{Mod}(\Sigma_1, A_1)) \cap \\ & \mathbf{Mod}(\iota_2^{-1}(\mathbf{Mod}(\theta_2)(\mathbf{Mod}(\Sigma_2, A_2))), \end{aligned}$$

i.e. there are $M_i \in \mathbf{Mod}(\Sigma_i, A_i)$ with $M_i|_{\theta_i} = M|_{\iota_i}$ for $i = 1, 2$. Thus, $M_1|_{\theta_1 \circ \kappa_1} = M_2|_{\theta_2 \circ \kappa_2}$. By the amalgamation property, there is an $M' \in \mathbf{Mod}(\Sigma_1 \cup \Sigma_2)$ with $M'|_{\tau_i} = M_i$ for $i = 1, 2$. Since $M'|_{\tau_i} = M_i \models A_i$ for $i = 1, 2$, by the satisfaction condition, $M' \models A_1 \cup A_2$. Hence, $M'|_{\iota} \in \mathbf{Mod}(\iota)(\mathbf{Mod}(\Sigma_1 \cup \Sigma_2, A_1 \cup A_2))$. Since the right rhombus is a pushout, by uniqueness of amalgamation, $M'|_{\iota} = M$. The downwards inclusion of $(*)$ is easy.

Renaming Let $[\![M]\!] = (\Phi, \Sigma, A)$, $[\![M \star h]\!] = (\Phi', \Sigma_h, h_\Sigma(A))$, and $M \rhd_S \mathcal{S}, N_1$. Then

$$\begin{aligned}
&\mathbf{Mod}([\![M \star h]\!]) = \\
&\mathbf{Mod}(\Phi', \Sigma_h, h_\Sigma(A)) = \\
&\mathbf{Mod}(\Phi' \hookrightarrow \Sigma_h)(\mathbf{Mod}(\Sigma_h, h_\Sigma(A))) = (*) \\
&\mathbf{Mod}(h)^{-1}(\mathbf{Mod}(\Phi \hookrightarrow \Sigma(\mathbf{Mod}(\Sigma, A)))) = \\
&\mathbf{Mod}(h)^{-1}(\mathbf{Mod}(\Phi, \Sigma, A)) = \\
&\mathbf{Mod}(h)^{-1}(\mathbf{Mod}([\![M]\!])) = \text{(induction hypothesis)} \\
&\mathbf{Mod}(h)^{-1}(\mathbf{Mod}_{\mathcal{S}}(N_1)) = \\
&\mathbf{Mod}_{\mathcal{S}}(N)
\end{aligned}$$

The equation $(*)$ follows by amalgamation.

Enrichment Let $[\![M]\!] = (\Phi, \Sigma, A)$, $[\![M@(\Phi', \Sigma', A')]\!] = (\Phi', \Sigma', A \cup A')$, $M \rhd_S \mathcal{S}, N_1$, and N_2 and $N_3 = N$ be constructed as in the rule for enrichment. Then

$$\begin{aligned}
&\mathbf{Mod}([\![M@(\Phi', \Sigma', A')]\!]) = \\
&\mathbf{Mod}(\Phi', \Sigma', A \cup A') = \\
&\mathbf{Mod}(\Phi' \hookrightarrow \Sigma')(\mathbf{Mod}(\Sigma', A \cup A'))
\end{aligned}$$

Now since A consists of Σ-sentences only, $\mathbf{Mod}(\Phi' \hookrightarrow \Sigma')(\mathbf{Mod}(\Sigma', A))$ is just $\mathbf{Mod}(\Phi \hookrightarrow \Phi')^{-1}(\mathbf{Mod}(\Phi \hookrightarrow \Sigma)(\mathbf{Mod}(\Sigma, A)))$, which is in turn $\mathbf{Mod}(\Phi \hookrightarrow \Phi')^{-1}(\mathbf{Mod}(\Phi, \Sigma, A))$, hence is $\mathbf{Mod}(\Phi \hookrightarrow \Phi')^{-1}(\mathbf{Mod}([\![M]\!]))$ and, by induction hypothesis, equals $\mathbf{Mod}(\Phi \hookrightarrow \Phi')^{-1}(\mathbf{Mod}_{\mathcal{S}}(N_1))$.

This implies that $\mathbf{Mod}_{\mathcal{S}}(N_2)) = \mathbf{Mod}(\Sigma', A) \cap \mathbf{Mod}(\Sigma', A')$. Hence $\mathbf{Mod}_{\mathcal{S}}(N_3)) = \mathbf{Mod}(\Phi' \hookrightarrow \Sigma')(\mathbf{Mod}(\Sigma', A \cup A'))$.

Hiding Let $[\![M]\!] = (\Phi, \Sigma, A)$, $[\![\Psi \square M]\!] = (\Psi, \Sigma, A)$ and $M \rhd_S \mathcal{S}, N_1$. Then

$$\begin{aligned}
&\mathbf{Mod}([\![\Psi \square M]\!]) = \\
&\mathbf{Mod}(\Psi, \Sigma, A) = \\
&\mathbf{Mod}(\Psi \hookrightarrow \Sigma)(\mathbf{Mod}(\Sigma, A)) = \\
&\mathbf{Mod}(\Psi \hookrightarrow \Phi)(\mathbf{Mod}(\Phi \hookrightarrow \Sigma)(\mathbf{Mod}(\Sigma, A))) = \\
&\mathbf{Mod}(\Psi \hookrightarrow \Phi)(\mathbf{Mod}(\Phi, \Sigma, A)) = \\
&\mathbf{Mod}(\Psi \hookrightarrow \Phi)([\![M]\!]) = \text{(induction hypothesis)} \\
&\mathbf{Mod}(\Psi \hookrightarrow \Phi)(\mathbf{Mod}_{\mathcal{S}}(N_1)) = \\
&\mathbf{Mod}_{\mathcal{S}}(N)
\end{aligned}$$

□

8.4 Maude's Module Algebra

The module algebra of the Maude language [8] also can be translated to development graphs. Duran and Meseguer [9] translate Maude modules to theories in the institution of *structured theories*. Structured theories are basically diagrams of theories. Now a diagram of theories is nearly a development graph: one just has to forget composition. Note that development graphs without hiding suffice here.

9 Conclusion and related work

We have extended the notion of development graph [3] to deal also with hiding. We have developed a semantics and a proof system for development graphs with hiding. The proof system can easily shown to be sound.

We have shown that three logic independent module systems from the literature (namely structured specifications [21], Casl specifications [19] and hidden information modules [20]) can be mapped to development graphs. Concerning Maude's module algebra, the translation into diagrams of theories given in the literature about Maude [9] can be easily adapted to development graphs as target formalism.

Concerning completeness of our rule system, with a counterexample, we show that there can be no complete recursively axiomatized proof system for development graphs with hiding. However, we have shown our proof rules to be complete relative to a given oracle for detecting conservative extensions. We thus have achieved the same degree of completeness as Borzyszkowski's [6] rule system for structured specifications. In one sense our system is more complete than Borzyszkowski's: since our *Theorem-Hide-Shift* rule simulates something like Borzyszkowski's normal forms, we do not have to rely on the Craig interpolation property. For example, it is possible to solve a counterexample showing incompleteness in case of failure of interpolation in [6] with our rules. Borzyszkowski refrains from doing normal form inference because with his way of computing normal forms, the structure of the specification is lost. Note that this is not the case with our rules, since they just extend the structure of the development graph, while the axioms are kept locally. One can even further optimize the rule *Theorem-Hide-Shift* by reducing the constructed diagram to those nodes that are really necessary, which can be achieved by taking the so-called final subcategory [1].

Compared with the rules in [6], we have fewer but more complex rules. Our rules involve colimit computations that may be tedious for humans using the rules directly, but that are no problem for computer assisted proofs. Indeed, by exploiting the graph structure, development graphs with hiding can lead to much more efficient proofs than possible when using the usual proof rules for structured specifications as in [6].

The calculus rules presented in this paper have been extended to heterogeneous development graphs [15,16] and implemented in the heterogeneous tool set [14,16]. Moreover, we expect no difficulty when extending the management of change developed in [3] to the case of development graphs with hiding in order to integrate the presented approach into the development graph systems of INKA 5.0 [2], MAYA[4] and the heterogeneous tool set. Then it will be possible to support the maintenance of changes in developments by machine also when hidings are present.

References

[1] J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories.* Cambridge University Press, 1994.

[2] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. System description: Inka 5.0 - a logic voyager. In H. Ganzinger, editor, *Proceedings of CADE-16, Trento, Italy*, volume 1632 of *LNAI.* Springer-Verlag, 1999.

[3] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an evolutionary formal software-development using CASL. In C. Choppy and D. Bert, editors, *Recent Trends in Algebraic Development Techniques, 14th International Workshop, WADT'99, Bonas, France*, volume 1827 of *Lecture Notes in Computer Science*, pages 73–88. Springer-Verlag, 2000.

[4] S. Autexier and T. Mossakowski. Integrating HOL-Casl into the development graph manager Maya. In A. Armando, editor, *Frontiers of Combining Systems, 4th International Workshop*, volume 2309 of *Lecture Notes in Computer Science*, pages 2–17. Springer-Verlag, 2002.

[5] J. Bergstra, J. Heering, and P. Klint. Module algebra. *J. ACM*, 37(2):335–372, 1990.

[6] T. Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286:197–245, 2002.

[7] R. Diaconescu, J. Goguen, and P. Stefaneas. Logical support for modularisation. In G. Huet and G. Plotkin, editors, *Proceedings of a Workshop on Logical Frameworks*, 1991.

[8] F. Durán. *A Reflective Module Algebra with Applications to the Maude Language*. PhD thesis, Universidad de Málaga, Spain, June 1999. `http://maude.csl.sri.com/papers`.

[9] F. Durán and J. Meseguer. Structured theories and institutions. *Theor. Comput. Sci.*, 309(1-3):357–380, 2003.

[10] 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.

[11] R. Hennicker, M. Wirsing, and M. Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173(2):393–443, Feb. 1997.

[12] D. Hutter, B. Langenstein, C. Sengler, J. H. Siekmann, W. Stephan, and A. Wolpers. Verification support environment. *High Integrity Systems*, 1(6):523–530, 1996.

[13] J. Meseguer. General logics. In *Logic Colloquium 87*, pages 275–329. North Holland, 1989.

[14] T. Mossakowski. The heterogeneous tool set. Available at `www.tzi.de/cofi/hets`, University of Bremen.

[15] T. Mossakowski. Comorphism-based Grothendieck logics. In K. Diks and W. Rytter, editors, *Mathematical foundations of computer science*, volume 2420 of *LNCS*, pages 593–604. Springer, 2002.

[16] T. Mossakowski. Heterogeneous specification and the heterogeneous tool set. Habilitation thesis, University of Bremen, 2005.

[17] T. Mossakowski, S. Autexier, and D. Hutter. Extending development graphs with hiding. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering*, volume 2029 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 2001.

[18] T. Mossakowski, P. Hoffman, S. Autexier, and D. Hutter. CASL proof calculus. *In [19]*, pages 273–359, 2004.

[19] P. D. Mosses (ed.). CASL— *the common algebraic specification language. Reference Manual*, volume 2960 of *Lecture Notes in Computer Science*. Springer, 2004.

[20] G. Rosu and J. Goguen. Composing hidden information modules over inclusive institutions. In O. Owe, S. Krogdahl, and T. Lyche, editors, *From Object-Orientation to Formal Methods. Essays in Memory of Ole-Johan Dahl*, volume 2635 of *Lecture Notes in Computer Science*, pages 96–123. Springer, 2004.

[21] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.

[22] D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.

[23] L. Schröder, T. Mossakowski, A. Tarlecki, P. Hoffman, and B. Klin. Amalgamation in the semantics of CASL. *Theoretical Computer Science*, 331(1):215–247, 2005.