A General Methodology for Internalising Multi-Level Model Typing

Harald König
Inst. for Computer Science
FHDW Hannover
Hannover, Germany
harald.koenig@fhdw.de

Uwe Wolter
Department of Informatics
University of Bergen
Bergen, Norway
uwe.wolter@uib.no

***Abstract*—Multilevel Modelling approaches allow for an arbitrary number of abstraction levels in typing chains. In this paper, a transformation of a multi-level typing chain into a *single all-covering* representing model is proposed. This comprehensive model is of equal size as the most concrete model in the chain and encodes all typing information in its labels, such that the typing chain can completely be restored. This guideline for maintaining multi-level typing chains in respective implementations of multi-level typing environments is based on a categorical equivalence theorem, which we generalize to a more convenient graph-oriented version.**

***Index Terms*—Multi-Level Modeling, Model Typing, Category Theory.**

I. Introduction and Motivation

The framework of Multi-Level Modeling (MLM) [1], [2] extends the traditional Model-Driven Software Engineering 4-layer approach of UML 2 by allowing an arbitrary number of abstraction levels of software artefacts with type-instance-relations between them. This yields a sequence of graphical artefacts each component being an instance of components of adjacent higher levels, a *typing chain*. Amongst the different tool-based approaches for maintaining typing chains and for implementing operations on them, e.g. [3], [4], there are some, which already aim at a common consensus for a formal underpinning. One aspect of such a consensus describes a typing chain no longer as a collection of artefacts being inter-related by typing mappings, but rather treat them as a single artefact (deep characterisation, potency). E.g., [5] provides a formal foundation based on category theory, where operations on models are multi-level coupled model transformations.

Whereas [5] achieves precision and reusability of rule definitions still by inherent multilevelness of domains, we go one step further: We transform a typing chain into a *single all-covering* representing model, in which all typing information is encoded and from which the typing chain can uniquely be reconstructed. By showing how to algorithmically internalise typings into this all-covering artefact on the one hand and reconstruct and unfold the typing chain on the other hand, we provide a guideline for maintaining multi-level typing chains in respective implementations of multi-level typing environments. Concretely, this paper's main innovations are:

- For each typing chain $\mathcal{G}$ we explain, how to calculate an all-covering compressed representation in a single model M, in fact a single graph or graph-like structure, in which the entire typing chain structure is hidden. This model has the same size as the model on the chain's lowest abstraction level w.r.t. the number of nodes and edges. It essentially codes all used typing information along the chain in the labels of the nodes and edges. Thus there is only little increase w.r.t. disk space. Whereas the labeling of the nodes is just a list of all direct typing assignments, the labeling of the edges is such that the source-target-arrangement of the nodes and edges along *all* levels of the typing chain are implicitely coded in the all-covering graph and need not be repeated. Moreover the typing morphisms' operation compatibility is encoded in the all-covering model and is accordingly and automatically rebuilt during reconstruction.
- Correctness of the correspondence between $\mathcal{G}$ and M is based on a fundamental and hence precisely verified categorical equivalence. As a second innovation, we present and prove a novel graph-based formulation of this equivalence, which is much more convenient for its use in practical applications. This graph-based formulation will also be the guidance for our future work in the area.
- Formalisations are to some extent based on a different view on model structures: Besides the usual instance semantics principle[1], we also utilize the interpretation semantics principle[2], cf. [6].

With these innovations, this paper's contributions can be beneficial for any kind of operations on multilevel typed structures, e.g. the management of multilevel typed graph transformations [5], [7]: Instead of working with morphisms between complete typing chains and extending a multi-level typed DPO rule [8] to a diagram in the category of typing chains, we can support multilevel coupled model transformations [5], e.g. computations of pushouts, pushout complements etc, by a reduction to the classical one-level methodology. Such a reduction might also be helpful in model analysis, but presumably not in user management.

Our approach is not yet complete due to the following limitations:

[1]Each object has a type.

[2]Each type is interpreted as a set of valid instances.

- Whereas typing morphisms in typing chains are usually allowed to be partial [7], we still assume direct typings between adjacent model levels to be total mappings. Thus, we consider the present paper as a proof-of-concept – "part one" – of our entire project. Certainly, the goal is to extend the results to partial mappings in a second part. We think, it is very likely that there is a straightforward generalisation, see the short discussion in Sect. VII.
- So far, we did not consider tooling, because we first want to complete the internalisation algorithm also for partial typings (in "part two" of our work), see above.

We also note that the transformation of typing chain $\mathcal{G}$ into model M is a reversible one-to-one operation, only whenever all typings are surjective, i.e. in case of full meta-model footprints. This limitation, however, is very natural, if the entire typing chain shall be encoded in a graph, which is structurally similar to the model on the lowest level of the typing chain. We additionally explain, how to fully reconstruct in case of non-surjectivity with little extra effort.

The paper is organised as follows: After having presented a running example in Sect. II, preliminary topic centered definitions are contained in Sects III and IV. Sect. V presents the internalisation algorithm and the theorem it is based upon (Theorem 5.1), its iteration and hence the transformation from $\mathcal{G}$ to M is described in Sect. VI. We conclude with remarks about related and future work in Sect. VII and provide an appendix (Sect. VIII) for the proof of the main theorem.

II. Running Example

To illustrate all theoretical results, we use an example, which is inspired by a bigger case study presented in [9] and dealing with the definition of behaviour for simple, autonomous robots, see the typing chain in Fig. 1. The most abstract level M_1 prescribes the basic modeling formalism - binary navigable associations from an owner class to a target class. According to M_2, any DSL shall consist of transitions that have preceding and succeeding tasks. One of these languages (M_3) specifies transitions from an initial task to two alternative movements (go forward or go back)[3]. In a concrete process M_4 a forward move succeeds the initial task.

To ease reading, these models are depicted in a concrete syntax in the right column, e.g. transition instances are displayed as grey circles (•), pre and *succ* are differentiated by their font type, colors are used accordingly, in M_4 instantiated elements are written in the usual "instance":"type"-notation.

The typing chain's abstract syntax, i.e. the model's representations according to their respective metamodels (typing), is shown in the left column. The typing mapping between the models is given via blue small rectangles between the adjacent levels, e.g. from in M_3 has type pre in M_2. Edge typings are not explicitely given, but can be derived from the necessary edge-node-compatibilities[4].

We adopt here the *algebraic* view on graphical structures, where a signature (this term will be defined precisely in

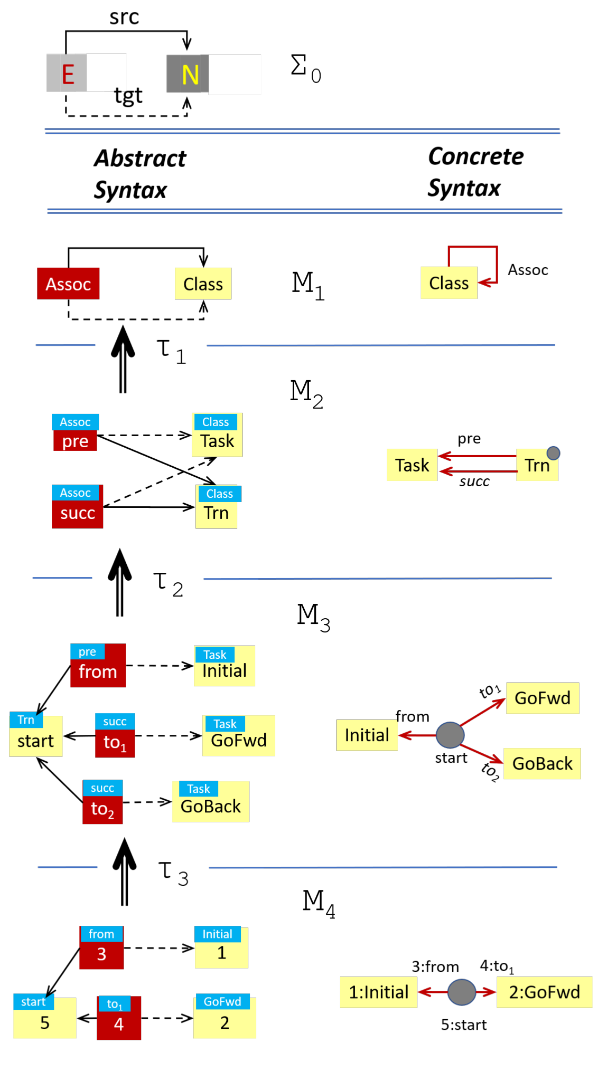

Fig. 1. Typing Chain in Abstract and Concrete Syntax

Sect. IV) guides the structures of all used artefacts (models): In the typing chain $M_4 \stackrel{\tau_3}{\to} M_3 \stackrel{\tau_2}{\to} M_2 \stackrel{\tau_1}{\to} M_1$ in the left column of Fig. 1, all models conform to an overall signature Σ_0 shown in the top of Fig. 1, where E / N are abbreviations for edges and nodes and src / tgt specify that each edge possesses a source and target node. For each model M_i, its set of nodes is coloured yellow, the set of (reified) edges is colored red. The source / target of an edge is the node reached by the outgoing solid / dashed line.[5]

Of course, the forthcoming theoretical results admit an arbitrary guiding structure Σ_0, e.g. so-called "E-Graphs" [8], in which a distinction is made between complex and primitive data types, or even the MOF-model for Ecore[6].

Asking, which objects x of a given sort (node or edge) are present, is unusual in software engineering, because one physically does not assign to a class the set of all its instances

[3] The condition, which decides the alternative, is omitted.

[4] [9] additionally annotate edges with their respective type.

[5] The similarity of model M_1 and signature Σ_0 is discussed in Remark 5.1.

[6] https://www.omg.org/spec/MOF/2.5.1/PDF

(e.g. at runtime). Instead, there is the assignment of a type to an object x (in Java, e.g., by calling x.getClass()). However, the two viewpoints are equivalent, and we will - on a formal level - switch back and forth between them.

III. Preliminaries

For a self-contained study we start with some basic categorical background and some close-by concepts: A category $\mathbb{C}$ is a collection of *objects*, written $|\mathbb{C}|$, and for each pair $A, B \in |\mathbb{C}|$ a set $\mathcal{Mor}_{\mathbb{C}}(A,B)$ of morphisms from A to B. There is the identity morphism $id_A \in \mathcal{Mor}_{\mathbb{C}}(A,A)$ for each $A \in |\mathbb{C}|$ and there is the usual neutral and associative composition operator $\circ : \mathcal{Mor}_{\mathbb{C}}(A,B) \times \mathcal{Mor}_{\mathbb{C}}(B,C) \to \mathcal{Mor}_{\mathbb{C}}(A,C)$. As usual, $f \in \mathcal{Mor}_{\mathbb{C}}(A,B)$ will be written $f : A \to B$ and composition of $f \in \mathcal{Mor}_{\mathbb{C}}(A,B)$ and $g \in \mathcal{Mor}_{\mathbb{C}}(B,C)$ is written $g \circ f$.

We omit the precise definition of the term "collection", since it is not important for the forthcoming considerations. We only note that each set is also a collection, but not vice versa, see [10] for further details[7].

A special category arises, if one fixes an object $T \in |\mathbb{C}|$ and considers morphisms $\tau : M \to T$ for varying objects M. This so-called *comma category* $\mathbb{C} \downarrow T$ has objects these morphisms, i.e. objects M typed over T. Morphisms of $\mathbb{C} \downarrow T$ are type compatible $\mathbb{C}$-morphisms: If $\tau : M \to T$ and $\tau' : M' \to T$ are two typed structures, a type compatible morphism is any $f : M \to M'$, for which $\tau' \circ f = \tau$.

A functor $F : \mathbb{C} \to \mathbb{D}$ consists of two mappings, one for mapping objects (of $\mathbb{C}$ to $\mathbb{D}$), the other for mapping morphisms, such that identities and composition are preserved. We say that two categories $\mathbb{C}$ and $\mathbb{D}$ are *equivalent*, written

$$\mathbb{C} \cong \mathbb{D}$$

if there is a functor $F : \mathbb{C} \to \mathbb{D}$, which can essentially be inverted, i.e. there is a functor $F^{-1} := G : \mathbb{D} \to \mathbb{C}$, such that the composites $F \circ G$ and $G \circ F$ are isomorphically related to identities (on $\mathbb{C}$ and $\mathbb{D}$, resp.): $G(F(C))$ and $F(G(D))$ must not be equal, but are isomorphic to C, D, resp.[8]

IV. Definitions

Diagrammatic models in Software-Engineering are based on graphs. Hence, on a higher (mathematical) level, we need a specifying concept for the modeling domain in its entirety. To distinguish between this (meta-)level and the (software) modeling domain, in which graphs are also present (as software artefacts), we call this concept *Meta-Graphs*, cf. [6]:

Definition 4.1 (Meta-Graphs and Homomorphisms): A *meta-graph* $G = (N^G, E^G = (E^G_{n \to n'})_{n,n' \in N_G})$ consists of a collection N^G of **N**odes and for each pair (n, n') of nodes a set of **E**dges $E^G_{n \to n'}$ from n to n'. I.e. all edges are partitioned w.r.t. their source- and target-nodes n and n'.

Whenever the partitioning w.r.t. source- and target-nodes is clear from the context, we will write the edges of a meta-graph as the union of the collection of the edge-sets:

$$E^G := \bigcup_{(n,n') \in N^G \times N^G} E^G_{n \to n'}$$

A *homomorphism* $\phi : G \to H$ between two meta-graphs $G = (N^G, E^G)$ and $H = (N^H, E^H)$ is a pair (ϕ_N, ϕ_E) of mappings $\phi_N : N^G \to N^H$ and $\phi_E : E^G \to E^H$ with the edge-node-incidence-condition

$$\forall n, n' \in N^G \text{ and } e \in E^G_{n \to n'} : \phi_E(e) \in E^H_{\phi_N(n) \to \phi_N(n')}$$

i.e. source and target of the image of the edge e must coincide with the image of source and target of e, resp.

To simplify reading, we omit subscripts N and E in ϕ_N and ϕ_E and write for both of them just ϕ, if the differentiation becomes clear from the context.

It is not forbidden for a meta-graph to contain an infinite collection of nodes. Hence we can consider the meta-graph whose nodes are finite sets and whose edges are the mappings between these sets:

Definition 4.2 (The Meta-Graph of Finite Sets): Let

$$\mathcal{Set} = (N^{\mathcal{Set}}, E^{\mathcal{Set}})$$

where $\quad N^{\mathcal{Set}} = \{X \mid X \text{ is a finite set}\} \quad$ and

$$E^{\mathcal{Set}} = (E^{\mathcal{Set}}_{X \to Y} = \{f : X \to Y \mid f \text{ is a total map}\})_{X,Y \in N^{\mathcal{Set}}}.$$

Considering only finite sets is no restriction, because, in Software Engineering, involved sets like the set of all types (in a data- or class-model) are always finite.[9]

Our goal in this section is to provide a formal description of models in software engineering with the help of *meta-graphs* and *algebraic specifications*. To keep the elaboration simple, we will, however, not work with additional constraints such as multiplicities[10], such that an edge always specifies an arbitrary (binary) relation between source and target nodes.

Algebraic signatures [12] in its classical form rely on functions, i.e. they don't know about arbitrary relations. E.g. the signature of natural numbers consist of one sort s, a unary operation inc $: s \to s$ and the constant zero (an operation without parameters). The signature is intended to inductively describe the set $\mathbb{N}$ of natural numbers: Constant zero is interpreted as the number 0 and operation inc as function $f : \mathbb{N} \to \mathbb{N}$, defined by $f(x) := x + 1$.

In the sequel, we will consider signatures with several sorts, but only unary operations. Its (semantic) interpretation is then a set X_s for each sort s and for each operation $op : s \to s'$ a *total function* $f : X_s \to X_{s'}$. To capture arbitrary binary relations (as in M_2) with algebraic specifications we work with *reification*, which is a recurring technique: E.g., in UML modeling this means to break the relationship between two

[7]The category of sets and mappings has as objects a proper collection, because there is no such thing as the set of all sets, see e.g. [10].

[8]To understand the forthcoming results, it is not necessary to delve deeper into these definitions, the interested reader may consult [10] or [11].

[9]It can even be shown that there is no such thing as the set of all finite sets, cf. footnote 7, which justifies necessity of nodes in a meta-graph to be a *collection* rather than a set.

[10]E.g. for model M_2 to prescribe precedence arrangements of concrete transitions and tasks, multiplicity constraints should be used.

$$
\begin{tikzcd}
pre \arrow[r, "targetOfPre"] \arrow[dr, "sourceOfPre" near end] & Task \\
succ \arrow[ur, "targetOfSucc" near start] \arrow[r, "sourceOfSucc"'] & Trn
\end{tikzcd}
$$

Fig. 2. Signature Σ'_2 for Process Modelling

classes down into two functional relationships by creating an association class with projections (total functions) to the two classes. Similarly, arbitrary relations between tables in a relational database can always be implemented with junction tables, their foreign keys representing functional dependencies. Hence, by reification, each data model with many-to-many relationships can be transformed into a model, which only specifies functions between types.

Example 4.1: The signature Σ_0 in the top of Fig. 1 contains sorts N and E (nodes and edges) and unary operations $src, tgt : E \to N$. An interpretation is then a set of nodes and a set of edges and two functions, which assign to each edge its source and target node. All models M_1-M_4 in abstract syntax are interpretations of Σ_0, relations (arrows in the concrete syntax) being already reified.

Example 4.2 (Signatures as Meta-Graphs): We can also specify a signature Σ'_2, which contains basic concepts for process modelling by requiring sorts Task and Trn (for Transition) together with operations that assign to a transition preceding and succeeding tasks, cf. Fig. 2. pre and succ are (reified) sorts and functions (source/target)Of(Pre/Succ) assign source and target to them.

It is important *not to mix up* this signature with model M_2 in Fig.1 (an interpretation of Σ_0)! Note, e.g. that there is no labeling of edges in M_2. In the sequel, we will, however, precisely elaborate on their commonalities, and we will justify the deliberately chosen name Σ'_2.

It is remarkable that the signature in Example 4.2 *is* a meta-graph: We obtain nodes

$$N^{\Sigma'_2} = \{\mathsf{pre}, \mathsf{succ}, \mathsf{Task}, \mathsf{Trn}\}$$

from the sorts. Furthermore, from the operations, we obtain edges $E^{\Sigma'_2}$ with edge sets

$$
\begin{aligned}
E^{\Sigma'_2}_{\mathsf{pre}\to\mathsf{Task}} &= \{\mathsf{targetOfPre}\} \\
E^{\Sigma'_2}_{\mathsf{pre}\to\mathsf{Trn}} &= \{\mathsf{sourceOfPre}\} \\
E^{\Sigma'_2}_{\mathsf{succ}\to\mathsf{Task}} &= \{\mathsf{targetOfSucc}\} \\
E^{\Sigma'_2}_{\mathsf{succ}\to\mathsf{Trn}} &= \{\mathsf{sourceOfSucc}\}
\end{aligned}
$$

and $E^{\Sigma'_2}_{n\to n'} = \emptyset$ for the remaining 12 edges sets.

This example justifies the following definition:

Definition 4.3 (Signature): A signature Σ is a meta-graph with nodes the set of sorts of Σ. For each pair of sorts s and s' the set $E^{\Sigma}_{s\to s'}$ is the set of all operations $op : s \to s'$ with domain s and codomain s'.

In the sequel, we will use the terms *sort* / *node* as well as the terms *operation* / *edge* synonymously, but use either one of the alternatives depending on whether we speak of signatures or meta-graphs.

As in Example 4.1, a general signature Σ specifies models by assigning a concrete set to each sort / node, and by assigning a function between these sets to each operation / edge. Moreover, the definition of interpretations enforces the function for an edge $e \in E^{\Sigma}_{n\to n'}$ in the signature to have domain / codomain the interpretation of n / n'. This function-set-incidence-condition justifies the following definition:

Definition 4.4 (Σ-Model): A *model* M that conforms to signature Σ, called a Σ-*Model*, is a homomorphism

$$M : \Sigma \to \mathcal{S}et.$$

between meta-graphs Σ and $\mathcal{S}et$. M is also said to *conform to* signature Σ. We call set $M(n)$ and for each $e \in E^{\Sigma}_{n\to n'}$ function $M(e) : M(n) \to M(n')$ the *interpretations* or *instantiations* of nodes n and edge e in model M.[11]

Thus M_1-M_4 in Fig. 1 are rather *homomorphisms* from Σ_0 to $\mathcal{S}et$ and we say that the mapping of function $M(e)$ establishes an e-link from $x \in M(n)$ to $M(e)(x)$. More vividly, "$M(e)(x)$ is the e-property of x", e.g. Class is the src(-property) of Assoc in M_1, start is the src of from in M_3.

Of course, models alone are not enough to cope with important disciplines in the multifaceted world of model-driven engineering, e.g. model composition, model differencing, model repair, and other sorts of operations on models. I.e. we also have to consider interrelations between models: *Model homomorphisms*. In the spirit of natural transformations in category theory, model homomorphisms must be understood as a family of mappings, one for each sort in the signature, with the usual compatibility conditions:

Definition 4.5: Let Σ be a signature and M and M' be two Σ-models. A (Σ-)*model homomorphism*

$$\alpha : M \Rightarrow M'$$

is a family

$$(\alpha_s : M(s) \to M'(s))_{s\in N^{\Sigma}}$$

of mappings, each of them being a mapping between the two interpretations of sort s in models M and M', such that

$$\forall s_1, s_2 \in N^{\Sigma}, \forall e \in E^{\Sigma}_{s_1\to s_2} : M'(e) \circ \alpha_{s_1} = \alpha_{s_2} \circ M(e),$$

see Fig. 3. I.e. for each element x in the interpretation of sort s_1 in M, the e-linked element of x's α-image in M' equals the α-image of x's e-linked element, i.e. model homomorphisms are compatible with all specified operations.

The compatibility condition is very natural: E.g. a mapping α from model M to model M' both conformimg to signature Σ'_2 (see Fig. 2) must map the target task of a precedence p in M to the target task of $\alpha_{\mathsf{pre}}(p)$ in M', thus preserving the precedence arrangements of M after mapping it to M'.

[11]In Algebraic Specifications $M(n)$ is called the *carrier set* of sort n and $M(e)$ the implementation of operation e. The term "model" for a meta-graph homomorphism from M to $\mathcal{S}et$ has frequently been used, e.g. in [11].

$$
s_1 \xrightarrow{e} s_2 \qquad
\begin{tikzcd}
M(s_1) \arrow[r, "M(e)"] \arrow[d, "\alpha_{s_1}"'] & M(s_2) \arrow[d, "\alpha_{s_2}"] \\
M'(s_1) \arrow[r, "M'(e)"] & M'(s_2)
\end{tikzcd}
$$

Fig. 3. Model Homomorphism Compatibility

$$
\begin{tikzcd}
\Sigma_0 \arrow[d, "M_2"'] & E \arrow[r, shift left, "src"] \arrow[r, shift right, "tgt"'] \arrow[d, mapsto] & N \arrow[d, mapsto] \\
\mathcal{S}et & \{\mathsf{pre}, \mathsf{succ}\} \arrow[r, shift left, "M_2(src)"] \arrow[r, shift right, "M_2(tgt)"'] & \{\mathsf{Task}, \mathsf{Trn}\}
\end{tikzcd}
$$

Fig. 4. Meta-graph homomorphism $M_2 : \Sigma_0 \to \mathcal{S}et$

We obtain an important mathematical object, which is the central artefact for our considerations in the next sections[12]:

Definition 4.6 (Category of Σ-Models): The category $\mathcal{M}od(\Sigma)$ has objects all Σ-models and morphisms Σ-model homomorphisms. An identity is the family of identical mappings, composition is componentwise composition of the mappings in the respective families.

V. Internalising Model Typings

For the Σ_0-models in Fig. 1 it is necessary to classify elements in lower models by elements in the next higher level, i.e. there are model homomorphisms τ_i, which define the typing from one level to the next higher level, see the blue rectangles in the top left corner of their nodes. E.g. $\tau_1 : M_2 \Rightarrow M_1$ with e.g. $\tau_1(\mathsf{pre}) = \mathsf{Assoc}$, $\tau_1(\mathsf{Task}) = \mathsf{Class}$.

In general, a model M is typed in model T, both conforming to an arbitrary signature Σ, i.e. there are (meta-graph-)homomorphisms $M, T : \Sigma \to \mathcal{S}et$ and a Σ-model homomorphism

$$\tau : M \Rightarrow T.$$

In the sequel, we show, how we can *internalise* the typing into model M by accepting an extended signature for it.

A. From $\mathcal{M}od(\Sigma)$-Model T to Extended Signature $\mathcal{G}r(\Sigma, T)$

In the sequel, for nodes of Σ we use letter s to indicate that these nodes are sorts of the base signature, and we use *op* for edges (operations). Recall that $T(s)$ is the interpretation of sort s in model T. For any edge *op* in Σ with source s_1 and target s_2, $T(op) : T(s_1) \to T(s_2)$ assigns to each $t \in T(s_1)$ an appropriate linked element in $T(s_2)$. To emphasise the fact that elements of $T(s)$ are types of the typing model T, we use letter t for these elements. Then we define the meta-graph $\mathcal{G}r(\Sigma, T)$ as follows[13]:

- $N^{\mathcal{G}r(\Sigma,T)} := \{(t\!:\!s) \mid t \in T(s)\}$
- $E^{\mathcal{G}r(\Sigma,T)}_{(t:s)\to(t':s')} := \{(t\!:\!op) \mid op \in E^{\Sigma}_{s\to s'}, t' = T(op)(t)\}$

Whereas Σ-model T consists of sets $T(s)$, an element t of such a set is now converted into a node of meta-graph $\mathcal{G}r(\Sigma, T)$ containing its sort s in its new label. And for each $op \in E^{\Sigma}_{s_1\to s_2}$, the pairs $(t, T(op)(t))$ of input and output elements of $T(op)$ for $t \in T(s_1)$ are made explicit as edges in the new graph $\mathcal{G}r(\Sigma, T)$, a manifestation of map elements.

Example 5.1: Σ_0-model M_2 of Fig. 1 is a meta-graph homomorphism $M_2 : \Sigma_0 \to \mathcal{S}et$, which is shown in Fig. 4.

The mappings are $M_2(src) = \{(\mathsf{pre} \mapsto \mathsf{Trn}), (\mathsf{succ} \mapsto \mathsf{Trn})\}$, $M_2(tgt) = \{(\mathsf{pre} \mapsto \mathsf{Task}), (\mathsf{succ} \mapsto \mathsf{Task})\}$. We obtain $\mathcal{G}r(\Sigma_0, M_2) = (N^{\mathcal{G}r(\Sigma_0,M_2)}, E^{\mathcal{G}r(\Sigma_0,M_2)})$ with nodes

$$N^{\mathcal{G}r(\Sigma_0,M_2)} = \{(pre\!:\!E), (succ\!:\!E), (Task\!:\!N), (Trn\!:\!N)\}$$

and non-empty edge sets

$$
\begin{aligned}
E^{\mathcal{G}r(\Sigma_0,M_2)}_{pre:E\to Trn:N} &= \{(pre\!:\!src)\} && (\mathsf{Trn\ is\ src\ of\ pre}) \\
E^{\mathcal{G}r(\Sigma_0,M_2)}_{succ:E\to Trn:N} &= \{(succ\!:\!src)\} && (\mathsf{Trn\ is\ src\ of\ succ}) \\
E^{\mathcal{G}r(\Sigma_0,M_2)}_{pre:E\to Task:N} &= \{(pre\!:\!tgt)\} && (\mathsf{Task\ is\ tgt\ of\ pre}) \\
E^{\mathcal{G}r(\Sigma_0,M_2)}_{succ:E\to Task:N} &= \{(succ\!:\!tgt)\} && (\mathsf{Task\ is\ tgt\ of\ succ})
\end{aligned}
$$

This signature is depicted in the top of Fig.5.

B. From T-typed model M to a single $\mathcal{G}r(\Sigma, T)$-model

The second step of the construction will now interpret $\tau : M \Rightarrow T$ as a single model $\overline{M}$ in $\mathcal{M}od(\mathcal{G}r(\Sigma, T))$, thus internalising the typing τ between two models M and T into a single model $\overline{M}$ by accepting an extended signature for it. For this we have to define a meta-graph homomorphism $\overline{M} : \mathcal{G}r(\Sigma, T) \to \mathcal{S}et$. Recall model homomorphism τ to consist of a family $(\tau_s : M(s) \to T(s))_{s\in N^{\Sigma}}$ and recall that $\overline{M}$ must map any node $(t\!:\!s)$ to some set and any edge $(t\!:\!op)$ to a mapping between two of these sets. We let $\overline{M}$ be defined

- …on nodes of $\mathcal{G}r(\Sigma, T)$: $\overline{M}((t\!:\!s)) := (\tau_s)^{-1}(t)$
- …on edges of $\mathcal{G}r(\Sigma, T)$: For edge $(t\!:\!op) \in E^{\mathcal{G}r(\Sigma,T)}_{(t:s)\to(t':s')}$ we define the mapping

  $$\overline{M}(t\!:\!op) : (\tau_s)^{-1}(t) \to (\tau_{s'})^{-1}(t)$$

  by

  $$\xi \mapsto M(op)(\xi).$$

Before we check whether this definition of $\overline{M}$ is really a homomorphism, let's investigate the construction along our running example:

Example 5.2: We continue example 5.1. Let M_3 and M_2 be the Σ_0-model(s) in Fig. 1, then there is the typing homomorphism $\tau_2 : M_3 \Rightarrow M_2$. The above construction yields the

[12] The reader may recall the definitions in Sect. III.

[13] We denote the result of the construction with $\mathcal{G}r(\_,\_)$ in honour of Alexander **Gro**thendieck, who invented this construction for categories.

$$
\begin{tikzcd}
\mathcal{G}r(\Sigma_0, M_2) \arrow[dddd, "\overline{M}"'] & (pre\,{:}\,E) \arrow[rr, "pre:tgt"] \arrow[drr, "pre:src"' near end] & & (Task\,{:}\,N) \\
& (succ\,{:}\,E) \arrow[urr, "succ:tgt" near start] \arrow[rr, "succ:src"'] & & (Trn\,{:}\,N) \\
& & \arrow[d, mapsto] & \\
& & {} & \\
\mathcal{S}et & \{\mathsf{from}\} \arrow[rr, "\overline{M}(pre:tgt)"] \arrow[drr, "\overline{M}(pre:src)"' near end] & & \{\mathsf{Initial}, \mathsf{GoFwd}, \mathsf{GoBack}\} \\
& \{\mathsf{to}_1, \mathsf{to}_2\} \arrow[urr, "\overline{M}(succ:tgt)" near start] \arrow[rr, "\overline{M}(succ:src)"'] & & \{\mathsf{start}\}
\end{tikzcd}
$$

Fig. 5. Translated model $\mathcal{G}r(\Sigma_0, M_2) \to \mathcal{S}et$

$\mathcal{G}r(\Sigma_0, M_2)$-model in Fig. 5 with the obvious map elements, e.g. $\overline{M}(succ\,{:}\,tgt)$ maps $\mathsf{to}_1 \mapsto \mathsf{GoFwd}, \mathsf{to}_2 \mapsto \mathsf{GoBack}$.

Proposition 5.1: $\overline{M}$ is a meta-graph homomorphism.

Proof: We have to check the edge-node-incidence-condition of Def. 4.1. Let for this two nodes $(t\,{:}\,s)$ and $(t'\,{:}\,s')$ and an edge $(t : op) \in E^{\mathcal{G}r(\Sigma,T)}_{(t:s)\to(t':s')}$ be given. Thus, by the definition of $E^{\mathcal{G}r(\Sigma,T)}_{(t:s)\to(t':s')}$ in Sect. V-A,

$$t' = T(op)(t). \tag{1}$$

We have to verify

$$\overline{M}(t\,{:}\,op) \overset{?}{\in} E^{\mathcal{S}et}_{(\tau_s)^{-1}(t)\to(\tau_{s'})^{-1}(t')}.$$

i.e. we must show that $\overline{M}(t\,{:}\,op)$ is a function from $(\tau_s)^{-1}(t)$ to $(\tau_{s'})^{-1}(t')$. Let for this $\xi \in (\tau_s)^{-1}(t)$ be given, i.e.

$$t = \tau_s(\xi) \tag{2}$$

i.e. ξ (in M) is t-typed (in T) and both belong to sort s of signature Σ. By the definition of $\overline{M}(t\,{:}\,op)$, we must show that $M(op)(\xi) \in (\tau_{s'})^{-1}(t')$.

This follows from the model homomorphism compatibility, see Fig. 3, which in the case of $\tau : M \Rightarrow T$ becomes

$$\tau_{s'} \circ M(op) = T(op) \circ \tau_s \tag{3}$$

for all edges $op \in E^{\Sigma}_{s\to s'}$, because then

$$
\begin{aligned}
\tau_{s'}(M(op))(\xi) &= T(op)(\tau_s(\xi)) && \text{by (3)} \\
&= T(op)(t) && \text{by (2)} \\
&= t' && \text{by (1)}
\end{aligned}
$$

i.e. $M(op)(\xi) \in (\tau_{s'})^{-1}(t')$ as desired. ■

We call $\overline{M}$ the *internalisation* of $\tau : M \Rightarrow T$ and write

$$\mathcal{INT}(\tau) := \overline{M}.$$

$$
\begin{tikzcd}
M \arrow[rr, "h"] \arrow[dr, "\tau"'] & & M' \arrow[dl, "\tau'"] & \mapsto & \mathcal{INT}(\tau) \arrow[r, "\mathcal{INT}(h)"] & \mathcal{INT}(\tau') \\
& T & & & &
\end{tikzcd}
$$

Fig. 6. Internalisation $\mathcal{INT} : \mathcal{M}od(\Sigma) \downarrow T \to \mathcal{M}od(\mathcal{G}r(\Sigma, T))$

$\mathcal{INT}$ maps objects of the comma category $\mathcal{M}od(\Sigma) \downarrow T$ (see Sect. III) to objects of $\mathcal{M}od(\mathcal{G}r(\Sigma, T))$. The categorical view, however, demands also an assignment of morphisms, i.e., we have to define $\mathcal{INT}(h)$ for a morphism h in $\mathcal{M}od(\Sigma) \downarrow T$, see Fig. 6. We do this by defining the action of $\mathcal{INT}(h)$ exactly as the one by h: If $\xi \in M(s)$ for some node s in Σ, then let $t = \tau_s(\xi)$ and define $\mathcal{INT}(h)_{(t:s)}(\xi) := h_s(\xi)$. i.e. $\mathcal{INT}(h)$ is now a family of mappings, one for each sort in (the fine-grained) signature $\mathcal{G}r(\Sigma, T)$, whereas h was a collection of mappings, only one for each sort in (the coarse-grained) signature Σ. Because h was a model-homomorphism *and* type-compatible as a morphism of the comma category, $\mathcal{INT}(h)$ is also compatible with operations, thus

$$\mathcal{INT} : \mathcal{M}od(\Sigma) \downarrow T \to \mathcal{M}od(\mathcal{G}r(\Sigma, T))$$

becomes a functor.

The following theorem is the main result of the paper:

Theorem 5.1: $\mathcal{INT}$ is an equivalence of categories, i.e.

$$\mathcal{M}od(\Sigma) \downarrow T \cong \mathcal{M}od(\mathcal{G}r(\Sigma, T)).$$

Proof: See Appendix Sect. VIII-B ■

The equivalence property in the theorem has an important consequence, which can be illustrated along Example 5.2: From the Σ_2-model in Fig. 5, in which the original Σ_0-structure (sorts E and N and src / tgt-structure) has at first sight vanished, we can nevertheless *fully and uniquely reconstruct* the typing $\tau_2 : M_3 \Rightarrow M_2$ and the mapping behaviour of homomorphisms $M_3, M_2 : \Sigma_0 \to \mathcal{S}et$ just by applying $\mathcal{INT}^{-1}$.

Note also the similarity of signature Σ_2 for the translated model with the signature Σ_2' in Fig. 2. Whereas we had added the labels of the edges in Σ_2' according to common sense from the structures in Fig. 1, are names now determined according to the described algorithm of Sections. V-A and V-B yielding the same information content.

For future purposes, we conclude this section with some results that are easily derivable from Theorem 5.1:

Definition 5.1 (Terminal Object): In any category $\mathbb{C}$, a *terminal object* U is an object with the following property: For each object $M \in \mathbb{C}$ there is exactly one morphism $M \to U$.

Remark 5.1 (Terminal Models): In Fig. 1, M_1 is a terminal object in $\mathcal{M}od(\Sigma_0)$: Since the interpretations of nodes and edges are singletons, resp., there is only one way of mapping the interpretations of any Σ_0-model M to the interpretations of M_1, namely each element in $M(N)$ is mapped to Class and each element in $M(E)$ is mapped to Assoc.

Since $M_1(E)$ and $M_2(N)$ are singletons the diagrammatic representations of the signature Σ_0 and the terminal Σ_0-model M_1 in Fig. 1 become nearly identical. In other words: We can consider M_1 as an internal representation of the "linguistic meta-model" Σ_0.

The following statements are well-known [12]:

Proposition 5.2: A terminal object in $\mathcal{Mod}(\Sigma)$ is given by interpreting each node as singleton set and each operation as the only possible function between the corresponding sets.

Proposition 5.3: Let $\mathbb{C}$ be a category with terminal object U, then $\mathbb{C} \cong \mathbb{C} \downarrow U$.

We obtain

Corollary 5.1 (of Theorem 5.1): $\mathcal{INT}(id_T : T \Rightarrow T)$ is a terminal object in $\mathcal{Mod}(\mathcal{Gr}(\Sigma, T))$.

Proof: Because $id_T^{-1}(t)$ is a singleton, it is easy to see that $\mathcal{INT}(id_T)$ assigns to each node of $\mathcal{Gr}(\Sigma, T)$ a singleton set. Thus the result follows from Prop. 5.2. ∎

VI. From Typing Chain to Single Graph and Back

Let

$$M_n \xrightarrow{\tau_{n-1}} M_{n-1} \xrightarrow{\tau_{n-2}} \cdots \xrightarrow{\tau_2} M_2 \xrightarrow{\tau_1} M_1 \qquad (4)$$

be a general typing chain, where M_i are Σ_0-models for some initial signature Σ_0. In this section, we use the results of the previous parts to collapse the *complete* typing chain into a single model (more concrete: a single meta-graph), which encodes all higher typing levels and the whole typing structure, and from which the typing chain can be reconstructed.

A. Iterating the Internalisation

Let Σ_0 be some initial signature,

$$M_i^0 := M_i \ (1 \leq i \leq n), \tau_i^0 := \tau_i \ (1 \leq i \leq n-1),$$

then we define recursively

$$\Sigma_i := \mathcal{Gr}(\Sigma_{i-1}, M_i^{i-1}), \qquad (5)$$

see Sect. V-A for the definition of $\mathcal{Gr}(\_,\_)$, and

$$\mathcal{INT}_i : \mathcal{Mod}(\Sigma_{i-1}) \downarrow M_i^{i-1} \to \mathcal{Mod}(\Sigma_i)$$

for all $i \in \{1, \ldots, n\}$, where $\mathcal{INT}_i$ is the functor from Theorem 5.1 with $T := M_i^{i-1}$. All models M_j^{i-1} $(i < j \leq n)$ are directly or indirectly typed over M_i^{i-1}, because for each i we have composed typing morphisms

$$\tau_{ij} := (\tau_i^{i-1} \circ \tau_{i+1}^{i-1} \circ \cdots \circ \tau_{j-1}^{i-1} : M_j^{i-1} \to M_i^{i-1})_{i<j\leq n},$$

which are objects of the comma category $\mathcal{Mod}(\Sigma_{i-1}) \downarrow M_i^{i-1}$ and for which we define

$$M_j^i := \mathcal{INT}_i(\tau_{ij}), \tau_j^i := \mathcal{INT}_i(\tau_j^{i-1}) \qquad (6)$$

for all $1 \leq i < j \leq n$, the first assignment being on objects, the second on morphisms of $\mathcal{Mod}(\Sigma_{i-1}) \downarrow M_i^{i-1}$ (note that $\tau_j^{i-1} : \tau_{i(j+1)} \to \tau_{ij}$ is a morphism in $\mathcal{Mod}(\Sigma_{i-1}) \downarrow M_i^{i-1}$), thus establishing a new shortened typing chain in $\mathcal{Mod}(\Sigma_i)$.

In fact, the initial typing chain (4) in $\mathcal{Mod}(\Sigma_0)$ is the input for the iteration:

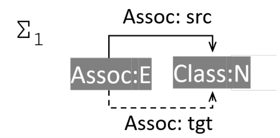

Fig. 7. Signature $\Sigma_1 = \mathcal{Gr}(\Sigma_0, M_1^0)$

- $M_n^0 \xrightarrow{\tau_{n-1}^0} M_{n-1}^0 \xrightarrow{\tau_{n-2}^0} \cdots \xrightarrow{\tau_2^0} M_2^0 \xrightarrow{\tau_1^0} M_1^0$

All these models are directly or indirectly typed over M_1^0, i.e. we obtain objects of $\mathcal{Mod}(\Sigma_0) \downarrow M_1^0$. They are mapped by $\mathcal{INT}_1$ to

- $M_n^1 \xrightarrow{\tau_{n-1}^1} M_{n-1}^1 \xrightarrow{\tau_{n-2}^1} \cdots \xrightarrow{\tau_2^1} M_2^1$

i.e. for $i = 1$, the chain is shortened by the first internalisation on the right: $M_2^1 = \mathcal{INT}_1(\tau_{12}) = \mathcal{INT}_1(\tau_1^0)$ and $\tau_2^1 := \mathcal{INT}_1(\tau_2^0)$ by (6) with $j = 2$. Furthermore, M_j^1 and τ_j^1 for $j > 2$ are calculated accordingly.

Iterating further over index i $(i \geq 2)$ yields after iteration $i = n-2$ (by mapping with $\mathcal{INT}_{n-2}$)

- $M_n^{n-2} \xrightarrow{\tau_{n-1}^{n-2}} M_{n-1}^{n-2}$,

in $\mathcal{Mod}(\Sigma_{n-2})$, then after iteration $i = n-1$ (by mapping with $\mathcal{INT}_{n-1}$)

- M_n^{n-1}

in $\mathcal{Mod}(\Sigma_{n-1})$.

We can even go one step further and artificially add $\tau_n := id_{M_n}$ in the beginning of the original typing chain (4), i.e. $M_{n+1} := M_n$. We can then apply a last iteration round $(i = n)$, which produces the terminal object in $\mathcal{Mod}(\Sigma_n)$ by Corollary 5.1. By Prop. 5.2, this terminal object has singletons for all sorts of signature Σ_n, i.e. *it is fully represented by* Σ_n itself. We obtain the

- **Final result**: Σ_n.

The lines starting with "•" demonstrate the successive collapse of the typing chain into the single meta-graph Σ_n. Of course, the transformation of the typing chain in (4) to Σ_n is invertible, because it is composed of invertible transformations $\mathcal{INT}_1, \ldots, \mathcal{INT}_n$ by Theorem 5.1.

Example 6.1: Let's demonstrate everything along our running example: $\Sigma_1 := \mathcal{Gr}(\Sigma_0, M_1)$ is depicted in Fig. 7. Because M_1 is the terminal object by Prop. 5.2, Σ_1-models M_2^1, M_3^1, M_4^1 are structurally identical to M_2, M_3, M_4 in Fig. 1, the only difference being the renaming of nodes and their typings in M_j^1, which now have suffix Assoc or Class. Additionally, M_2^1 is no longer typed, since τ_1 has been internalised.

$\Sigma_2 := \mathcal{Gr}(\Sigma_1, M_2^1)$ together with the remaining models is shown in Fig. 8. Sorts in the signature are colored grey, operations and their mapping behavior in the models are distinguished by different colors. Elements of the sets in the models are positioned according to the positioning of the sorts in the signature, see the grey rectangles, e.g. all elements in M_3^2, which belong to sort Task : Class : N are grouped together in the top right corner of M_3^2, in this case the elements

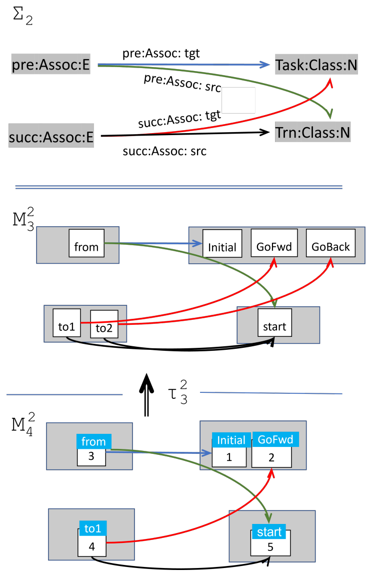

Fig. 8. Signature $\Sigma_2 = \mathcal{G}r(\Sigma_1, M_2^1)$ and Σ_2-models

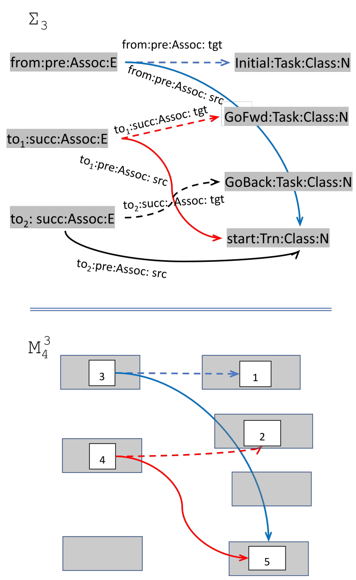

Fig. 9. Signature $\Sigma_3 = \mathcal{G}r(\Sigma_2, M_3^2)$ and Σ_3-models

Initial, GoFwd, GoBack. Note the similarity with Fig.5, where elements are grouped in the same way, but the intermediate step via M_1 is skipped, thus omitting the terms Assoc and Class in the names.

Finally, the construction of Σ_3 and internalisation of τ_3^2 yields the Σ_3-model in Fig.9, where we observe the occurence of two empty sets. The last step according to the above algorithm then yields signature Σ_4 in Fig.10, in which the part of the chain is coded, which is needed to reconstruct the direct and indirect typings of the contained elements (Corollary 6.1 below provides a criterion to completely restore the chain).

Note that it is nowhere necessary to store the src-tgt-arrangement of all higher levels. This arrangement is encoded

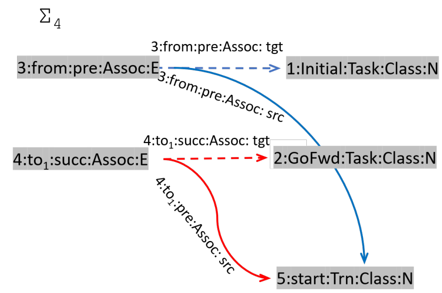

Fig. 10. Signature $\Sigma_4 = \mathcal{G}r(\Sigma_3, M_4^3)$

in the labeling of the sorts and operations of Σ_n, e.g. in Σ_4 by the suffixes of the labels.

B. Reconstruction

The most important application of Theorem 5.1 is the ability to unfold the final result Σ_n in order to reconstruct the complete typing chain, including its labeling and the inner structure of models on all levels: Let M_j^i be a Σ_i-model $(j > i)$, then the sorts in Σ_i yield sets

$$N := \{s_2 : \cdots : s_{i+1} \mid s_1 : s_2 : \cdots : s_{i+1} \in N^{\Sigma_i}\}$$

and

$$E := \{s_2 : \cdots : s_i : op \mid s_1 : s_2 : \cdots : s_i : op \in E^{\Sigma_i}, op \in E^{\Sigma_0}\}$$

which together with corresponding domain and codomain of the latter make up a signature, which faithfully restores sorts and operations of Σ_{i-1}. In contrast, the cut off sorts s_1 of N^{Σ_i} constitute the model M_i^{i-1} together with the appropriate linkings. These sorts are simultaneously the respective typings of the models M_j^{i-1} on lower levels $j > i$. In such a way, e.g., Σ_2 in Fig. 8 and Σ_2-models are reverted back to Σ_1-models M_4^1, M_3^1, M_2^1 together with their (possibly composed) typings.

It is easy to see from the described reconstruction methodology, that the following corollary of Theorem 5.1 holds:

Corollary 6.1: Let a typing chain be given as in (4), where all typing morphisms τ_i are surjective. Let Σ_n be the final result of the iterated internalisation. Then the typing chain can completely be reconstructed from Σ_n.

It is no serious restriction to claim surjectivity of the τ_i, because this just means that there is at least one instance in model M_{i+1} of type t (located in model M_i) for each t and each $i \in \{1, \ldots, n\}$.

Consider, as an example, the typing chain of Fig.1 without the lowest level, i.e. $M_3 \overset{\tau_2}{\to} M_2 \overset{\tau_1}{\to} M_1$. The algorithm terinates with Σ_3 in the top of Fig. 9, from which the chain can be restored, because both τ_1 and τ_2 are surjective.

We constructed our example, such that we can also demonstrate the effects in case of non-surjectivity. In that case, the algorithm reconstructs only those types that are actually instantiated (needed) in the typing chain. Consider for this again the complete chain in Fig. 1: to_2 vanishes during internalisation, because it is not used as a type in M_4.

To *fully* reconstruct the typing chain in case of non-surjectivity, it is necessary to store, for each i, those sorts s of Σ_i, for which $M^i_{i+1}(s) = \emptyset$ together with all operations with domain or codomain s. E.g. in Fig. 9 $\mathsf{to_2 : succ : assoc : E}$ of Σ_3 and the two operations $\mathsf{to_2 : succ : assoc : tgt/src}$ must be stored during the next internalisation step.

Consequently, an implementing tool (e.g. for multi-level coupled model transformations [5]) must only maintain meta-graph Σ_n and possibly not needed types. All higher level models and their intermediate typings are coded in Σ_n's labels and structure.

VII. Related and Future Work

A. Additional Related Work

As already pointed out in the introduction, reasoning about implementations of multi-level modeling structures is always accompanied by *internalisation techniques* and by construing typing chains as single entities, where especially the elements of the models are similarly labelled as in our approach. This has been carried out in connection with DSLs [13], definitions of model behavior [5], coupled model transformations and multilevel graph transformations [7], [9], but also when improving reuse opportunities of language families [14] or embedding a one-level situation into a multi-level environment [15]. Different ways of internalisations of model typings and typing chains have also been used in several tools, e.g. MetaDepth [3], Melanee [16], or AtomPM [17].

Categorical foundations for understanding relations between objects as entities in their own right are comma- or arrow categories [18], but especially appear in the concepts of profunctors, graphs of a functor [19], cartesian closedness [11], and - the background of Theorem 5.1 - the Yoneda Embedding and the Grothendieck Construction [18]. Practical approaches for depicting typing chains as a single artefact exist, but these transformations are only reversible, if the typing relation is captured by (e.g. OCL-) constraints [20].

B. Future Work

As mentioned before, we plan to extend the result to *partial typing morphisms* in a continuation of the present paper. We decided not to include partiality in the present paper ("part one"), because we expect certain extra effort due to different formulations of composition (of partial mappings) and node-edge-incidences along a typing chain, cf. [9]. Moreover, in a final stage of extension, constraints and inheritance must be included in the theory.

Reification can be avoided, if carriers of algebras are arbitrary relations. It may be worth investigating a corresponding generalisation. This, however, poses new questions about composition and (co-)completeness, which complicates the theory significantly.

References

[1] C. Atkinson, "Meta-modeling for distributed object environments," in *1st International Enterprise Distributed Object Computing Conference (EDOC '97), 24-26 October 1997, Gold Coast, Australia, Proceedings*. IEEE Computer Society, 1997, p. 90. [Online]. Available: https://doi.org/10.1109/EDOC.1997.628350

[2] C. Atkinson and T. Kühne, "Processes and products in a multi-level metamodeling architecture," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 11, no. 6, pp. 761–783, 2001. [Online]. Available: https://doi.org/10.1142/S0218194001000724

[3] J. de Lara and E. Guerra, "Deep meta-modelling with metadepth," in *Objects, Models, Components, Patterns, 48th International Conference, TOOLS 2010, Málaga, Spain, June 28 - July 2, 2010. Proceedings*, ser. Lecture Notes in Computer Science, J. Vitek, Ed., vol. 6141. Springer, 2010, pp. 1–20. [Online]. Available: https://doi.org/10.1007/978-3-642-13953-6_1

[4] C. Atkinson and R. Gerbig, "Flexible deep modeling with melanee," in *Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband*, ser. LNI, S. Betz and U. Reimer, Eds., vol. P-255. GI, 2016, pp. 117–122. [Online]. Available: https://dl.gi.de/20.500.12116/843

[5] F. Macías, U. Wolter, A. Rutle, F. Durán, and R. Rodríguez-Echeverría, "Multilevel coupled model transformations for precise and reusable definition of model behaviour," *J. Log. Algebraic Methods Program.*, vol. 106, pp. 167–195, 2019. [Online]. Available: https://doi.org/10.1016/j.jlamp.2018.12.005

[6] U. Wolter and Z. Diskin, "From indexed to fibred semantics – the generalized sketch file –," Dep. of Informatics, University of Bergen, Reports in Informatics 361, 2007.

[7] U. Wolter, F. Macías, and A. Rutle, "Multilevel typed graph transformations," in *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*, ser. Lecture Notes in Computer Science, F. Gadducci and T. Kehrer, Eds., vol. 12150. Springer, 2020, pp. 163–182. [Online]. Available: https://doi.org/10.1007/978-3-030-51372-6_10

[8] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, *Fundamentals of Algebraic Graph Transformations*. Springer, 2006.

[9] F. Macías, A. Rutle, V. Stolz, R. Rodríguez-Echeverría, and U. Wolter, "An approach to flexible multilevel modelling," *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, vol. 13, pp. 10:1–10:35, 2018. [Online]. Available: https://doi.org/10.18417/emisa.13.10

[10] S. Mac Lane, *Categories for the Working Mathematician, Second edition*. Springer, 1998.

[11] M. Barr and C. Wells, *Category theory for computing science*. Prentice Hall, 1990.

[12] W. Wechler, *Universal Algebra for Computer Scientists*. Springer-Verlag Berlin, Heidelberg, 1992.

[13] F. Macías, "Multilevel modelling and domain-specific languages," *CoRR*, vol. abs/1910.03313, 2019. [Online]. Available: http://arxiv.org/abs/1910.03313

[14] J. de Lara and E. Guerra, "Multi-level model product lines - open and closed variability for modelling language families," in *Fundamental Approaches to Software Engineering - 23rd International Conference, FASE 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, ser. Lecture Notes in Computer Science, H. Wehrheim and J. Cabot, Eds., vol. 12076. Springer, 2020, pp. 161–181. [Online]. Available: https://doi.org/10.1007/978-3-030-45234-6_8

[15] S. P. Jácome-Guerrero and J. de Lara, "*TOTEM*: Reconciling multi-level modelling with standard two-level modelling," *Comput. Stand. Interfaces*, vol. 69, p. 103390, 2020. [Online]. Available: https://doi.org/10.1016/j.csi.2019.103390

[16] C. Atkinson and R. Gerbig, "Flexible deep modeling with melanee," in *Modellierung 2016 - Workshopband*, S. Betz and U. Reimer, Eds. Bonn: Gesellschaft für Informatik e.V., 2016, pp. 117–121.

[17] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. V. Mierlo, and H. Ergin, "Atompm: A web-based modeling environment," in *Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), Miami, USA, September 29 - October 4, 2013*, ser. CEUR Workshop Proceedings, Y. Liu, S. Zschaler, B. Baudry, S. Ghosh, D. D. Ruscio, E. K. Jackson, and M. Wimmer, Eds., vol. 1115. CEUR-WS.org, 2013, pp. 21–25. [Online]. Available: http://ceur-ws.org/Vol-1115/demo4.pdf

[18] S. Awodey, *Category Theory*, 2nd ed. USA: Oxford University Press, Inc., 2010.

[19] J. Bénabou, "Les distributeurs," Université Catholique de Louvain, Institut de Mathématique Pure et Appliquée, rapport 33, 1973.

[20] M. Gogolla, J.-M. Favre, and F. Büttner, "On squeezing M0, M1, M2, and M3 into a single object diagram," Proceedings Tool-Support for OCL and Related Formalisms-Needs and Trends, Tech. Rep., 2005.

VIII. Appendix

In this section, we eleborate the technical details behind the main theorem in Sect. V-B, Thm. 5.1. Statements without further explanations or explicit reference can be found in [10].

A. From Meta-Graphs to Categories

In the sequel, we call meta-graphs just graphs. Categories can be interpreted as graphs enriched with identities, composition and equationally constrained w.r.t. neutrality and associativity. Each graph G can canonically be transformed into a category $\mathbb{P}(G)$, which has as objects the nodes of G and as arrows the paths in G. A path

$$p = (n_0, e_1; \cdots ; e_k)$$

for some $k \geq 0$ in G consists of a node n_0 together with a sequence $e_i \in E^G_{n_{i-1} \to n_i}$ ($i \in \{1, \ldots, k\}$) of consecutive edges with n_0 being the source of e_1. Identity morphisms are all empty paths ($k = 0$) at n: $id_n := (n, [])$. Composition in $\mathbb{P}(G)$ is concatenation of paths modulo neutrality (w.r.t. empty paths) and associativity (of concatenation). In the literature $\mathbb{P}(G)$ is also called the *path category* of graph G.

There is the reverse construction U, which assigns to a category $\mathbb{C}$ its (underlying) graph G by defining the collection of nodes of G to be the objects of $\mathbb{C}$ and edges its morphisms. The monoidal structure is "forgotten", i.e. identities become (meaningless) loops at all nodes, and composition is no longer known. Additionally, we can convert a graph homomorphism $T : G \to \mathcal{S}et$ to a functor $T^* : \mathbb{P}(G) \to \mathcal{SET}$ (where $\mathcal{SET}$ is now the category of sets and total mappings such that $\mathcal{S}et$ becomes its underlying graph). This is simply done by defining $T^* = T$ on objects and

$$T^*(p) = T(e_k) \circ \cdots \circ T(e_1)$$

for a path $p = (n_0, e_1; \cdots ; e_k)$ and $k \geq 0$.

Since in our setting these graphs are always signatures, we use letter Σ instead of G from now on. Furthermore, we denote with $\mathcal{SET}^{\mathbb{C}}$ the category, which has objects the functors from a category $\mathbb{C}$ to $\mathcal{SET}$ and whose morphisms are the natural transformations between them [11].

An important theorem in category theory states that $\mathbb{P}$ and U extend to functors (between the category of graphs and the category of categories) and that the assignment $T \mapsto T^*$ is a bijection between graph homomorphisms from Σ to $\mathcal{S}et$ on the one hand and objects of $\mathcal{SET}^{\mathbb{P}(\Sigma)}$ on the other hand, which can be shown to extend to an equivalence of categories between Σ-models and the category of $\mathcal{SET}$-valued functors [10]:

Lemma 8.1: For any signature Σ: $\mathcal{M}od(\Sigma) \cong \mathcal{SET}^{\mathbb{P}(\Sigma)}$, the object assignment being realized by $T \mapsto T^*$.

This lemma immediately yields

Lemma 8.2: $\mathcal{M}od(\Sigma) \downarrow T \cong \mathcal{SET}^{\mathbb{P}(\Sigma)} \downarrow T^*$.

The definition of the extended signature in Sect. V-A is based on the so-called *Grothendieck Construction* for sets, cf. [11], which converts any set-valued functor $F : \mathbb{C} \to \mathcal{SET}$ into a functor from a category $\mathcal{G}r(\mathbb{C}, F)$ to $\mathbb{C}$, the latter functor having special fibrational properties. In this essentially invertible construction the category $\mathcal{G}r(\mathbb{C}, F)$ is defined similarly as $\mathcal{G}r(\Sigma, T)$ in Sect. V-A. Identities and composition of $\mathcal{G}r(\mathbb{C}, F)$ are due to the existence of them in $\mathbb{C}$. Note that we overloaded the operator $\mathcal{G}r(\_,\_)$ corresponding to the type of arguments (signature and model in Sect.V-A, category and functor in the Grothendieck Construction). Hence $\mathcal{G}r(\mathbb{C}, F)$ has

- objects $\{(x\!:\!C) \mid x \in F(C), C \in |\mathbb{C}|\}$ and
- morphisms $\{(x : op) : (x : C) \to (x' : C') \mid op \in \mathcal{M}or_{\mathbb{C}}(C, C'),\ x' = F(op)(x)\}$

cf. the definition of $\mathcal{G}r(\Sigma, T)$ in Sect. V-A. Because $\mathcal{G}r(\mathbb{C}, F)$ is the disjoint union of all elements in $F(C)$ for all $C \in |\mathbb{C}|$, it is called the *category of elements* (of F).

B. The Proof of Theorem 5.1

For the proof we need two auxiliary results:

Lemma 8.3: For any homomorphism $T : \Sigma \to \mathcal{S}et$:

$$\mathcal{G}r(\mathbb{P}(\Sigma), T^*) \cong \mathbb{P}(\mathcal{G}r(\Sigma, T))$$

Proof: The category of elements of the path category of Σ w.r.t. functor T^* has the same objects as the path category of $\mathcal{G}r(\Sigma, T)$, see the above definitions. I.e. we can define a functor $\varphi : \mathcal{G}r(\mathbb{P}(\Sigma), T^*) \to \mathbb{P}(\mathcal{G}r(\Sigma, T))$ to be identical on objects. If $(x : p)$ is a morphism in $\mathcal{G}r(\mathbb{P}(\Sigma), T^*)$ with p a morphism in $\mathbb{P}(\Sigma)$, i.e. $p := (n_0, e_1; \cdots ; e_k)$ is a path in Σ and $x \in T(n_0)$, we define

$$\varphi(x\!:\!p) := ((x\!:\!n_0), ((x_0\!:\!e_1); (x_1\!:\!e_2); \cdots ; (x_{k-1}\!:\!e_k)))$$

where $x_0 := x$ and $x_i := T(e_i)(x_{i-1}), i = 1..k$, if the path length $k > 0$, which yields a path in $\mathcal{G}r(\Sigma, T)$. For $k = 0$, we map the identity morphism $id_{(x:n_0)}$ to the empty path at $(x : n_0)$ in $\mathbb{P}(\mathcal{G}r(\Sigma, T))$. It is easy to see that these definitions yield a faithful and full (injective and surjective on each set $\mathcal{M}or(\_,\_)$ of morphisms) functor, which is essentially surjective on objects. These three properties are known to be necessary and sufficient for φ being an equivalence of categories.[14] ■

The second auxiliary result is the main ingredient of our result and can be found in [18], Lemma 9.23.:

Lemma 8.4: [Awodey, 2005] For signature Σ and functor $F : \mathbb{P}(\Sigma) \to \mathcal{SET}$:

$$\mathcal{SET}^{\mathbb{P}(\Sigma)} \downarrow F \cong \mathcal{SET}^{\mathcal{G}r(\mathbb{P}(\Sigma), F)}$$

Proof of Theorem 5.1: The various equivalences stated so far yield

$$\begin{aligned}
\mathcal{M}od(\Sigma) \downarrow T &\cong \mathcal{SET}^{\mathbb{P}(\Sigma)} \downarrow T^* && \text{(Lemma 8.2)}\\
&\cong \mathcal{SET}^{\mathcal{G}r(\mathbb{P}(\Sigma), T^*)} && \text{(Lemma 8.4)}\\
&\cong \mathcal{SET}^{\mathbb{P}(\mathcal{G}r(\Sigma, T))} && \text{(Lemma 8.3)}\\
&\cong \mathcal{M}od(\mathcal{G}r(\Sigma, T)) && \text{(Lemma 8.1)},
\end{aligned}$$

where we use the extended signature $\mathcal{G}r(\Sigma, T)$ instead of Σ in Lemma 8.1. ■

[14] For a more detailed definition of these properties and why they characterise equivalences, see [10]