FRAGMENTA: a theory of separation to design fragmented MDE models

Nuno Amálio Juan de Lara Esther Guerra

Abstract

Model-Driven Engineering (MDE) promotes models throughout development. However, models may become large and unwieldy even for small to medium-sized systems. This paper tackles the MDE challenges of model complexity and scalability. It proposes FRAGMENTA, a theory of modular design that allows overall models to be broken down into fragments that can be put together to build meaningful wholes, in contrast to classical MDE approaches that are essentially monolithic. The theory is based on an algebraic description of *models*, *fragments* and *clusters* based on graphs and morphisms. The paper's novelties include: (i) a mathematical treatment of fragments and their joints, called *proxies*, that enable referencing across fragments, (ii) FRAGMENTA's *fragmentation strategies*, which prescribe a fragmentation structure to model instances, (iii) FRAGMENTA's support for both top-down and bottom-up design, and (iv) our formally proved result that shows that inheritance hierarchies remain well-formed (acyclic) globally when fragments are composed provided some local fragment constraints are met.

Contents

Chapter 1

Introduction

The construction of large software systems entails issues of complexity and scalability. Model-Driven Engineering (MDE) emphasises design; it raises the level of abstraction by making models the primary artifacts of software development. The goal is to master and alleviate the complexity of software through abstraction; however, models' sizes can be overwhelmingly large and complex even for small to medium-size systems, impairing comprehensibility and complicating the refinement of models into running systems [KRM$^+$13].

This paper presents FRAGMENTA, a mathematical theory that tackles the complexity and scalability challenges of modern day MDE. FRAGMENTA is based on the ideas of *modularity* and *separation of concerns* [Par72, TOHSMS99]; it allows an overall model to be broken down into *fragments* that are organised around *clusters*. A fragment is a smaller model, a sub-model of an ensemble constituting the overall model. FRAGMENTA is a modular approach that supports both top-down and bottom-up ways of building bigger fragments from smaller ones that covers both the instance and type perspectives of models (also known as models and metamodels). The FRAGMENTA theory presented here uses *proxies*, which act as the *seams* or *joints* of fragments and enable referencing across fragments; this mimics a similar mechanism of the popular EMF [SBPM08].

The primary goal of FRAGMENTA is to provide a mathematical theory of MDE model fragmentation that is formally verified and validated, offering a firm and rigorous foundation for implementations of the theory as part of MDE languages, frameworks and tools. FRAGMENTA builds upon the algebraic theory of graphs and their morphisms. The theory's inherent complexity was tackled with the aid of formal languages and tools, namely: the Z language and its CZT typechecker, and the Isabelle proof assistant [NPW02]. All formal proofs undertaken to validate and verify the theory were done in Isabelle.

1.1 Contributions

The paper's contributions are as follows:

- A mathematical theory of model fragments and the associated seaming mechanism of proxies, which mimics a similar mechanism used in practice [SBPM08]. To our knowledge, this particular combination together with a study on the particularities of proxies, is missing in similar works.

- A formal treatment of the meta-level notion of fragmentation strategies, which is, to our knowledge, missing in other theories such as ours.

- The formally proved result that our local fragment constraints ensure that the resulting compositions will be inheritance cycle free, a fundamental well-formedness property of object-oriented inheritance, precluding the need for global checks.
- A theory of incremental definition, based on proxies, that supports both bottom-up and top-down design. To our knowledge, this has not been emphasised before; Fragmenta's top-down concept of continuation is novel, as far we know.
- Fragmenta's three-level architecture: local fragment, global fragment and cluster, which is, to our knowledge, absent in previous works.

1.2 Outline

This chapter introduces the report. The subsequent chapters and appendices of this report are as follows:

- Chapter 2 gives an overview of Fragmenta, presenting the chapter's running examples.
- Chapter 3 introduces the base graphs upon which Fragmenta theory is built, in particular, structural graphs (SGs) to capture MDE structural models .
- Chapter 4 describes the basis of Fragmenta's models based on fragments and clusters.
- Chapter 5 presents Fragmenta's model composition approach based on the colimit construction of category theory.
- Chapter 6 introduces Fragmenta's approach to typing and metamodel-defined fragmentation strategies.
- Chapter 7 discusses the results of the paper, chapter 8 discusses related work and chapter 9 concludes the paper.
- Appendix A presents the mathematical definitions that complement the main text, which, resorts, essentially, to either informal or less rigorous mathematical definitions.
- Appendix B presents the complete Z specification of Fragmenta's theory.

Chapter 2

Fragmenta in a Nutshell

FRAGMENTA is a theory to design fragmented models. Its goal is to enable the construction of model fragments that can be processed and understood in isolation and put together to make consistent and meaningful bigger fragments; an overall model is a collection of fragments. FRAGMENTA's primitive units are *fragments*, *clusters* and *models*:

- A fragment is a graph with *proxy* nodes that act as *seams* or *joints*; proxies are surrogates that represent some other element of some fragment.
- Clusters are containers to put related fragments together. They enable hierarchical organisation: a cluster may contain other clusters and fragments.
- A model is a collection of fragments organised with clusters. This enables fragmentations that mimic modern programming projects; in implementations, fragments may be deployed as files and clusters as folders.

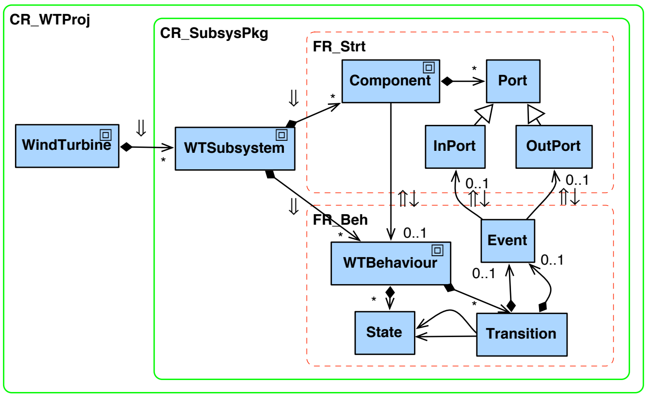
Fragmentation strategies (FSs) are metamodel annotations that stipulate a fragmentation structure to model instances. FRAGMENTA supports both top-down and bottom-up fragmented designs based on imports and continuations, which although related are different:

- If a fragment B imports or continues a fragment A, it means, in both cases, that B may have proxies that reference elements of A.
- A fragment to be continued is deferred; its completion rests upon the fragments that continue it; this gives top-down because it is like defining the root of a tree that is continued in the leafs (continuations).
- A fragment that imports others, on the other hand, gives bottom-up, because defining a root node involves incrementally building upon the leafs (imported fragments). If we see definition as a tree, then top-down involves going down from the root to the leafs; bottom-up does the reverse.

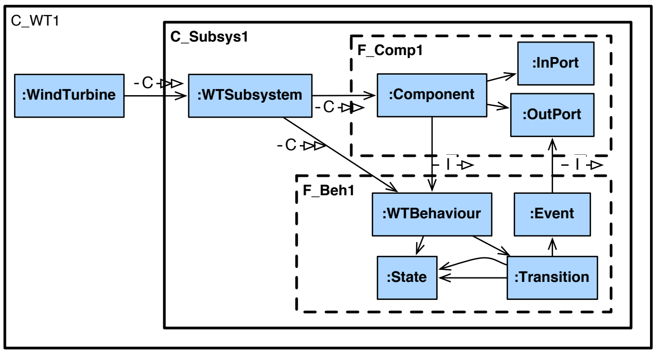
2.1 MONDO Example

Fig. 2.1 presents this paper's running example, based on an industrial language, taken from the MONDO EU project[1]. Fig. 2.1(a) shows a simple meta-model of a language to model software

[1] `http://www.mondo-project.org/`



(a) Metamodel with fragmentation strategy



(b) Overview of fragmented model

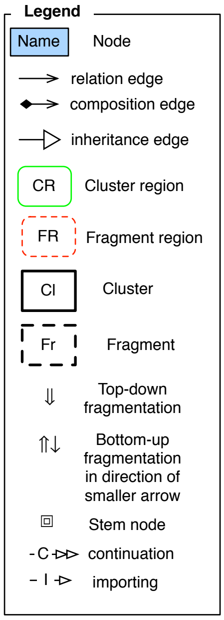
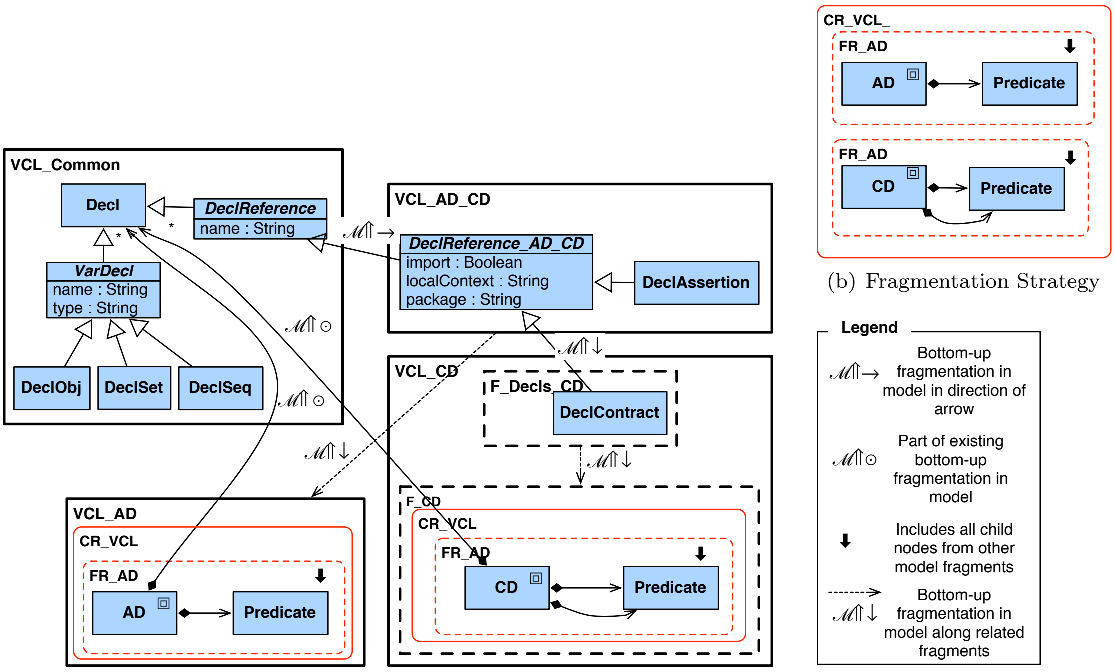


Figure 2.1: Running Example: metamodel with fragmentation strategy and fragmented model instance

controllers for wind turbines (WTs); an abstracted instance model that omits proxy nodes is given in Fig. 2.1(b). WT controllers are organised in subsystems made up of components, containing several input and output ports. A component's behaviour is described by a state machine. The metamodel's FS defines regions (rounded rectangles) of type cluster (solid line) or fragment (dashed line). Related instances of the nodes inside a region must pertain to a corresponding instance-level cluster or fragment.

FS of Fig. 2.1(a) stipulates the following:

- WT models are placed in clusters (cluster region `CR_WTProj`), containing clusters for each subsystem of the modelled WT (region `CR_SubsysPkg`); a subsystem cluster contains a structural and a behavioural fragment (regions `FR_strt` and `FR_beh`, respectively). A region's *stem* node (symbol ⧈) indicates that the creation of its instances entails the creation of the corresponding instance-level cluster or fragment.
- A FS specifies how cross-border associations are to be fragmented. We consider two alternatives: *top-down* (symbol $\Downarrow$) and *bottom-up* (symbol $\Uparrow$). Top-down fragmentations are realised as *continuations*; bottom-up as *importings*. In Fig. 2.1(a), cross-border edges coming out of `WindTurbine` and `WTSubsystem` are top-down; the remaining ones, bottom-up.



(a) Overview of metamodel of VCL declarations

(b) Fragmentation Strategy

Figure 2.2: Simplified metamodel of VCL assertion and contract diagrams, illustrating incremental definition.

The overview instance model of Fig. 2.1(b) (a detailed model is given in Fig. 4.4) complies with its metamodel FS. Top-down edge fragmentation is realised as continuations; bottom-up as importings.

2.2 VCL Example: ADs and CDs

The next example is drawn from the definition of the Visual Contract Language (VCL) [AK10, AKMG10, AGK11, AG14]. VCL assertion diagrams (ADs) and contract diagrams (CDs) have many components in common. Using the modular approach proposed here, we factor the common components into separate fragments, and then build ADs and CDs by composing the common fragments with other parts that are specific to ADs and CDs. Figure 2.2 presents this example, which is based on the metamodel of VCL ADs and CDs.

Figure 2.2 illustrates bottom-up incremental definition. The larger metamodel fragments are built on top of smaller ones through importing mechanisms, where the elements of the smaller fragment become available in the bigger fragment. In the overview metamodel of Fig. 2.2(a), this is described using the $\mathcal{M} \Uparrow$ symbol, which says that there is a bottom-up composition from one fragment to the other in the direction of the second arrow; the actual metamodel with proxy nodes is given in Fig. 4.5. The fragments of Fig. 2.2 are as follows:

- Fragment `F_Decls_Common`, part of `VCL_Common` cluster, describes a metamodel for declaring variables that is common across all diagram types of VCL. It introduces the abstract class `Decl` to represent some declaration, which is subclassed by `VarDecl` and

DeclReference. VarDecl represents a variable declaration; it is subclassed by DeclObj (a scalar variable declaration), DeclSet (a set variable declaration) and DeclSeq (a sequence variable declaration). Class DeclReference represents a reference to other parts of the model, as an AD or a CD.

- Fragment F_Decls_AD_CD, part of the VCL_AD_CD cluster, extends the fragment F_Decls_Common for the purpose of the declarations that are common to CDs and ADs. This introduces the classes Decl_Reference_AD_CD, which represents an assertion or contract, and the class DeclAssertion to represent assertions that are imported and that can be placed on the declarations compartment of ADs or CDs. Class Decl_Reference_AD_CD subclasses DeclReference from fragment F_Decls_Common.

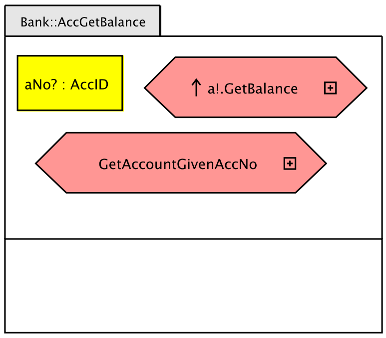
- Fragment F_Decls_CD, part of the VCL_CD cluster, extends F_Decls_AD_CD by introducing the class Decl_Contract, which represents a contract reference that can be placed in the declarations compartment of a CD. Decl_Contract specialises Decl_Reference_AD_CD of fragment F_Decls_AD_CD, which is allowed because Decl_Reference_AD_CD is defined as extensible.

- Fragment F_AD, part of the VCL_AD cluster, defines the metamodel of ADs. It introduces class AD to represent an AD, which contains a set of declarations (class Decl as defined in the fragment F_Decls_AD_CD; this is legal because Decl is made visible in fragment F_Decls_AD_CD. The declarations compartment of AD can, therefore, contain any variable declaration (class VarDecl of F_Decls_Common) and any assertion reference (class DeclAssertion), but not contracts as fragment F_Decls_AD_CD does not include the class Decl_Contract.

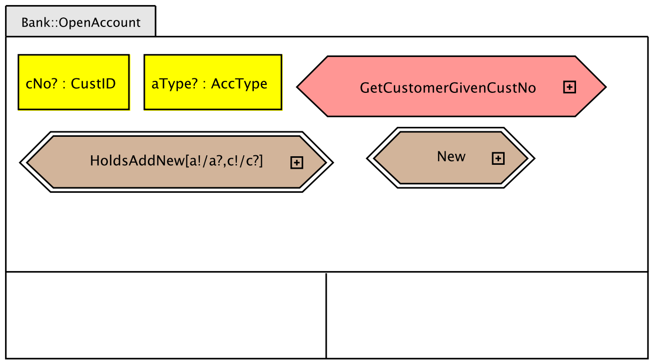
- Fragment F_CD, part of the VCL_CD cluster, defines the metamodel of CDs. It introduces class CD that holds declarations (class Decl as defined in the fragment F_Decls_CD. This means that the declarations compartment of AD can contain any variable declaration (class VarDecl of F_Decls_Common), any assertion reference (class DeclAssertion of fragment F_Decls_AD_CD) and contracts (class Decl_Contract of fragment F_Decls_CD).

In Fig. 2.2(a), the metamodel-defined FS is described using rounded rectangles (in red). This is abstracted in Fig. 2.2(b): the FS defines one cluster region with two fragment regions corresponding to a models's ADs and CDs.

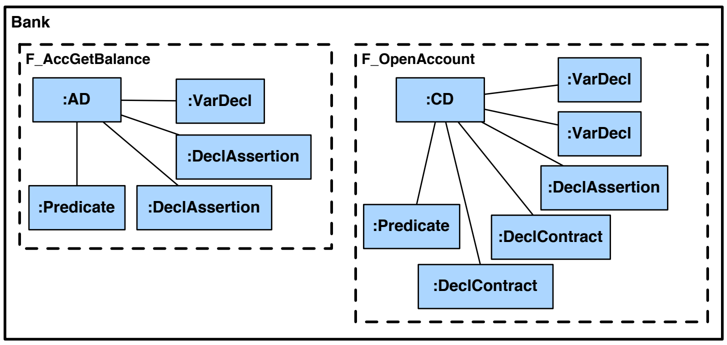
Example model instances, corresponding to the metamodel of Fig. 2.2, are given in Fig. 2.3. This shows the metamodel in action, illustrated with two VCL operations, one described using an AD, and the other using a CD.



(a) VCL Operation `AccGetBalance`



(b) VCL Operation `OpenAccount`



(c) Cluster and Fragments of VCL Operations

Figure 2.3: Model instances corresponding to VCL ADs and CDs

Chapter 3

Graphs as the Foundations of FRAGMENTA

FRAGMENTA's foundations lie on graphs and their morphisms. We present most notions informally and in an intuitive way.

3.1 Notation

In the following, we use the symbol $\mathbb{P}$ to denote a powerset (e.g. $\mathbb{P}\,\mathbb{N}$). The symbol $\leftrightarrow$ denotes a binary relation (e.g. $\mathbb{N} \leftrightarrow \mathbb{N}$), a powerset of a cross-product (e.g. $\mathbb{N} \leftrightarrow \mathbb{N}$ gives $\mathbb{P}(\mathbb{N} \times \mathbb{N})$). The symbol $\rightarrow$ denotes a total function; $\nrightarrow$ denotes a partial function; and $\rightarrowtail$ an injective total function. Whenever possible, given a function f, we write $f\ x$ and not f(x), omitting unnecessary parenthesis.

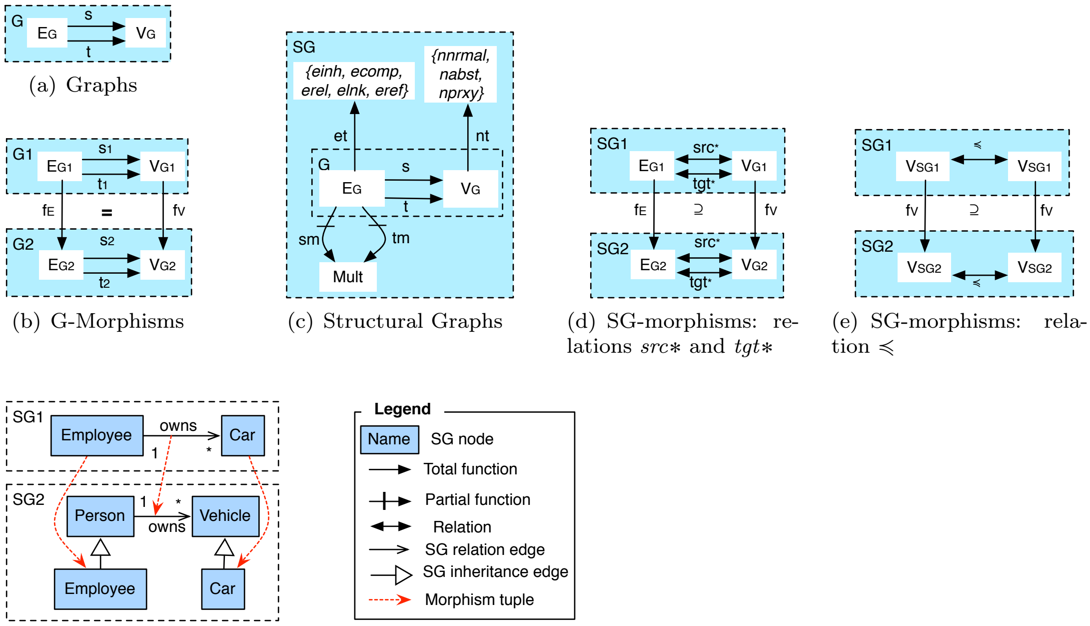
3.2 Graphs and graph morphisms

FRAGMENTA is based on graphs, graph morphisms (G-morphisms) and their composition. We assume sets V and E of all possible nodes and edges of graphs (def. 2). As usual, a graph G, a member of set Gr (def. 3), is made of sets $V_G \subseteq V$ and $E_G \subseteq E$ of nodes and edges, and (total) functions $s, t \colon E_G \rightarrow V_G$ for the source and target of edges (see Fig. 3.1(a)). G-morphisms (def. 5) are made of two functions mapping nodes and edges, and preserving the source and target functions – functions fV and fE depicted in Fig. 3.1(b). Graph morphisms can be composed (def. 6). Graphs and their morphisms form category **Graph** (fact 3).

3.3 Structural Graphs

FRAGMENTA's *structural graphs* (SGs) enrich graphs to support MDE models. SGs capture *conceptual* or *structural* models, such as UML class and entity-relationship diagrams. Typically, such models include:

- families of concepts related through *inheritance*,
- concepts related through *containment*, *whole-part* or *composition* relations,



(a) Graphs

(b) G-Morphisms

(c) Structural Graphs

(d) SG-morphisms: relations $src*$ and $tgt*$

(e) SG-morphisms: relation $\preccurlyeq$

(f) An SG-morphism

Figure 3.1: Graphs, graph morphisms, structural graphs

- and relations between concepts that are subject to multiplicity constraints.

An SG, member of set SGr (def. 11), is a tuple $SG = (G, nt, et, sm, tm)$ (see Fig. 3.1(c)), comprising: (a) a graph $G : Gr$, (b) two colouring functions nt, et giving the kinds of nodes and edges, and (c) two partial multiplicity functions sm, tm to assign multiplicities to the source and target of edges.

SGs support edges of type inheritance (*einh*), composition (*ecomp*), relation (*erel*), link (*elnk*) and reference (*eref*, used by proxies in sec. 4.1). We call *association* edges to edges of type composition, relation and link. All relation and composition edges (and no other) have multiplicities. Inheritance is reified with edges, and we permit dummy self edges (to enable more morphisms), but require the inheritance graph formed by restricting to non-self inheritance edges to be acyclic. SGs' node types are normal (*nnrml*), abstract (*nabst* for abstract classes) and proxy (*nprxy*). Fig. 3.1(f) shows two SGs.

SG-morphisms (def. 13) cater to the semantics of inheritance: if two nodes are inheritance-related, the association edges of the parent become edges of the child. In Fig. 3.1(f), `owns` of `SG2` is also an edge of nodes `Employee` and `Car`. To capture this semantics, we introduce functions src^* and tgt^*, which yield relations $E \leftrightarrow V$ between edges and vertices that extend functions s and t to support the fact that an edge can have more than one source or target node (see def. 11)[1]. The transition from G- to SG-morphisms considers this new set-up: the equality commuting expressed in terms of functional composition (Fig. 3.1(b)) is replaced by subset commuting expressed in terms of relation composition (Fig. 3.1(d)). Likewise, for the actual inheritance relation between nodes, captured by relation $\preccurlyeq$; SG morphisms may shrink

[1]In Isabelle, we proved that src^* and tgt^* preserve the information of base source and target functions; see def. 11.

(removing nodes) or extend (adding nodes) inheritance hierarchies and they should, therefore, preserve the inheritance information, which is described as subset commuting (Fig. 3.1(e)).

At this stage, SG-morphisms disregard the preservation multiplicities and colouring; this is considered as part of typing (see chapter. 6). Structural graphs and their morphisms form category **SGraphs** (Fact 5).

Figure 3.1(f) presents a valid SG-morphism. It is also possible to build a (non-injective) morphism from `SG2` to `SG1` by adding dummy inheritance self-edges to `SG1` (omitted in figures); both morphisms were proved correct in Isabelle.

Chapter 4

Fragmented Models

Figure 4.1 gives a schematic representation of a fragmented model, comprising two clusters and three fragments. It highlights an architecture made up of three layers, local fragment (LF_i), global fragment (F_i) and cluster (C_i), related through morphisms. These layers are explained in the next sections. Figure 4.1 highlights Fragmenta's proxy nodes (grey nodes with solid bold lines), which enable referencing.

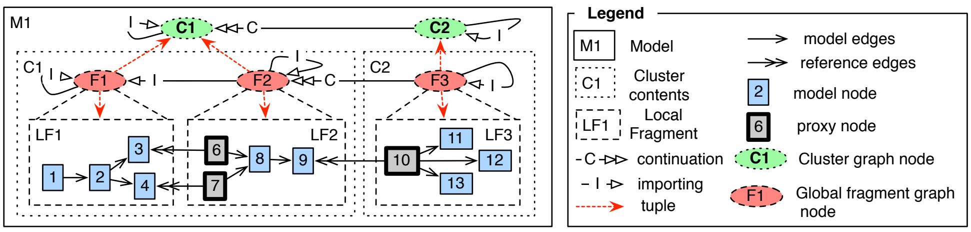


Figure 4.1: Example Fragmenta model (`M1`) made up of two clusters (`C1` and `C2`) and three fragments (`F1`, `F2` and `F3`). A model has three levels: cluster (C_i), global fragment (F_i) and local fragment (LF_i).

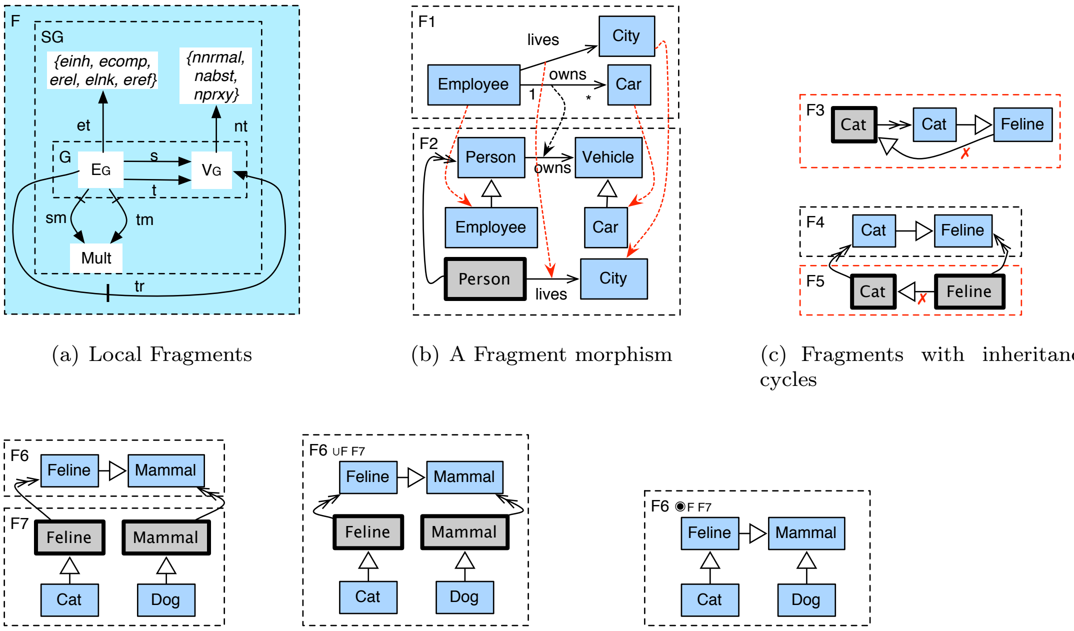
The three levels of FragmentaâĂŹs architecture are as follows:

- *Local Fragment (LFi).* This defines the actual sub-models of an overall Fragmenta model. Each sub-model being a graph with proxy nodes (in grey with solid-bold lines), which refer to nodes defined in other fragments; this reference is depicted using reference edges (double-arrowed lines).

- *Global Fragment (Fi).* This defines the relations between fragments, where each fragment is represented as an atom (dashed red ovals). A fragment can either import (white-triangle arrowhead) or continue (double white-triangle arrowhead) another.

- *Cluster (Ci).* This defines the relations between clusters: each cluster being an atom (pointed green ovals). A cluster can either import, continue or contain another cluster.

Fragmenta's three levels are related in the theory using graph morphisms: (i) A morphism from the global fragment level to the cluster level indicates the assignment of fragments to



(a) Local Fragments

(b) A Fragment morphism

(c) Fragments with inheritance cycles

(d) Two valid fragments

(e) Union fragment composition

(f) Colimit fragment composition

Figure 4.2: Fragments

clusters; (ii) a morphism from the local fragment level to the global fragment level indicates the assignment of local fragment elements to global fragment atoms with reference edges highlighting the inter-fragment relations.

4.1 Fragments

Fragments provide a referencing mechanism, allowing proxy nodes to refer to other nodes, possibly belonging to other fragments. This is realised through reference edges (introduced as part of SGs in chapter 3); in SGs such edges point to themselves – they are *unreferenced*. Fragments complete reference edges by providing their actual targets[1].

A fragment (see Fig. 4.2(a)) is a pair $F = (SG, tr)$, comprising an SG plus a target function for reference edges (def. 14 defines set Fr, $F \in Fr$). Function tr is illustrated in Fig. 4.2: in Fragment $F2$ of Fig. 4.2(b), for instance, proxy node `Person` (thick line) refers to node with same name, likewise for Figs 4.2(c), 4.2(d) and 4.2(e). A referred node may be either in the proxy's fragment or in another one (`F2` in Fig. 4.2(b) contains an intra-fragment reference, and `F5` and `F7` in Figs. 4.2(c) and 4.2(d) contains inter-fragment references). Function `tr` purveys three different fragment representations: (i) a graph with unreferenced references (SG view), (ii) a graph with proxies and their references only, and (iii) the fragment's SG with referred nodes.

Fragmenta forbids inheritance cycles, such as the ones illustrated in Fig. 4.2(c): $F3$ contains an explicit (direct) cycle that is excluded through a constraint that says that the inheritance relation enriched with references must be acyclic, and $F4$ together with $F5$ contain a semantic

[1]Reference edges are kept unreferenced in SGs because SGs require that all nodes pertain to the graph, not allowing references that may be located in other graphs

(indirect) cycle that is excluded by stating that proxy nodes cannot have supertypes – see def. 14 for details. In Isabelle, we proved that our local fragments constraints preclude inheritance cycles both locally and globally (see fact 6 in appendix).

FRAGMENTA uses a form of composition based on the union of fragments as a way to put fragments together without resolving the references (def. 15). This is illustrated in Fig. 4.2(e), which puts together fragments F_6 and F_7 of Fig. 4.2(d). The composition that resolves the references (called *colimit composition*, chapter 5 below) is illustrated in Fig. 4.2(f). The inheritance edges of proxies in Figs. 4.2(d) and 4.2(e) are valid: proxies may not have supertypes, but subtypes are allowed.

Fragment morphisms handle the semantics of reference edges, which is akin to inheritance: an edge attached to a node is an edge of that node and all its representations in the fragment. In fragment `F2` of Fig. 4.2(b), edges `lives` and `owns` pertain to both nodes named `Person`. To support this, fragments extend relations $\prec$, $\preccurlyeq$, src^* and tgt^* of SGs to cover the semantics of references[2]. This extension is based on functions *refs*, which gives the references relation between proxies and their referred nodes (obtained from a restricted graph that considers reference edges only), and function $\rightsquigarrow$, which yields a relation giving all the representatives of a given node ($\rightsquigarrow_F = refs_F \cup (refs_F)^{\sim}$), and the actual inheritance relation for fragments, which extends the inheritance of SGs with the representatives relation ($\prec_F = \prec_{sg\ F} \cup \rightsquigarrow_F$).

The definition of fragment morphisms (def. 16) is similar to SG-morphisms, but taking references into account using the extended relations. In Isabelle, we proved the correctness of the morphism of Fig. 4.2(b) and the one in the inverse direction.

4.2 Global Fragment Graphs

Global fragment graphs (GFGs) represent fragment relations. A GFG (Fig. 4.3(a)) is a pair $GFG = (G, et)$ made of a graph and an edge colouring function, stating whether the edge is an imports or continues (def. 17 introduces set $GFGr$, such that $GFG \in GFGr$). Graph `GFG_MONDO_M` of Fig. 4.4 is an example GFG. We define two sets of morphisms for GFGs:

- GFG-morphisms, which preserve edge-colouring (see def. 18).
- Fragment to GFG morphisms, which maps fragment local nodes to the global fragment nodes to which they belong (see def. 19).

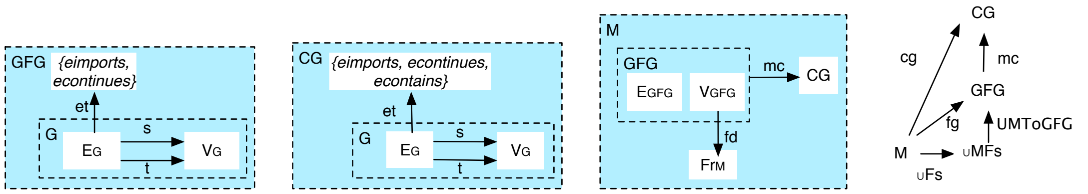
4.3 Cluster Graphs

Fragments are grouped and organised around clusters, FRAGMENTA's hierachical structuring mechanism. A cluster graph (CG) identifies clusters and their relations. As shown in Fig. 4.3(b), a CG is a pair $CG = (G, et)$ made up of a graph G and an edge colouring function et, stating whether the related clusters are in a relation of imports, continues or contains (see def. 20, which defines set CGr, such that $CG \in CGr$). Graph `CG_MONDO_M` of Fig. 4.4 is an example of a CG; likewise for graph `CG_VCL_AD_CD_MM` of Fig. 4.5.

We define two sets of colouring preserving morphisms involving CGs:

- CG-morphisms (see def. 21).
- GFG to CG morphisms (see def. 22).

[2]In Isabelle, we proved that the extensions preserve the information of the corresponding SG relation (e.g $\preccurlyeq_F \subseteq \preccurlyeq_{sg\ F}$); see def. 14.



(a) Global Fragment Graphs (b) Cluster Graphs (c) Models (d) Tower of morphisms of a model

Figure 4.3: Global fragment graphs, cluster graphs and models

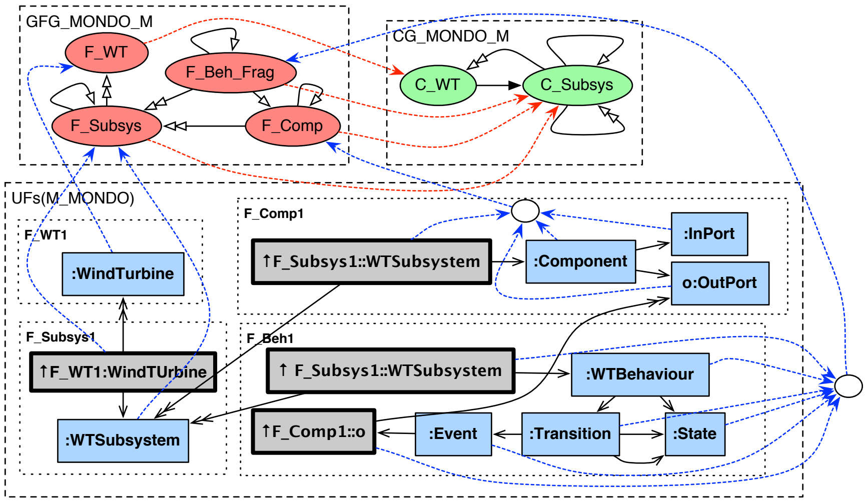


Figure 4.4: FRAGMENTA MONDO model highlighting underlying morphisms. Bottom graph describes union of all fragments of the MONDO model

4.4 Models

A FRAGMENTA model is a collection of fragments. As shown in Fig. 4.3(c), a model is a tuple $M = (GFG, CG, mc, fd)$, comprising a GFG, a CG, a morphism $mc\colon GFG \to CG$, and a function $fd : Ns_{GFG} \to Fr$ mapping nodes of the GFG to fragment definitions (Fr is set of all fragments) – def. 23 introduces set Mdl, $M \in Mdl$. In Fig. 4.3(c), Fr_M is the set of fragments of a model, as given by the range of fd. Each fragment has its own nodes and edges.

As outlined in Fig. 4.1, FRAGMENTA models consist of three inter-related levels. Hence, each model has an underlying tower of morphisms relating these three levels. Fig. 4.3(d) depicts this: from a model M, we can obtain the union of all the model's fragments (function UFs def. 23), and from this we can construct a morphism to the model's GFG (function $UMToGFG$, def. 23), and from here the model's morphism mc gets to the model's CG. Figure 4.4 illustrates this: `M_MONDO` at the bottom is the fragment resulting from function UFs (union of all model fragments).

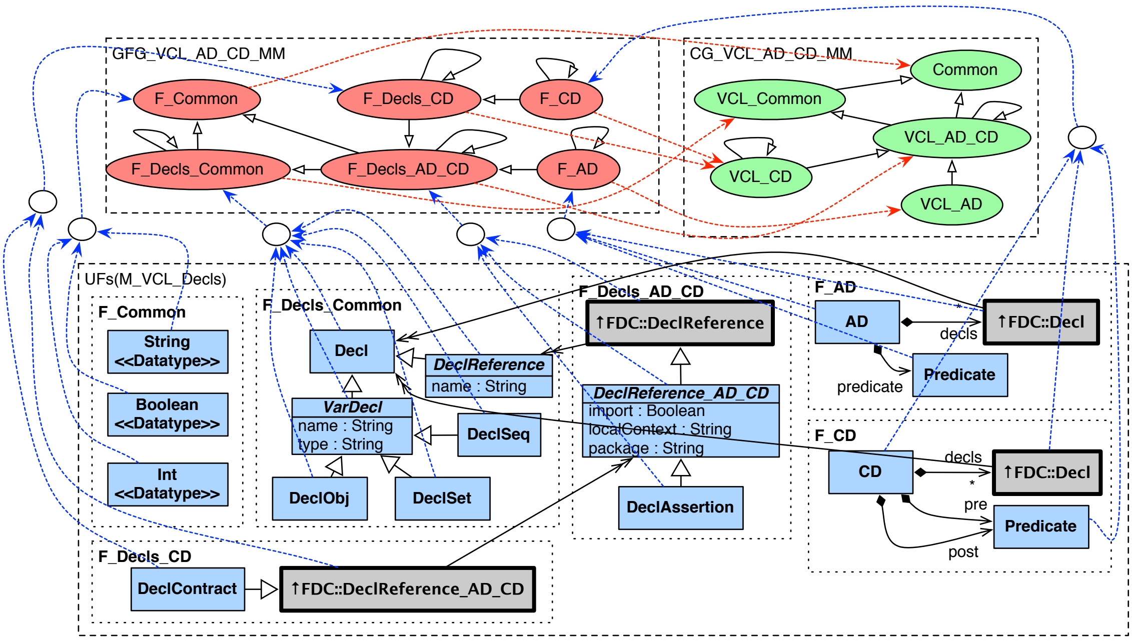


Figure 4.5: Fragmenta VCL model highlighting underlying morphisms

Chapter 5

Model Composition

The previous chapter highlighted FRAGMENTA's overall model built as the union of all fragments (fragment `M_MONDO` in Fig. 4.4). This constitutes a simple form of composition; overall model retains proxy nodes and their references.

This section shows how to compose fragments through a process of reference resolution, where proxy and referred nodes are merged, and the reference edges eliminated. This is based on the *colimit* construction of category theory [Pie91, BW98, Lan71].

5.1 Background: Category Theory

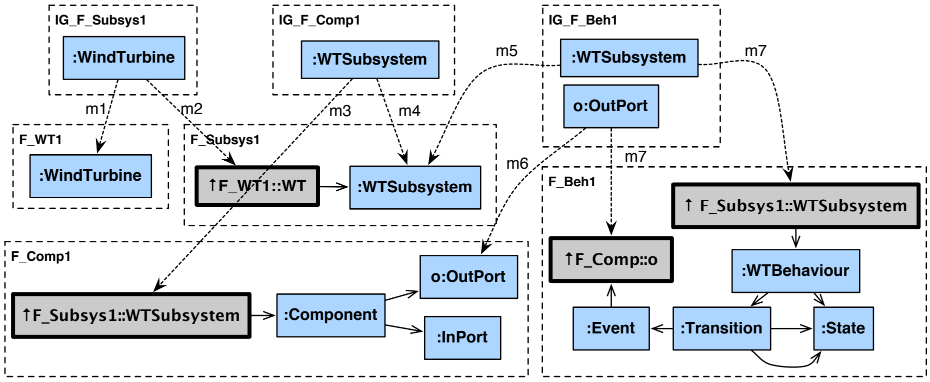
We outline the concepts of category theory that underlie FRAGMENTA's colimit composition.

- In general, a category is a mathematical structure that has objects and morphisms, with a composition operation on the morphisms and an identity morphism for each object [EEPT06]. Categories are formally defined in def. 8.
- FRAGMENTA's colimit composition is a generalisation of the binary *pushout* operator, which we describe in def. 24 to better understand what the more complicated colimit does.
- The concept of a diagram over a category is important for the concept of colimit, a diagram being a graph with a morphism to some category. Morphisms from graphs to categories are defined in def. 25 and actual diagrams are defined in def 26.
- A colimit is a special cocone; these categorical notions are defined in def. 27.

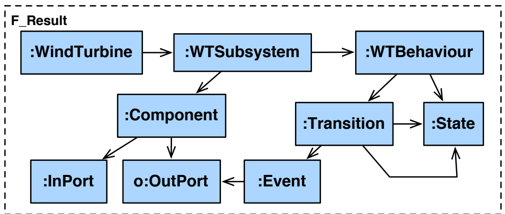
5.2 Colimit composition in FRAGMENTA: overview

Here, we outline the approach using the MONDO example of Fig. 4.4 (whose composition is given in Fig. 5.1(b)):

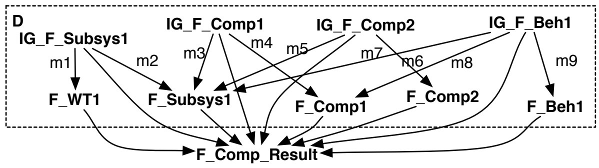
- We construct *interface graphs* (IGs) for each fragment containing proxies only. This is illustrated in Fig. 5.1(a) (graphs named `IG_F...`).
- For each IG, we construct morphisms from the reference edges, using the source and target reference functions of the fragment. In Fig. 5.1(a), we have morphisms that map node `:WT` of `IG_F_Subsys1` to nodes with same name in `F_WT1` and `F_Subsys1` (the target reference and source of corresponding reference edge, respectively).



(a) Machinery of colimit composition



(b) Result of colimit composition



(c) Colimit composition diagram

Figure 5.1: FRAGMENTA's colimit-based composition

- Following this scheme, we build a diagram of IGs and SGs without reference edges corresponding to the fragments being composed as shown in Fig. 5.1(c).
- By applying the colimit to all the graphs behind such a diagram, we obtain a SG without references as shown in Fig. 5.1(b).

To carry out the composition, we first define the diagram that describes the relation between the different fragments and the interface graphs that relate them. This diagram will then allow the specification of the composition based on the co-limit construction of category theory. Definition 28 defines this diagram.

Chapter 6

Typing and Fragmentation Strategies

This section develops FRAGMENTA's approach to the typing between models and metamodels and the compliance to fragmentation strategies (FSs). This section is as follows:

- Our study of typing starts in a monolithic world, where one graph represents the whole model. This is done by resorting to the notion of typed SGs, developed in section 6.1.
- We then move to a world of graphs with proxies by developing the notion of typed fragments (section 6.2).
- Finally, we develop the notion of typing at the level of models and the associated notion of FSs. This is done by developing the notion of a typed model in section 6.3.

6.1 Typed Structural Graphs

Figure 6.1 illustrates the typed SGs that we want to represent, highlighting inheritance and composition.

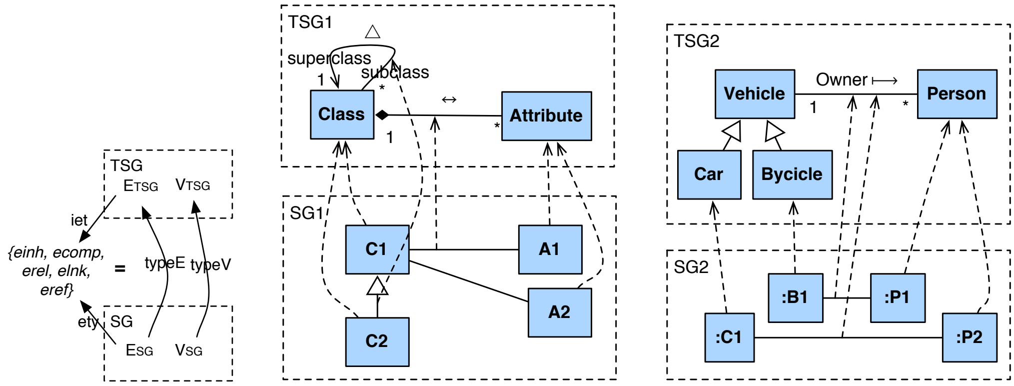
We introduce two structures to represent typing at the level of SGs:

- A type SG is a pair $TSG = (SG, iet)$, comprising a $SG : SGr$ and a colouring funcion $iet : EsA_{SG} \rightarrow SGET$, mapping edges to the type of instance edge being prescribed (def. 29, which defines set $TySGr$, $TSG \in TySGr$).
- A typed SG, depicted in Fig. 6.1(a), is a triple $SGT = (SG, TSG, type)$, consisting of an instance-level SG $SG : SGr$, a type SG $TSG : TySGr$ and a fragment morphism $type : SGr \rightarrow TySGr$, mapping the instance SG to the type one (see def. 30, which defines $SGTy$, such that $SGT \in SGTy$).

We proved in Isabelle that the typed SGs of Fig. 6.1 are valid according to the def.30.

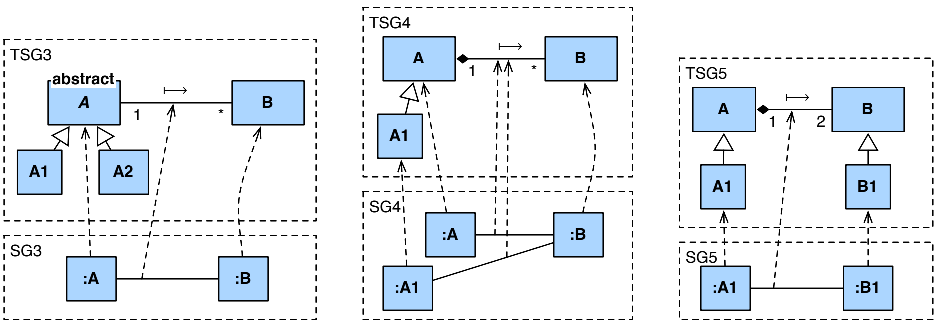
When used as a type, an SG introduces constraints that must be satisfied by its instances; when these constrains are satisfied, we say that the instance *conforms* to its type. The conformance constraints (illustrated in Fig. 6.2) are as follows:

- Edge types of instance SG conform to those prescribed by type SG (commuting of diagram of Fig.6.1(a)).



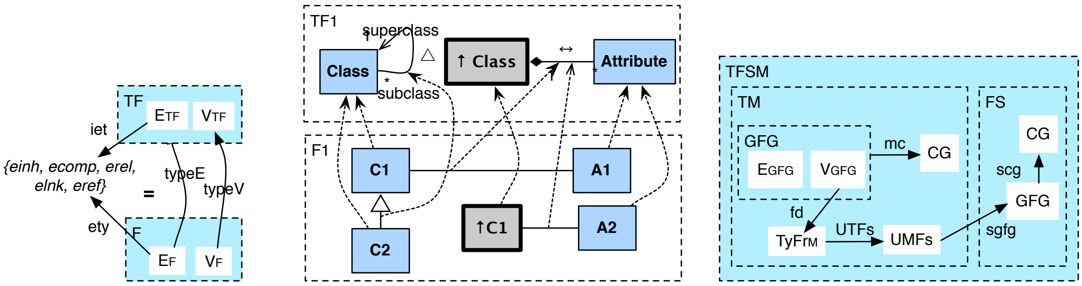
(a) Typed SG structure (b) Typing with inheritance at instance level (c) Typing with inheritance at type level

Figure 6.1: Examples of typed structural graphs, comprising a type graph (top) and an instance graph (bottom). The edges of type graph are decorated with the edge type prescribed to its instances (△ – inheritance; ↔– association; or ◆ – composition).



(a) Invalid Typing: abstract nodes must not have direct instances (b) Invalid Typing: containments must not shared (c) Invalid Typing: multiplicity not satisfied

Figure 6.2: Examples of typed structural graphs involving composition and multiplicity



(a) Typed fragment structure (b) Typed fragment example (c) Structure of type models

Figure 6.3: Typed fragments and typed models. A typed fragment is made of a type fragment and an instance fragment (a). A type fragment is decorated with the prescribed edge type as illustrated in (b): $\triangle$ is inheritance; $\leftrightarrow$ is relation. A type model (c) is a normal model holding type fragments and a fragmentation strategy.

- Abstract nodes may not have direct instances.
- Containments are not shared. That is, at the instance level, if the type of a particular edge is containment, then we need to ensure that those nodes that are contained are not shared among containers.
- Multiplicity constraints must be satisfied by the edges. The edges that are instances of a relation type with multiplicity constraints must ensure that those constraints are satisfied in the instance.
- The relation formed by the instance edges of containment types must form a forest.

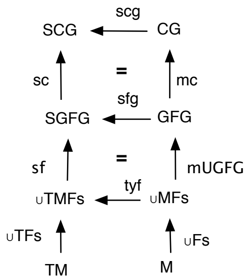
Definition 31 introduces set *SGTyConf* of all conformable typed SGs; in Isabelle, we proved that the examples of Fig. 6.1 belongs to this set.

6.2 Typed Fragments

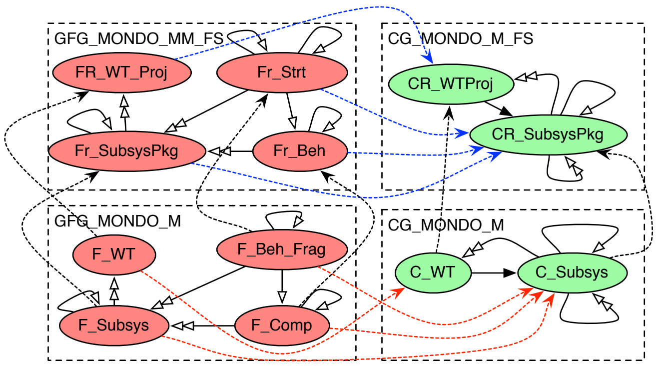
The core of FRAGMENTA's typing approach is described at the level of fragments. This covers both the local and global realms; like in section 4.1, global properties (including conformance) are then considered in the realm of a global fragment that is built as the union of all of model's fragments. The work done here builds up on the notion of typed SG developed in the previous section, which is extended to consider proxy nodes and their references.

We introduce two structures to represent typing at the level of fragments:

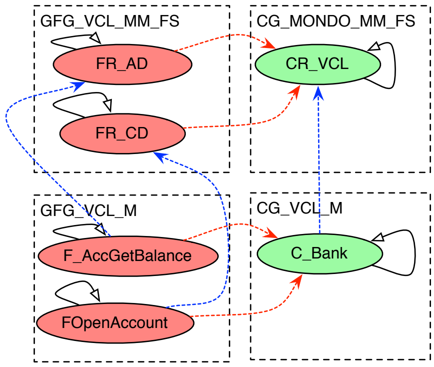
- A type fragment is a pair $TF = (F, iet)$, comprising a fragment $F : Fr$ and a colouring funcion $iet : EsA_F \to SGET$, mapping edges to the type of instance edge being prescribed (def. 32, which defines set TFr, $TF \in TFr$).
- A typed fragment, depicted in Fig. 6.3(a), is a triple $FT = (F, TF, type)$, consisting of an instance-level fragment $F : Fr$, a type fragment $TF : TFr$ and a fragment morphism $type : Fr \to TFr$, mapping the instance fragment to the type one (see def. 33, which defines $FrTy$, such that $FT \in FrTy$).



(a) Morphisms of a typed model



(b) CG and GFG morphisms for FS of MONDO example



(c) CG and GFG morphisms for FS of VCL example

Figure 6.4: Morphisms of typed Fragmenta models

Figure 6.3(b) presents a *FrTy* specimen, describing a simple class model made up of classes and attributes; both type and instance fragments include proxies.

Section 4.1 introduced a relaxed notion of fragment morphism. It covers a variety of model relations at same and different meta-levels (like typing); but it doesn't check certain specificities, such as multiplicities. To complement fragment morphisms, we introduce the notion of conformance between type and instance fragments, to check that the instance conforms to the constraints imposed by the type. This mimics the conformance constraints defined above for typed SGs. The conformance constraints are: (a) edge types of instance fragment conform to those prescribed by type fragment (commutativity of diagram in Fig. 6.3(a)); (b) abstract nodes may not have direct instances; (c) containments are not shared; (d) multiplicity constraints; and (e) the relation formed by instances of containment edges forms a forest. The specification of these constraints takes proxy nodes into account (as illustrated in Fig. 6.3(b)) – see def. 34.

6.3 Typed Models with Fragmentation Strategies

Model typing builds up on the notion of fragment typing and FSs enrich model typing. The following structures provide models with typing and FSs:

- A FS (def. 36) is a tuple $FS = (GFG_S, CG_S, sc, sf)$, comprising the FS's CG (cluster regions), a FS's GFG (fragment regions), and morphisms sc (GFG_S to CG_S) and sf (model fragment elements to GFG_S) – illustrated in Fig. 2.1(a).
- A type model (a fragmented metamodel) differs from a model (section 4.4) in that it uses type rather than plain fragments. A type model with FS, depicted in Fig. 6.3(c), is a tuple $TFSM = (TM, FS)$, containing a type model $TM = (GFG, CG, mc, fd)$ and a FS.
- A typed model (def. 38)puts together type and instance models. It is a tuple $MT = (M, TM, scg, sgfg, ty)$, made of a model M, a type model TM and three morphisms: (i) scg maps CG of M into the FS's CG of TM, (ii) $sgfg$ maps GFG of M into the FS's GFG of TM, and (iii) ty maps model elements of M into its TM counter-part. Typed models and their morphisms are depicted in Fig. 6.4(a).

A typed model requires the commutativity of the diagrams in Fig. 6.4(a), which entail FS conformance (scg for clusters, and $sgfg$ for fragments) and typing (ty, through union of fragments of M and TM).

Fig. 6.4(b) depicts the morphisms that exist between a model's CG and GFG and their counterparts in the metamodel's FS for the example of Fig. 2.1. Likewise for Fig. 6.4(c), which described the VCL example of Figs. 2.2 and 2.3. The top graphs describe the cluster and fragment regions of the FSs described in each example (e.g see Fig.2.1(a)).

Chapter 7

Discussion

We now discuss the results presented in this report.

Modular design. FRAGMENTA aims to support separation of concerns effectively. This, however, brings a complexity cost to the underlying theory. SGs, with their support for inheritance, add complexity to plain graphs; fragments, with their proxies, add further complexity to SGs. FRAGMENTA hides this complexity to enable design of fragmented models that harness separation of concerns. The support for both top-down and bottom-up design means that designers can choose the scheme that best suits their problems and way of thinking. This is realised through FRAGMENTA's concepts of continuations and imports that are variations on how proxies and their references are interpreted at upper level of GFGs.

To gain the important result of global preservation of inheritance acyclicity checked locally (fact 6), we forbid proxies with supertypes. We do not see this as a serious restriction. It can be seen as a design rule whereby a concept's supertypes must be defined when the concept is first introduced; proxies may then have subtypes, but no supertypes. In the end, what we gain is greater than what we loose, given the applicability of the result at both meta and instance levels, and the pervasive use of inheritance in MDE- and DSL-based modelling.

A theory of separation. Chapter 5 presented colimit-based model composition, which resolves references through substitution. FRAGMENTA, however, keeps the models fragmented. The compositions that are required for global purposes are based on the union of all model fragments without reference resolution, a simpler operation. FRAGMENTA lives well with separation; its machinery handles a world where a concept may be represented by many nodes, in contrast with monolithic approaches that support one node per concept only. We envision the resolution compositions outlined in chapter 5 as being an aid to designers to get a clean big picture.

The definition of fragments connects proxies to their referring nodes (function tr, def. 14), which does not preclude or impede use of fragments in isolation. This function may be implemented externally to the fragment definition.

Fragmentation strategies complement metalevel definitions of types with definitions of fragmentation structure. This ensures uniform fragmentations across model instances, which is useful when dealing with big models and collections of related models. This paper's running example (Fig. 2.1) illustrates usefulness of FSs concept; the different wind-turbine controllers should have a uniform structure. Often, such uniformities are agreed among developers with no means to express or enforce them, which complicates the processing of models, introducing accidental complexity. Our approach formally defines FSs so that their conformity can be enforced and checked by tools. In our theory, such conformances are described as a commuting of instance and type diagrams, as shown in Fig. 6.4(a).

Fragmenta's realisations. Fragmenta and its founding ideas have been implemented in two Eclipse-based tools as part of EU project MONDO: *DSL-tao* [PGG+] enables the pattern-based construction of DSL meta-models and their supporting modelling environments, supporting Fragmenta's concepts of fragment and cluster; *EMF-Splitter* [GGKdL14] implements the notion of FS proposed here[1]. Fragmenta can also be used as a modularity paradigm with the notions of cluster and fragments realised in its many guises. The modularity mechanisms of the Visual Contract Language (VCL) [AKMG10, AK10, AGK11, AG15] resemble Fragmenta. In VCL, Fragmenta's clusters are packages and fragments are VCL diagrams. VCL does not provide any support for top-down design. Fragmenta's contructions could greatly simplify the design of a modelling language such as VCL.

Verification	268
Validation	123
Total	**391**

Table 7.1: Number of Isabelle proofs undertaken to validate and verify Fragmenta.

Machine-assisted specification and proof. Fragmenta was specified in the Z language and its consistency was checked using the CZT typechecker to ensure consistency with respect to names and types. Z's expressivity, grounded on its mathematical generality, high-order capabilities and its Zermelo-Fraenkel set-theory underpinning (a widely accepted foundation of mathematics), enabled us to describe Fragmenta with its mathematical definitions based on graphs, functions, sets, relations and categories. This level of expressivity was known to us based on our prior experience with Z. The Z specification (very close to the presentation given here and provided in appendix B) was then encoded in the state of the art Isabelle proof assistant[2] with its underlying expressive high-order logic. This step required some meaning-preserving changes to cater to Isabelle's specificities (e.g., Isabelle's lack of partial function primitive). Isabelle was used to validate and verify Fragmenta; we proved general theorems concerning desired properties (verification) and theorems concerning examples (validation). Table 7.1 gives the number of Isabelle proofs that were undertaken.

The real world. Our case studies include the industrial language used here and several examples drawn from VCL [AKMG10, AG15], a medium sized modelling language. Fragmenta's SGs are an abstraction of MDE structural models, supporting inheritance, composition and multiplicities. Fragmenta's proxies are an abstraction of EMF proxies [SBPM08] and VCL's referencing mechanism. Our proved result (fact 6) showing that the well-formedness of a inheritance hierarchy (acyclicity) checked locally at the fragment level is preserved globally (provided some local constraints are met, namely that proxies may not have supertypes) is relevant for the current practice due to the popularity of EMF; this means that any code that is generated from a Fragmenta-like structure of models and metamodels and that complies with its constraints is guaranteed to be free of compilation errors concerning inheritance well-formedness.

Fragmenta's three-level architecture can capture the tree-based structure of modern modelling and programming projects; in terms of a file system, fragments can be mapped to files and clusters to folders.

[1] DSL-Tao: `http://bit.ly/1CPTYZd`. EMF-Splitter: `http://bit.ly/1Eq1TZD`

[2] The Isabelle theories can be found at `http://www.miso.es/fragmenta/`

Formalisation. FRAGMENTA formalises inheritance using coloured edges in SGs, as any other edge, unlike similar graphs [HEE09, JT12], which capture inheritance as a relation. The edge solution gives uniformity to our theory and makes inheritance amenable to typing (as illustrated in Fig. 6.3(b)); our edge-colouring solution also simplifies checking the prescribed edge type to a simple diagram commuting (Fig. 6.3(a)).

A formalisation of references as coloured edges was chosen in detriment of a partial function ($refs : V \nrightarrow V$). This choice benefits FRAGMENTA's uniformity, coherence (all edges are formalised as such) and clarity (such edges appear in the morphisms from local fragment nodes to GFGs as inter-fragment GFG edges). The drawback of reference edges is that they lie unreferenced in SGs, requiring use of the reference target function of fragments to get graphs that are referenced.

Chapter 8

Related Work

There is a widespread acknowledgement of MDE's scalability challenge and the need for modularity. The popular EMF provides the means to partition models with proxies, but lacks support for fragmentation strategies (FSs). To improve this, [SZFK12] proposes a non-formal persistence framework for EMF to fragment models along annotated metamodel compositions. Our theory is formal and provides a powerful notion of fragmentation regions that allows metamodel-defined fragmentations along our container primitive of clusters.

Heidenreich et al [HHJZ09] propose a non-formal language independent modularisation approach that puts together fragments through composition interfaces made of reference and variation points. Fragmenta is more abstract than [HHJZ09]; it provides a mathematical notion of joints based on proxys and their references, similar to the reference points of [HHJZ09], that is amenable to model composition based on the general colimit.

Weisemöller and Schürn [WS08] try to improve the modularisation of MOF, a popular metamodelling language. Their formalisation introduces metamodel components equipped with export and import interfaces to enable composition. Their definition of metamodel equates to the simple graphs presented here, not considering important concepts such as inheritance, composition and multiplicities. Furthermore, [WS08] deals with metamodels only; Fragmenta covers both levels, not making a substantial distinction between models and metamodels.

Certain formal approaches to *merge composition* [NSC$^+$07, SE05] also use the colimit construction of category theory. Our work does a more thorough treatment of the proxy mechanism for referencing and incremental definition, which is slightly different from the merge, and puts forward the simpler union composition, where references are not resolved.

Hermann et al [HEE09] investigate inheritance in a graph transformation setting, considering a special condition in meta-model morphisms to ensure existence of co-limits of arbitrary categorical diagrams. Fragmenta does not perform co-limits over arbitrary diagrams, considering only those that are related through proxies (interface graphs, see Fig. 5.1). Although related, settings of [HEE09] and Fragmenta are different; [HEE09] is not concerned at all by inheritance acyclicity and proxies.

Component graphs [JT12] with its two-layer structuring, local and network, resemble Fragmenta's local and global fragment levels. Fragmenta provides an extra third level of clusters. [JT12] provides IC-graphs, which are similar to SGs but without multiplicities, and uses import and export interfaces to enable composition. Fragmenta uses proxies to build fragments incrementally in either a bottom-up or top-down fashion, which is closer to EMF proxies. [JT12] acknowledges how such graph structures are capable of capturing the EMF, but without providing a formal study of proxies (an EMF concept). [JT12] also acknowledges that inheritance well-formedness issues (cycles) may arise when parts are composed, but there is no proved result, like the one presented here, concerning the global preservation of inheritance well-formedness (acyclicity, fact 6) provided some local constraints are met.

Hamiaz et al [HPCT14] formalise in the Coq theorem prover the model composition operations of [HHJZ09]. This shares Fragmenta's emphasis on formalisations developed with proof assistants. Fragmenta, however, is more abstract; it is a general approach that mimics common features of MDE; composition is expressed in terms of general mathematical operators, such as colimit and set-union.

Several approaches split monolithic models. Kelsen et al [KMG11] propose an algorithm to split a model into submodels, where each submodel is conformant to the original metamodel with association multiplicities taken into account. Strüber et al [STJS13] provide a splitting mechanism for both metamodels and models based on the component graphs of [JT12]. In [SRTC14], Strüber et al use [JT12] as the basis of an approach to split a model based on the relevance of its elements using information retrieval methods. Unlike these works, Fragmenta is a design theory, supporting the novel idea of metamodel defined FSs and a hierarchical organisation of fragments into clusters.

Chapter 9

Conclusions

This paper presented FRAGMENTA, a formal theory to fragment MDE models. This paper's main result (fact 6), formally derived from the theory, is that the satisfaction of some local fragments constraints (particularly, the fact that proxies may not have supertypes) is enough to ensure that inheritance hierarchies remain well-formed (acyclic) globally when fragments are composed. This is relevant because the widely diffused EMF uses a similar proxy mechanism. FRAGMENTA's main novelties include: (a) the formal treatment of model fragments exploiting the particularities of a seaming mechanism based on proxies, (b) metalevel fragmentation strategies that stipulate a fragmentation structure to model instances, (c) support for both bottom-up and top-down fragmented designs and (d) three-level model architecture. Other minor novelties include: (i) the observation that although fragmented models are amenable to colimit-based composition, this operation is not necessary for the theory's internal global processing, which can live with unresolved references; and (ii) fragment graphs and the way they capture the proxy concept.

FRAGMENTA was developed with the assistance of tools, using specification type-checkers and proof assistants. Our team developed an initial tool prototype[1]. We are currently working on FRAGMENTA's merging mechanisms, further developing its tool and applying the theory to additional case studies.

[1] Available at `http://bit.ly/1Eq1TZD`

References

[AG14] Nuno Amálio and Christian Glodt. A tool for visual and formal modelling of software designs. *Science of Computer Programming*, 2014. In press.

[AG15] Nuno Amálio and Christian Glodt. A tool for visual and formal modelling of software designs. *Science of Computer Programming*, 98, Part 1:52 – 79, 2015.

[AGK11] Nuno Amálio, Christian Glodt, and Pierre Kelsen. Building VCL models and automatically generating Z specifications from them. In *FM 2011*, volume 6664 of *LNCS*, pages 149–153. Springer, 2011.

[AK10] Nuno Amálio and Pierre Kelsen. Modular design by contract visually and formally using VCL. In *VL/HCC*, pages 227–234. IEEE, 2010.

[AKMG10] Nuno Amálio, Pierre Kelsen, Qin Ma, and Christian Glodt. Using VCL as an aspect-oriented approach to requirements modelling. *TAOSD*, 7:151–199, 2010.

[BW98] Michael Barr and Charles Wells. *Category Theory for Computing Science*. 1998.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[GGKdL14] Antonio Garmendia, Esther Guerra, Dimitrios S. Kolovos, and Juan de Lara. EMF splitter: A structured approach to EMF modularity. In *Proc. XM@MODELS*, volume 1239 of *CEUR Workshop Proceedings*, pages 22–31. CEUR-WS.org, 2014.

[HEE09] Frank Hermann, Harmut Ehrig, and Claudia Ermel. Transformation of type graphs with inheritance for ensuring security in e-government networks. In *FASE 2009*, 2009.

[HHJZ09] Florian Heidenreich, Jakob Henriksson, Jendrik Johannes, and Steffen Zschaler. On language-independent model modularisation. *TAOSD VI*, 6:39–82, 2009.

[HPCT14] Mounira Kezadri Hamiaz, Marc Pantel, Benoît Combemale, and Xavier Thirioux. Correct-by-construction model composition: Application to the invasive software composition method. In *Proc. FESCA@ETAPS*, volume 147 of *EPTCS*, pages 108–122, 2014.

[JT12] Stefan Jurack and Gabriele Taentzer. Transformation of typed composite graphs with inheritance and containment structures. *Fundam. Inform.*, 118(1-2):97–134, 2012.

[KMG11] Pierre Kelsen, Qin Ma, and Christian Glodt. Models within models: Taming model complexity using the sub-model lattice. In *FASE'11*, volume 6603 of *LNCS*, pages 171–185. Springer, 2011.

[KRM+13] Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, and Jordi Cabot. A research roadmap towards achieving scalability in model driven engineering. In *BigMDE*, pages 1–10, 2013.

[Lan71] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, 1971.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[NSC+07] Shiva Nejati, Mehrdad Sabetzadeh, Marsha Chechik, Steve Easterbrook, and Pamela Zave. Matching and merging of statecharts specifications. In *ICSE'2007*, 2007.

[Par72] David Lodge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[PGG+] Ana Pescador, Antonio Garmendia, Esther Guerra, Jesus Sánchez Cuadrado, and Juan de Lara. Pattern-based development of domain-specific modelling languages. In *MODELS 2015.* (this volume).

[Pie91] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists.* MIT Press, 1991.

[SBPM08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework.* Addison-Wesley, 2008.

[SE05] Mehrdad Sabetzadeh and Steve Easterbrook. An algebraic framework for merging incomplete and inconsistent views. In *RE 2005*. IEEE, 2005.

[SRTC14] Daniel Strüber, Julia Rubin, Gabriele Taentzer, and Marsha Chechik. Splitting models using information retrieval and model crawling techniques. In *FASE*, volume 8411 of *LNCS*, 2014.

[STJS13] Daniel Strüber, Gabriele Taentzer, Stefan Jurack, and Tim Schäfer. Towards a distributed modeling process based on composite models. In *FASE*, volume 7793 of *LNCS*. Springer, 2013.

[SZFK12] Markus Scheidgen, Anatolij Zubow, Joachim Fischer, and Thomas H. Kolbe. Automated and transparent model fragmentation for persisting large models. In *MODELS 2012*, volume 7590 of *LNCS*, pages 102–118. Springer, 2012.

[TOHSMS99] Peri Tarr, Harold Ossher, William Harrison, and Jr Stanley M. Sutton. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *ICSE'99*, 1999.

[WS08] Ingo Weisemöller and Andy Schürr. Formal definition of MOF 2.0 metamodel components and composition. In *MoDELS*, volume 5301 of *LNCS*, pages 386–400. Springer, 2008.

Appendix A

Auxiliary Definitions

This appendix presents the mathematical definitions that underpin FRAGMENTA. All these definitions have been specified using the Z specification language (appendix B). All theorems that are associated with the mathematical definitions were proved using the Isabelle proof assistant.[1]

A.1 Base Mathematical Definitions

Definition 1 (Relations)**.** Sets of *acyclic*, *connected*, *tree* and *forest*, and *injrel* relation are as follows:

$$
\begin{array}{l}
\mathit{acyclic}[X] = \{r : X \leftrightarrow X \mid r^{+} \cap \mathrm{id}[X] = \varnothing\} \\
\mathit{connected}[X] = \{r : X \leftrightarrow X \mid (\forall x : \mathrm{dom}\, r;\ y : \mathrm{ran}\, r \bullet x \mapsto y \in r^{+})\} \\
\mathit{tree}[X] = \{r : X \leftrightarrow X \mid r \in \mathit{acyclic} \land r \in X \rightarrow\!\!\!\!\!\!+\ X\} \\
\mathit{forest}[X] = \{r : X \leftrightarrow X \mid r \in \mathit{acyclic} \land (\forall s : X \leftrightarrow X \mid s \subseteq r \land s \in \mathit{connected} \bullet s \in \mathit{tree})\} \\
X\ \mathit{injrel}\ Y = \{r : X \leftrightarrow Y \mid (\forall x : X;\ y_2, y_3 : Y \bullet (x, y_1) \in r \land (x, y_2) \in r \Rightarrow y_1 = y_2\}
\end{array}
$$

Above, $^{+}$ stands for the transitive closure, and id stands for the identity relation. The definition of *forest* says that all connected sub-relations must be trees. □

Fact 1 (Transitive closure theorems)**.** Given relations r and s, we have the following laws:

$$
\begin{array}{l}
\vdash (r \cup s)^{+} \subseteq r^{+} \cup s^{+} \\
(\mathrm{dom}\, r \cup \mathrm{ran}\, r) \cap (\mathrm{dom}\, s \cup \mathrm{ran}\, s) = \varnothing \vdash (r \cup s)^{+} = r^{+} \cup s^{+} \\
(\mathrm{dom}\, r \cup \mathrm{ran}\, r) \cap (\mathrm{dom}\, s \cup \mathrm{ran}\, s) = \varnothing \vdash (r \cup s)^{*} = r^{*} \cup s^{*} \\
\mathrm{dom}\, r \cap \mathrm{dom}\, s = \varnothing;\ \mathrm{ran}\, s \cap \mathrm{dom}\, r = \varnothing \vdash (r \cup s)^{+} = r^{+} \cup s^{+} \cup r^{+} \mathbin{⨾} (\mathrm{ran}\ r \lhd s^{+}) \\
\mathrm{dom}\, r \cap \mathrm{dom}\, s = \varnothing;\ \mathrm{ran}\, s \cap \mathrm{dom}\, r = \varnothing \vdash (r \cup s)^{*} = r^{*} \cup s^{*} \cup r^{+} \mathbin{⨾} (\mathrm{ran}\ r \lhd s^{+}) \\
(\mathrm{dom}\, r \cup \mathrm{ran}\, r) \cap (\mathrm{dom}\, s \cup \mathrm{ran}\, s) = \varnothing \vdash r \in \mathit{acyclic} \land s \in \mathit{acyclic} \\
\mathrm{dom}\, r \cap \mathrm{dom}\, s = \varnothing;\ \mathrm{ran}\, s \cap \mathrm{dom}\, r = \varnothing \vdash r \in \mathit{acyclic} \land s \in \mathit{acyclic} \\
s \in \mathit{acyclic};\ r \subseteq s^{+} \vdash r \in \mathit{acyclic}
\end{array}
$$

In the definitions above, ⨾ is relation composition.

Proof. All laws given above have been proved in the Isabelle proof assistant. □

[1] The Isabelle enconding of FRAGMENTA, together with its theorems and proofs can be found in `http://www.miso.es/fragmenta/`

A.2 Graphs

Definition 2 (Vertices and Edges)**.** The disjoint sets V and E represent all possible nodes and all possible edges of graphs, respectively. $\square$

Definition 3 (Graphs)**.** A graph $G = (V_G, E_G, s, t)$ consists of sets $V_G \subseteq V$ of nodes and $E_G \subseteq E$ of edges, and source and target functions $s, t : E_G \to V_G$.

The set of graphs Gr, such that $G : Gr$, is defined as:

$$Gr = \{(V_G, E_G, s, t) \mid V_G \in \mathbb{P}\, V \land E_G \in \mathbb{P}\, E \land s \in E_G \to V_G \land t \in E_G \to V_G\}$$

Auxiliary Definitions. The next functions extract the components of a graph:

$$\begin{array}{llll} Ns : Gr \to \mathbb{P}\, V & Es : Gr \to \mathbb{P}\, E & src : Gr \to (E \to V) & tgt : Gr \to (E \to V) \\ Ns(V_G, E_G, s, t) = V_G & Es(V_G, E_G, s, t) = E_G & src(V_G, E_G, s, t) = s & tgt(V_G, E_G, s, t) = t \end{array}$$

In the following, given a graph G, we write Ns_G, Es_G, src_G and tgt_G to yields the components of a graph (nodes, edges, source and target functions), which abbreviates function application (e.g. we write Ns_G to mean $Ns\ G$).

We introduce several functions and predicates for graphs: (a) *EsId* gives all self edges, (b) *adjacent* indicates whether any two nodes are adjacent, (c) *rel* yields relation induced by a graph, (d) *restrict* extracts a sub-graph from the given graph that considers the given set of edges only, (e) *acyclicG* says whether a graph is acyclic or not, (f) *disj* says whether two graphs are disjoint (includes both nodes and edges), (g) *disjEs* says whether the edges of two graphs are disjoint, (h) *replaceGfun*: does a replacement of nodes on a source or target function, (i) *replaceG* replaces the nodes of a graph given a substitution.

$$\begin{array}{ll} EsId : Gr \to \mathbb{P}\, E & adjacent : \mathbb{P}(V \times V \times Gr) \\ EsId\ G = \{e : Es_G \mid src_G\ e = tgt_G\ e\} & adjacent(v_1, v_2, G) \Leftrightarrow \exists\, e : Es_G \bullet src_G\ e = v_1 \land tgt_G\ e = v_2 \\ rel : Gr \to (V \leftrightarrow V) & restrict : (Gr \times \mathbb{P}\, E) \to Gr \\ rel\ G = \{(v_1, v_2) \mid adjacent(v_1, v_2, G)\} & restrict(G, E_r) = (Ns_G, Es_G \cap E_r, E_r \lhd src_G, E_r \lhd tgt_G) \\ acyclicG\_ : \mathbb{P}(Gr) & disjEs\_ : \mathbb{P}(Gr \times Gr) \\ acyclicG\ G \Leftrightarrow rel\ G \in acyclic & disjEs(G_1, G_2) \Leftrightarrow Es_{G_1} \cap Es_{G_2} = \varnothing \end{array}$$

$$\begin{array}{l} disj\_ : \mathbb{P}(Gr \times Gr) \\ disj(G_1, G_2) \Leftrightarrow Ns_{G_1} \cap Ns_{G_2} = \varnothing \land disjEs(G_1, G_2) \\ replaceGfun : (E \nrightarrow V) \to (V \nrightarrow V) \to (E \nrightarrow V) \\ replaceGfun\ f\ sub = f \oplus \{(e, v) \mid e \in \operatorname{dom} f \land (f\ e) \in \operatorname{dom} sub \land v \in V \land sub\,(f\ e) = v\} \\ replaceG : Gr \to (V \nrightarrow V) \to Gr \\ replaceG\ G\ sub = (Ns_G \setminus \operatorname{dom} sub \cup \operatorname{ran}(Ns_G \lhd sub), Es_G, replaceGfun\ src_G\ sub, replaceGfun\ tgt_G\ sub) \end{array}$$

Above, symbol $\lhd$ stands for domain restriction; *acyclic* is set of acyclic relations (definition 1).

Properties. The following proof laws support proof involving graphs:

$$\begin{array}{l} G_1 \in Gr;\ G_2 \in Gr \vdash disjEs(G_1, G_2) = disjEs(G_2, G_1) \quad G_1 \in Gr;\ G_2 \in Gr \vdash disj(G_1, G_2) = disj(G_2, G_1) \\ G_1 \in Gr;\ G_2 \in Gr;\ disj(G_1, G_2) \vdash disj(restrict\ G_1, restrict\ G_2) \\ G \in Gr;\ \operatorname{dom} sub \cap \operatorname{ran} sub = \varnothing \vdash replaceG\ G\ sub \in Gr \\ G \in Gr;\ \operatorname{dom} sub \cap Ns_G = \varnothing \vdash replaceG\ G\ sub = G \\ disjEs\,(G_1, G_2) \vdash disjEs\,(replaceG\ G_1\ sub, replaceG\ G2\ sub) \end{array}$$

Proof. The laws given above have been proved in the Isabelle proof assistant. $\square$

Definition 4 (Union of graphs)**.** The union of graphs $G_1, G_2 : Gr$ is defined as:

$$G_1 \cup_G G_2 = (Ns_{G_1} \cup Ns_{G_2}, Es_{G_1} \cup Es_{G_2}, src_{G_1} \cup src_{G_2}, tgt_{G_1} \cup tgt_{G_2})$$

The union of two graphs is defined as the union of the graph's components.

Properties. There are the following proof laws for graph union:

$$G_1 \in Gr;\ G_2 \in Gr \vdash (G_1 \cup_G G_2) \in Gr$$
$$G_1 \in Gr;\ G_2 \in Gr;\ disjEs(G_1, G_2) \vdash G_1 \cup_G G_2 = G_2 \cup_G G_1$$
$$G_1 \in Gr;\ G_2 \in Gr;\ disjEs(G_1, G_2) \vdash restrict(G_1 \cup_G G_2, es) = restrict(G_1, es) \cup_G restrict(G_2, es)$$

Proof. The laws given above have been proved in Isabelle.
□

Fact 2 (Graph Acyclicity)**.** The union of graphs $G_1, G_2 : Gr$ is acyclic provided: (i) the individual graphs are also acyclic, and (ii) they are mutually disjoint:

$$G_1 \in Gr;\ G_2 \in Gr \vdash acyclicG(G_1 \cup_G G_2) \Leftrightarrow acyclicG\ G_1 \wedge acyclicG\ G_2 \wedge disj(G_1, G_2)$$

Properties. The following laws support the fact's theorems outlined above:

$$G_1 \in Gr;\ G_2 \in Gr;\ disjEs(G_1, G_2);\ adjacent(x, y, G_1 \cup_G G_2) \vdash adjacent(x, y, G_1) \vee adjacent(x, y, G_2)$$
$$G_1 \in Gr;\ G_2 \in Gr;\ disjEs(G_1, G_2);\ adjacent(x, y, G_1) \vdash adjacent(x, y, G_1 \cup_G G_2)$$
$$G_1 \in Gr;\ G_2 \in Gr;\ disjEs(G_1, G_2);\ adjacent(x, y, G_2) \vdash adjacent(x, y, G_1 \cup_G G_2)$$
$$G_1 \in Gr;\ G_2 \in Gr;\ disjEs(G_1, G_2) \vdash rel(G_1 \cup_G G_2) = rel\ G_1 \cup rel\ G_2$$
$$G_1 \in Gr;\ G_2 \in Gr;\ disj(G_1, G_2) \vdash (\mathrm{dom}(rel\ G_1) \cup \mathrm{ran}(rel\ G_1)) \cap ((\mathrm{dom}(rel\ G_2) \cup \mathrm{ran}(rel\ G_2) = \varnothing$$

Proof. All theorems outlined above were proved in the Isabelle proof assistant. □

Definition 5 (G-Morphisms)**.** A graph morphism $m : G_1 \rightarrow G_2$ defines a mapping between graphs $G_1, G_2 : Gr$; it comprises a pair of functions $m = (f_V, f_E)$, $f_V : Ns_{G1} \rightarrow Ns_{G2}$ and $f_E : Es_{G1} \rightarrow Es_{G2}$, mapping nodes and edges respectively that preserve the source and target functions of edges: $f_V \circ src_{G_1} = src_{G_2} \circ f_E$ and $f_V \circ tgt_{G_1} = tgt_{G_2} \circ f_E$ (Fig. 3.1(b)).

Sets *GrMorph* (all possible graph morphisms) and $G_1 \rightarrow G_2$ (morphisms between two graphs), such that $G_1 \rightarrow G_2 \subseteq GrMorph$, are defined as:

$$GrMorph = \{(fv, fe) \mid fv \in V \nrightarrow V \wedge fe \in E \nrightarrow E\}$$
$$G_1 \rightarrow G_2 = \{(fv, fe) \mid fv \in Ns_{G_1} \rightarrow Ns_{G_2} \wedge fe \in Es_{G_1} \rightarrow Es_{G_2} \wedge fv \circ src_{G_1} = src_{G_2} \circ fe \wedge fv \circ tgt_{G_1} = tgt_{G_2} \circ fe\}$$

Above, the two equations involving function composition (symbol $\circ$) ensure diagram commutativity (depicted in Fig. 3.1(b)).

Auxiliary Definitions. Functions f_V and f_E extract the two components of a graph morphism:

$$f_V : GrMorph \rightarrow V \nrightarrow V \qquad f_E : GrMorph \rightarrow E \nrightarrow E$$
$$f_V(fv, fe) = fv \qquad f_E(fv, fe) = fe$$

□

Definition 6 (Composition of Graph Morphisms)**.** The composition of graph morphisms $f : G_1 \rightarrow G_2$ and $g : G_2 \rightarrow G_3$, $G_{i \in \{1,3\}} : Gr$, is defined as:

$$g \circ_G f = ((f_V\ g) \circ (f_V\ f), (f_E\ g) \circ (f_E\ f))$$

□

A.3 Categories

Definition 7 (Category Objects and Morphisms)**.** The disjoint sets O and M represent all possible objects of categories and all possible morphisms between such objects, respectively. $\square$

Definition 8 (Category)**.** A category is defined by the tuple $\mathcal{C} = (O_C, M_C, dm, cd, id_C, \circ)$, comprising a set $O_C \subseteq O$ of objects, a set $M_C \subseteq M$ of morphisms, two functions $dm, cd : M_C \rightarrow O_C$ that give the domain and co-domain of a morphism, an identity operator $id_C : O_C \rightarrow M_C$ that gives the identity arrow associated with an object, and a morphism composition operator $\circ : M_C \times M_C \rightarrow M_C$.

The base set of all categories is defined as:

$$\begin{aligned}
&Cat_0 = \{(O_C, M_C, dm, cd, idn, \circ) \mid O_C \in \mathbb{P}\, O \wedge M_C \in \mathbb{P}\, M \wedge dm \in O_C \rightarrow M_C \\
&\quad \wedge cd \in O_C \rightarrow M_C \wedge idn \in O_C \rightarrow M_C \wedge \circ \in M_C \times M_C \rightarrow M_C\}
\end{aligned}$$

The functions that follow extract the individual components of a category:

$$\begin{array}{ll}
obs : Cat_0 \rightarrow \mathbb{P}\, O & morphs : Cat_0 \rightarrow \mathbb{P}\, M \\
obs(O_C, M_C, dm, cd, idn, \circ_C) = O_C & morphs(O_C, M_C, dm, cd, idn, \circ_C) = M_C \\
dom : Cat_0 \rightarrow M \nrightarrow O & cod : Cat_0 \rightarrow M \nrightarrow O \\
dom(O_C, M_C, dm, cd, idn, \circ_C) = dm & cod(O_C, M_C, dm, cd, idn, \circ_C) = cd \\
id : Cat_0 \rightarrow O \rightarrow M & \circ : Cat_0 \rightarrow M_C \times M_C \nrightarrow M_C \\
id(O_C, M_C, dm, cd, idn, \circ_C) = idn & \circ(O_C, M_C, dm, cd, idn, \circ) = \circ
\end{array}$$

In the following, given a category $\mathcal{C}$, we write obs_C, $morphs_C$, dom_C, cod_C and id_C to mean $obs\,\mathcal{C}$, $morphs\,\mathcal{C}$, $dom\,\mathcal{C}$, $cod\,\mathcal{C}$ and $id\,\mathcal{C}$, respectively. We write $g \circ_C f$ to mean $\circ\mathcal{C}\,(g, f)$.

We define the set of morphisms between two objects of some category C as:

$$A \rightarrow_C B = \{m : morphs_C \mid A \in obs_C \wedge B \in obs_C \wedge dom_C\, m = A \wedge cod_C\, m = B\}$$

From the definitions above, we define the set of valid categories as:

$$\begin{aligned}
&Cat = \{\mathcal{C} : Cat_0 \mid (\forall A : obs_C \bullet id_C\, A \in A \rightarrow_C A) \\
&\quad \wedge (\forall f, g : morphs_C \mid dom_C\, g = cod_C\, f \bullet g \circ_C f \in dom_C\, f \rightarrow_C cod_C\, g) \\
&\quad \wedge (\forall A, B, C, D : obs_C \bullet \forall f : A \rightarrow_C B;\ g : B \rightarrow_C C;\ h : C \rightarrow_C D \bullet \\
&\qquad h \circ_C (g \circ_C f) = (h \circ_C g) \circ_C f) \\
&\quad \wedge (\forall A, B : obs_C \bullet \forall f : A \rightarrow_C B \bullet id_C\, B \circ_C f = f \wedge f \circ id_C\, A = f)\}
\end{aligned}$$

$\square$

Fact 3 (Category of **Graph**)**.** We can form the category **Graph** by taking graphs as category objects (def. 3) and graph morphisms (def. 5) as category morphisms. We define the domain, co-domain, identity and composition of **Graph** as:

$$\begin{array}{ll}
domCG : GrMorph \rightarrow Gr & codCG : GrMorph \rightarrow Gr \\
domCG\, m = G_1 \Leftrightarrow m \in G_1 \rightarrow G_2 & codCG\, m = G_2 \Leftrightarrow m \in G_1 \rightarrow G_2 \\
idCG : Gr \rightarrow GrMorph & \circ_{CG} : GrMorph \times GrMorph \rightarrow GrMorph \\
idCG\, G1 = m \Leftrightarrow m \in G1 \rightarrow G2 & gm_1 \circ_{CG} m_2 = m3 \Leftrightarrow m3 = m_1 \circ_G m_2
\end{array}$$

The category **Graph** is defined as:

$$\mathbf{Graph} = (Gr, GrMorph, domCG, codCG, idCG, \circ_{CG})$$

Proof. All required proofs were done in Isabelle. $\square$

A.4 Structural Graphs

Definition 9 (Node and Edge Types)**.** The node types of a SG are: normal, abstract and proxy. The edge types of a SG are: inheritance, containment, relation, link and reference.

$$SGNT = \{nnrml, nabst, nprxy\} \qquad SGET = \{einh, ecomp, erel, elnk, eref\}$$

□

Definition 10 (Multiplicities)**.** Sets *MultUVal* (upper bound values) and *Mult* (multiplicities) are defined below. *MultUVal* is disjoint union (symbol $\uplus$) of natural numbers and singleton set with $*$ (*many*); *Mult* is a set of lower and upper bound pairs.

$$\begin{array}{l} MultUVal = \mathbb{N} \uplus \{*\} \\ Mult = \{(lb, ub) \mid lb \in \mathbb{N} \wedge ub \in MultUVal \wedge (ub = * \vee (ub \in \mathbb{N} \wedge lb \leqslant ub))\} \end{array}$$

Auxiliary Definitions. Predicate *multOk* checks whether a set is bounded by given multiplicity:

$$\begin{array}{l} multOk\_ : \mathbb{P}(\mathbb{P}\ V \times Mult) \\ multOk(vs, (lb, ub)) \Leftrightarrow \# vs \geqslant lb \wedge (ub = * \vee (ub \in \mathbb{N} \wedge \# vs \leqslant ub)) \end{array}$$

Above, # stands for set cardinality. □

Definition 11 (Structural Graphs)**.** A structural graph $SG = (G, nty, ety, sm, tm)$ comprises a graph $G : Gr$, two colouring functions for nodes and edges, $nt : Ns_G \rightarrow SGNT$ and $et : Es_G \rightarrow SGET$, and source and target multiplicity functions, $sm, tm : Es_G \rightarrow\!\!\!\!\!+ Mult$ (Fig. 3.1(c)).

Base set SGr_0 of SGs, such that $SG : SGr_0$, is defined as:

$$\begin{array}{l} SGr_0 = \{(G, nt, et, sm, tm) \mid G \in Gr \wedge nt \in Ns_G \rightarrow SGNT \wedge et \in Es_G \rightarrow SGET \\ \quad \wedge\ sm \in Es_G \rightarrow\!\!\!\!\!+ Mult \wedge tm \in Es_G \rightarrow\!\!\!\!\!+ Mult\} \end{array}$$

The next functions extract the components of a SG:

$$\begin{array}{lll} gr : SGr_0 \rightarrow Gr & nty : SGr_0 \rightarrow (V \rightarrow\!\!\!\!\!+ SGNT) & ety : SGr_0 \rightarrow (E \rightarrow\!\!\!\!\!+ SGET) \\ gr(G, nt, et, sm, tm) = G & nty(G, nty, et, sm, tm) = nt & ety(G, nt, et, sm, tm) = et \\ srcm : SGr_0 \rightarrow Mult & tgtm : SGr_0 \rightarrow Mult & \\ srcm(G, nt, et, sm, tm) = sm & tgtm(G, nt, et, sm, tm) = tm & \end{array}$$

We introduce several functions and predicates to operate upon SGr_0: (a) *EsTy* yields all edges of the given types, (b) *NsP* yields all proxy nodes, (c) *EsA* gives all *association* edges (relation, composition and link), (d) *EsR* gives all reference edges, (e) *EsRP* gives all reference edges that are attached to proxy nodes, (f) $\prec_G$ gives SG's inheritance graph formed as restriction of the SG's graph to inheritance edges and excluding the dummy self edges, and (g) $\prec$ is the inheritance relation obtained from the inheritance graph $\prec_G$.

$$\begin{array}{ll} EsTy : SGr_0 \times \mathbb{P}\,SGET \rightarrow\!\!\!\!\!+ \mathbb{P}\,E & NsP : SGr_0 \rightarrow\!\!\!\!\!+ \mathbb{P}\,V \\ EsTy(SG, ets) = ety_{SG}{}^{\sim} (\!| ets |\!) & NsP\ SG = nty_{SG}{}^{\sim} (\!| \{nprxy\} |\!) \\ EsA : SGr_0 \rightarrow \mathbb{P}\,E & EsR : SGr_0 \rightarrow \mathbb{P}\,E \\ EsA\ SG = EsTy(SG, \{erel, ecomp, elink\}) & EsR\ SG = EsTy(SG, \{eref\}) \\ EsRP : SGr_0 \rightarrow \mathbb{P}\,E & \\ EsRP\ SG = \{e : EsR\ SG \mid src_{SG}\, e \in NsP\ SG\} & \\ \prec_G : SGr_0 \rightarrow Gr \quad \prec : SGr_0 \rightarrow (V \leftrightarrow V) & \\ \multicolumn{2}{l}{\prec_G\ SG = restrict(gr\ SG, EsTy(SG, \{einh\}) \setminus (EsId_{SG})) \quad \prec_{SG} = rel(\prec_G\ SG)} \end{array}$$

Above, $^{\sim}$ is the inverse relation, $\setminus$ is set difference, and $(\!|\ |\!)$ denotes the relation image.

Actual set of SGs, SGr, is defined from the base set as:

$$SGr = \{SG : SGr_0 \mid EsR_{SG} \subseteq EsId_{SG} \wedge srcm_{SG} \in EsTy(SG, \{erel, ecomp\}) \to Mult \\ \wedge tgtm_{SG} \in EsTy(SG, \{erel, ecomp\}) \to Mult \wedge srcm_{SG} \llparenthesis EsTy(SG, \{ecomp\}) \rrparenthesis = \{(0,1), 1\} \\ \wedge acyclicG\,(\prec_G\ SG)\}$$

SGs have the following constraints: (a) reference edges (EsR_{SG}) are self edges ($EsId_{SG}$), (b) relation and containment edges must have multiplicities, (c) source multiplicity of containment edges should be $0 \mathinner{..} 1$ or 1, and (d) the inheritance graph must be acyclic (predicate $acyclicG$).

Auxiliary Definitions. Functions $\preccurlyeq$ yields the reflexive transitive closure of $\prec$.

$$\preccurlyeq : SGr \to (V \leftrightarrow V) \\ \preccurlyeq\ SG = (\prec_{SG})^*$$

Here, * denotes the reflexive transitive closure.

Function $clan$ yields inheritance-family of some SG node using $\prec^*$. Functions src^* and tgt^* yield relations that extend the source and target functions of graph to cater to inheritance. $\cup_{SG}$ returns union of two SGs:

$$clan : V \times SGr \to \mathbb{P}\, V \quad src^* : SGr \to (E \leftrightarrow V) \quad tgt^* : SGr \to (E \leftrightarrow V) \\ clan\,(v,\ SG) = \{v' \in Ns_{SG} \mid v' \preccurlyeq_{SG}\ v\} \\ src^*\ SG = \{(e, v) \mid e \in EsA_{SG} \wedge v \in Ns_{SG} \wedge (\exists\, v_2 : Ns_{SG} \bullet v \in clan(v_2, SG) \wedge src_{SG}\ e = v_2)\} \\ tgt^*\ SG = \{(e, v) \mid e \in EsA_{SG} \wedge v \in Ns_{SG} \wedge (\exists\, v_2 : Ns_{SG} \bullet v \in clan(v_2,\ SG) \wedge tgt_{SG}\ e = v_2)\}$$

Properties. We introduce some healthiness conditions for SGs. In particular, that src^* and tgt^* preserve the information of the original source and target functions restricted to association edges (the latter are subsets of the former), namely:

$$SG \in SGr \vdash EsA_{SG} \lhd src_{SG} \subseteq src^*_{SG} \qquad SG \in SGr \vdash EsA_{SG} \lhd tgt_{SG}\ \subseteq tgt^*_{SG}$$

Proof. All healthiness conditions given above were proved using Isabelle. □

Definition 12 (Union of Structural Graphs)**.** The union of SGs $SG_1, SG_2 : SGr$ is defined as:

$$SG_1 \cup_G SG_2 = (gr\ SG_1 \cup_G gr\ SG_2, nty\ SG_1 \cup nty\ SG_2, ety\ SG_1 \cup ety\ SG_2, srcm\ SG_1 \cup srcm\ SG_2, \\ tgtm\ SG_1 \cup tgtm\ SG_2)$$

The union of two SGs is defined as the union of the SG's components, which involves use of graph union (def. 4).

Properties. The following laws support proof with SG-Union:

$$\vdash src(SG_1 \cup_{SG} SG_2) = src\ SG_1 \cup src\ SG_2 \qquad \vdash tgt(SG_1 \cup_{SG} SG_2) = tgt\ SG_1 \cup tgt\ SG_2 \\ \vdash EsR(SG_1 \cup_{SG} SG_2) = EsR\ SG_1 \cup EsR\ SG_2 \qquad \vdash EsI(SG_1 \cup_{SG} SG_2) = EsI\ SG_1 \cup EsI\ SG_2 \\ \vdash EsId(SG_1 \cup_{SG} SG_2) = EsId\ SG_1 \cup EsId\ SG_2 \qquad \vdash NsP(SG_1 \cup_{SG} SG_2) = NsP\ SG_1 \cup NsP\ SG_2 \\ \vdash EsRP(SG_1 \cup_{SG} SG_2) = EsRP\ SG_1 \cup EsRP\ SG_2 \qquad \vdash \prec\ (SG_1 \cup_{SG} SG_2) = \prec_{SG_1} \cup \prec_{SG_2}$$

Proof. The laws given above have been proved in Isabelle. □

Fact 4 (Union Composition of Structural Graphs)**.** Given SGs $SG_1, SG_2 : SGr$, we have the following:

- The union of two SGs is inheritance acyclic provided: (i) the two individual SGs are inheritance acyclic, and (ii) the SGs are disjoint:

$$SG_1 \in SGr;\ SG_2 \in SGr;\ disj(SG_1, SG_2) \vdash acyclicG(\prec_G\ (SG_1 \cup_{SG} SG_2)) \\ \Leftrightarrow acyclicG\,(\prec_G\ SG_1) \wedge acyclicG(\prec_G\ SG_2)$$

- The union of two SGs is well-formed provided the individual SGs are well-formed also:

$$SG_1 \in SGr;\ SG_2 \in SGr \vdash (SG_1 \cup_{SG} SG_2) \in SGr$$

Properties. The following laws support this fact's theorems outlined above:

$$SG_1 \in SGr;\ SG_2 \in SGr;\ disj(SG_1, SG_2) \vdash disj(\prec_G\ SG_1, \prec_G\ SG_1)$$
$$SG_1 \in SGr;\ SG_2 \in SGr;\ disj(SG_1, SG_2) \vdash (\mathrm{dom} \prec_{SG_1} \cup \mathrm{ran} \prec_{SG_1}) \cap (\mathrm{dom} \prec_{SG_2} \cup \mathrm{ran} \prec_{SG_2}) = \varnothing$$

Proof. All theorems outlined above were proved in the Isabelle proof assistant. □

Definition 13 (SG Morphisms)**.** Given SG_1, SG_2 : SGr, a SG morphism $m : SG_1 \to SG_2$ is a pair of functions $m = (fv, fe)$ mapping nodes and edges, respectively.

The set of morphisms between two SGs, $SG_1 \to SG_2$, is defined as:

$$\begin{array}{l} \forall\, SG_1, SG_2 : SGr \bullet \\ SG_1 \to SG_2 = \{(fv, fe) \mid fv \in Ns_{SG_1} \to Ns_{SG_2} \wedge fe \in Es_{SG_1} \to Es_{SG_2} \\ \quad \wedge\ f_V \circ src^*_{SG_1} \subseteq src^*_{SG_2} \circ f_E \wedge f_V \circ tgt^*_{SG_1} \subseteq tgt^*_{SG_2} \circ f_E \wedge f_V \circ \preccurlyeq_{SG_1} \subseteq \preccurlyeq_{SG_2} \circ f_V\} \end{array}$$

The definition above requires the following: (a) there is a subset commuting for the extended source and target relations (src^* and tgt^*), which uses relational, rather than functional, composition; (b) there is a subset commuting for the extended inheritance relation to ensure that the morphism preserves inheritance information. □

Fact 5 (Category **SGraphs**)**.** We can form the category **SGraphs** by taking SGs as category objects (def. 11) and SG morphisms (def. 13) as category morphisms. We define the domain, co-domain, identity and composition of **SGraphs** as:

$$\begin{array}{ll} domCSG : GrMorph \to SGr & codCSG : GrMorph \to SGr \\ domCSG\, m = SG_1 \Leftrightarrow m \in SG_1 \to SG_2 & codCSG\, m = SG_2 \Leftrightarrow m \in SG_1 \to SG_2 \\ idCSG : SGr \to GrMorph & \circ_{CSG} : GrMorph \times GrMorph \to GrMorph \\ idCSG\, SG1 = m \Leftrightarrow m \in SG1 \to SG2 & m_1 \circ_{CSG} m_2 = m3 \Leftrightarrow m3 = m_1 \circ_G m_2 \end{array}$$

We define the set of all SG morphisms, a subset of $GrMorph$, as:

$$SGrMorph = \{m : GrMorph \mid \exists\, SG_1, SG_2 : SGr \bullet m \in SG_1 \to SG_2\}$$

The category **SGraphs** is defined as:

$$\mathbf{SGraphs} = (SGr, GrrMorph, domCSG, codCSG, idCSG, \circ_{CSG})$$

Proof. All required proofs were done in Isabelle. □

A.5 Fragments

Definition 14 (Fragment)**.** A fragment $F = (SG, tr)$ comprises a $SG : SGr$ and a total function $tr : EsRP_{SG} \twoheadrightarrow V$, mapping reference edges attached to proxies to referred nodes.

The base set of local fragments Fr_0, such that $F : Fr$, is defined as:

$$Fr_0 = \{(SG, tr) \mid SG \in SGr \wedge tr \in EsRP_{SG} \to V \wedge EsTy(SG, \{einh\}) \lhd src_{SG} \rhd NsP_{SG} = \varnothing\}$$

Above, $\lhd$ and $\rhd$ are domain and range restrictions, respectively. Last conjunct says that proxy nodes (NsP_{SG}) cannot have supertypes.

Several functions extract the components of a fragment:

$$\begin{array}{ll} sg : Fr_0 \to SGr & tgtr : Fr_0 \to SGr \\ sg(SG, tr) = SG & tgtr(SG, tr) = tr \end{array}$$

We introduce several functions and predicates to operate upon Fr_0: (a) *withRsG* gives the fragment's graph with the proxies conected to their actual references as defined in the fragment (function *tr*); (b) *refsG* yields a graph that gives proxies and their references; (c) *refs* gives the references relation derived from the *refsG* graph; (d) predicate *acyclicF* says whether the inheritance relation extended with refs is acyclic; (e) *refsOf* indicates the referred nodes of a given node; (f) *nonPRefsOf* indicates the non-proxy referred nodes of a given node.

$$\begin{array}{l} withRsG : Fr_0 \to Gr \quad refsG : Fr_0 \to Gr \quad refs : Fr \to V \leftrightarrow V \quad acyclicIF\_ : \mathbb{P}\, Fr_0 \\ withRsG(SG, tr) = (Ns_{SG} \cup \operatorname{ran} tr, Es_{SG}, src_{SG}, tgt_{SG} \oplus tr) \quad refsG\ F = restrict(withRsG_F, EsRP_F) \\ refs\ F = rel(refsG\ F) \qquad acyclicIF\ F \Leftrightarrow (\prec_F \cup refs_F) \in acyclic \\ refsOf : Fr_0 \to V \to \mathbb{P}\, V \qquad nonPRefsOf : Fr_0 \to V \to \mathbb{P}\, V \\ refsOf_F\ v = (refs_F)^+ (\!| \{v\} |\!) \quad nonPRefsOf_F\ v = \{v_2 : V \mid v_2 \in refsOf_F\ v \wedge \neg\, v_2 \in NsP_F\} \end{array}$$

Here, $\oplus$ denotes function overriding.

The actual set of fragments *Fr* is defined from Fr_0 as:

$$Fr = \{F : Fr_0 \mid (\forall\ v : NsP_F \bullet nonPRefsOf_F\ v \neq \varnothing) \wedge acyclicIF\ F\}$$

We require that all proxy nodes point a non proxy referred node, and the fragment's inheritance relation enriched with references is acyclic.

We introduce further functions and predicates to operation upon *Fr*: (a) $\rightsquigarrow$ gives all the representations of some node; (b) $\prec$ extends $\prec$ of SGs (def. 11) with $\rightsquigarrow$; (c) *repsOf* indicates the representatives of a given node.

$$\begin{array}{lll} \rightsquigarrow : Fr \to V \leftrightarrow V & \prec : Fr \to (V \leftrightarrow V) & repsOf : V \times Fr \to \mathbb{P}\, V \\ \rightsquigarrow_F = refs_F \cup (refs_F)^\sim & \prec_F = \prec_F \cup \rightsquigarrow_F & repsOf\ v\ F = \{v' : Ns_F \mid v \rightsquigarrow_F^* v'\} \end{array}$$

Auxiliary Definitions. Function $\preccurlyeq$ is the reflexive transitive closure of $\prec$ relation for fragments, and *disjFs* says whether two fragments are disjoint (if underlying SGs are disjoint):

$$\begin{array}{ll} \preccurlyeq : Fr \to (V \leftrightarrow V) & disjFs : \mathbb{P}(Fr \times Fr) \\ \preccurlyeq_F = (\prec\ F)^* & disjFs(F_1, F_2) \Leftrightarrow disj(sg\ F_1, sg\ F_2) \end{array}$$

Likewise, SG functions *clan*, *src** and *tgt** are extended for fragments by taking references into account:

$$\begin{array}{l} clan : V \times Fr \to \mathbb{P}\, V \quad src^* : Fr \to (E_L \leftrightarrow V_L) \quad tgt^* : Fr \to (E_L \leftrightarrow V_L) \\ clan(v, F) = \{v' : Ns_F \mid v' \preccurlyeq_F v\} \\ src^*\ F = \{(e, v) \mid e \in EsA_F \wedge v \in Ns_F \wedge (\exists\, v_2 : Ns_F \bullet v \in clan(v_2,\ F) \wedge (e, v_2) \in srcst(sg\ F))\} \\ tgt^*\ F = \{(e, v) \mid e \in EsA_F \wedge v \in Ns_F \wedge (\exists\, v_2 : Ns_F \bullet v \in clan(v_2,\ F) \wedge (e, v_2) \in tgtst(sg\ F))\} \end{array}$$

Properties. We proved in Isabelle, some healthiness conditions concerning Fragments. In particular, that $\prec$, $\preccurlyeq$, *clan*, *src** and *tgt** of fragments preserve the information of the corresponding SG functions and relations, and that $\preccurlyeq$ preserves the information of relation $\rightsquigarrow$:

$$\begin{array}{lll} \vdash \prec_{(sg\ F)} \subseteq \prec_F & \vdash \preccurlyeq_{(sg\ F)} \subseteq \preccurlyeq_F & \vdash clan(v, (sg\ F)) \subseteq clan(v, F) \\ \vdash src^*_{sg\ F} \subseteq src^*_F & \vdash tgt^*_{sg\ F} \subseteq tgt^*_F & \vdash \rightsquigarrow_F \subseteq \preccurlyeq_F \\ \vdash v \in clan(v, F) & v_1 \prec_{sg\ F} v_2 \vdash v_1 \in clan(v_2, F) & \end{array}$$

The following laws are related to fragments:

$$\vdash x \rightsquigarrow_F x \quad x \rightsquigarrow_F y \vdash y \rightsquigarrow_F x \quad \vdash x \preccurlyeq_F x$$

Proof. All laws and healthiness conditions outlined above were proved in the Isabelle proof assistant. □

Definition 15. The union composition of fragments $F_1, F_2 : Fr$ is defined as:

$$F_1 \cup_F F_2 = (sg\ F_1 \cup_{SG} sg\ F_2, tgtr_{F_1} \cup tgtr_{F_2})$$

The union of two fragments is the union of the fragments' SGs (function sg and operator $\cup_{SG}$ of def. 12) and union of fragments' target references functions (function $tgtr$).

Properties. The following laws are related to fragment union:

$$F_1 \in Fr;\ F_2 \in Fr \vdash sg(F_1 \cup_F F_2) = sg\ F_1 \cup_{SG} sg\ F_2 \quad F_1 \in Fr;\ F_2 \in Fr \vdash tgtr(F_1 \cup_F F_2) = tgtr\ F_1 \cup tgtr\ F_2$$
$$F_1 \in Fr;\ F_2 \in Fr \vdash \prec (F_1 \cup_F F_2) = \prec_{F_1} \cup \prec_{F_2}$$

Proof. All laws given above were proved in the Isabelle proof assistant. □

Fact 6. Given fragments $F1, F2 : Fr$, we have the following:

- The union of two fragments is inheritance acyclic provided that individually the fragments are acyclic also:

  $$F_1 \in Fr;\ F_2 \in Fr;\ disjFs(F_1, F_2) \vdash acyclicIF\ (F_1 \cup_F F2) \Leftrightarrow acyclicIF\ F_1 \wedge acyclicIF\ F2$$

- The union of two fragments is well-formed provided the individual fragments are well-formed also (a closure property of fragment union):

  $$disjFs(F_1, F_2) \vdash (F_1 \cup_F F2) \in Fr \Leftrightarrow F_1 \in Fr \wedge F2 \in Fr$$

- The inheritance graph of every fragment obtained after resolving the references is acyclic:

  $$F \in Fr \vdash acyclicG(replaceG\ (inhG\ F)\ consSubOfFr\ F)$$

Proof. These three theorems were proved in Isabelle. The proof outlines are as follows:

- First theorem shows impossibility of direct cycles (as per F3, Fig. 4.2(c)) in fragment compositions. The fragment's graph with references reduces to relations; the proof's difficulty lies in the fact that transitive closure is not distributive with respect to set union in general: $(r \cup s)^+ \neq r^+ \cup s^+$ (the equality only holds when the relations are disjoint, see fact 1, above). This proof's key lies in a smaller theorem proved in Isabelle that considers restrictions of fragments setting: the domains of relations are disjoint, only one fragment references the other; given this, we obtain: $(r \cup s)^+ = r^+ \cup s^+ \cup (r^+ ⨾ \operatorname{ran} r \lhd s^+)$ (see fact 1, above).

- The second theorem, a closure property of fragment union, is proved by using the previous theorem.

- Third theorem shows impossibility of indirect cycles (as per F4 and F5 in Fig. 4.2(c)) in a well-formed fragment. The proof resorts to replacement graphs: given a fragmented graph, we obtain a graph that replaces proxies by referred nodes. The proof shows that if fragment is well-formed then this graph is always acyclic; they key hypothesis is the restriction forbidding proxies from inheriting.

□

Definition 16 (Fragment Morphisms)**.** A fragment morphism $m : F_1 \to F_2$ is a mapping from $F_1 : Fr$ to fragment $F_2 : Fr$. It consists of a pair of functions $m = (fv, fe)$ mapping nodes and edges, respectively. The set of fragment morphisms is defined as:

$$\begin{aligned}
&\forall F_1, F_2 : Fr \bullet \\
&F_1 \to F_2 = \{(fv, fe) \mid fv \in Ns_{F_1} \to Ns_{F_2} \wedge fe \in Es_{F_1} \to Es_{F_2} \\
&\quad \wedge f_V \circ src^*_{F_1} \subseteq src^*_{F_2} \circ f_E \wedge f_V \circ tgt^*_{F_1} \subseteq tgt^*_{F_2} \circ f_E \wedge f_V \circ \preccurlyeq_{F_1} \subseteq \preccurlyeq_{F_2} \circ f_V\}
\end{aligned}$$

Above, we restate the same conditions as SG morphisms (def. 13), using the updated functions and relations from def. 14 that cater to the semantics of references. □

A.6 Global Fragment Graphs

Definition 17 (Global Fragment Graphs)**.** Set ExtEdgeTy defines the extension edges of kind imports and continues (common to clusters and global fragment graphs):

$$ExtEdgeTy = \{eimpo, econti\}$$

A GFG is a pair $GFG = (G, et)$, where $G : Gr$ is a graph (definition 3), and $et : Es_G \to ExtEdgeTy$ is a colouring function mapping edges to extension edge types. Each node of the GFG must have a corresponding self edge. The imports and continues relations taken together (the edges of the graph) and excluding the self edges must be acyclic. The set of valid GFGs is defined as:

$$\begin{aligned}
&GFGr = \{(G, et) \mid G \in Gr \wedge et \in Es_G \to ExtEdgeTy \wedge Ns_G \subseteq V_F \wedge Es_G \subseteq E_F \\
&\quad \wedge (\forall v \in Ns_G \bullet \exists e : Es_G \bullet src_G\ e = tgt_G\ e = v) \wedge acyclicG(restrict(Es_G \setminus EsId_G))\}
\end{aligned}$$

Auxiliary Definitions. We introduce functions to extract components of a GFG:

$$\begin{array}{ll}
gr : GFGr \to Gr & fet : GFGr \to E \nrightarrow ExtEdgeTy \\
gr(G, et) = G & fet(G, et) = et
\end{array}$$

□

Definition 18 (Global Fragment Morphisms)**.** A GFG morphism $m : GFG_1 \to GFG_2$ defines a specific mapping between GFGs (definition 17) from $GFG_1 : GFGr$ to $GFG_2 : GFGr$. The set of GFG-morphisms is defined as:

$$\begin{aligned}
&\forall G_1, G_2 : Gr;\ et_1, et_2 : E \nrightarrow ExtEdgeTy \bullet \\
&\quad (G_1, et_1) \to (G_2, et_2) = G_1 \to G_2 \cap \{(fv, fe) \mid et_2 \circ fe = et_1\}
\end{aligned}$$

Here, we require that GFG morphisms are normal graph morphisms that preserve the colouring of the edges. □

Definition 19 (Fragment to GFG Morphisms)**.** A fragment to GFG $m : Fr \to GFG$ maps local fragment nodes to GFG nodes. The set of such morphisms is defined as:

$$\begin{aligned}
&\forall F : Fr;\ GFG : GFGr \bullet \\
&\quad F \to GFG = \{(fv, fe) \mid fv \in Ns_F \to Ns_{GFG} \wedge fe \in Es_F \to Es_{GFG} \\
&\qquad \wedge (fv, fe) \in (withRsG\ F) \to (gr\ GFG) \\
&\qquad \wedge Ns_{GFG} \neq \varnothing \Rightarrow \exists\ vfg : Ns_{GFG} \bullet fv \,(\!|\, Ns_F \,|\!)\, = \{vfg\} \\
&\qquad\quad \wedge Es_F \setminus EsR_F \neq \varnothing \Rightarrow \exists\ efg : Es_{GFG} \bullet fe \,(\!|\, Es_F \setminus EsR_F \,|\!)\, = \{efg\} \wedge src_{GFG}\ efg = vfg \wedge tgt_{GFG}\ efg = vfg)
\end{aligned}$$

Above, we say that such a morphism is a graph morphism between the fragment's underlying graph and GFG. We require that all nodes and non-reference edges of the fragment are mapped to the same node and edge in the GFG, where the edge is a self-edge. We also require that the target of reference edges are mapped to this same node. □

A.7 Cluster Graphs

Definition 20 (Cluster Graphs)**.** The set of cluster edge kinds is formed by considering the extension edge kinds added with the containment relation:

$$CGEdgeTy = ExtEdgeTy \cup \{econta\}$$

A cluster graph is a pair $CG = (G, et)$, comprising a graph $G : Gr$ (definition 3) and a colouring function $et : Es_G \rightarrow CGEdgeTy$ mapping edges to cluster edge types (set $CGEdgeTy$). The set of valid cluster graphs is defined as:

$$\begin{aligned}&CGr = \{(G, et) \mid G \in Gr \wedge et \in Es_G \rightarrow CGEdgeTy\\&\quad \wedge acyclicG(restrict(G, et^{\sim} (\!| \{eimpo, econti\} |\!) \setminus EsId_G))\\&\quad \wedge rel(restrict(G, et^{\sim} (\!| \{econta\} |\!) \setminus EsId_G)) \in forest\}\end{aligned}$$

Above, we require that the relations formed by the imports and continues edges, subtracted with the self edges, must be acyclic, and that the relation formed by the contains edges, subtracted with the self edges, must constitute a *forest* (see def. 1).

Auxiliary Definitions. The next functions extract the components of a cluster graph:

$$\begin{array}{ll} gr : CGr \rightarrow Gr & cety : CGr \rightarrow E \nrightarrow ExtEdgeTy \\ gr(G, et) = G & cety(G, et) = et \end{array}$$

□

Definition 21 (Cluster Graph Morphisms)**.** A cluster graph morphism $m : CG_1 \rightarrow CG_2$ maps cluster graphs $CG_1, CG_2 : CGr$ (definition 20). The set of such morphisms is defined as:

$$\begin{aligned}&\forall\, CG_1, CG_2 : CGr \bullet\\&\quad CG_1 \rightarrow CG_2 = \{(fv, fe) \mid fv \in Ns_{CG_1} \rightarrow Ns_{CG_2} \wedge fe \in Es_{CG_1} \rightarrow Es_{CG_2}\\&\qquad \wedge (fv, fe) \in (gr\ CG_1) \rightarrow (gr\ CG_2) \wedge (cety\ CG_2) \circ fe = cety\ CG_1\}\end{aligned}$$

This requires such morphisms to be normal graph morphisms that preserve the colouring of the edges. □

Definition 22 (GFG to Cluster Graph Morphisms)**.** A GFG to cluster graph morphism $m : GFG \rightarrow CG$ maps a fragment graph $GFG : GFGr$ (definition 17) to a cluster graph $CG : CGr$ (definition 20). The set of such morphisms is defined as:

$$\begin{aligned}&\forall\, GFG : GFGr : CG : CGr \bullet\\&\quad FG \rightarrow CG = \{(fv, fe) \mid fv \in Ns_{GFG} \rightarrow Ns_{CG} \wedge fe \in Es_{GFG} \rightarrow Es_{CG}\\&\qquad \wedge (fv, fe) \in (gr\ GFG) \rightarrow (gr\ CG) \wedge (cety\ CG) \circ fe = fety\ GFG\}\end{aligned}$$

This requires such morphisms to be normal graph morphisms that preserve the colouring of the edges. □

A.8 Models

Definition 23 (Models)**.** A model is quadruple $M = (GFG, CG, mc, fd)$, consisting of a $GFG : GFGr$, a $CG : CGr$, a morphism $mc : GFG \rightarrow CG$, and a mapping from GFG nodes to fragment definitions $fd : Ns_{GFG} \rightarrow Fr$.

The base set of all models, such that $M \in Mdl0$, is defined as:

$$\begin{aligned}&Mdl_0 = \{(GFG, CG, mc, fd) \mid GFG \in GFGr \wedge CG \in CGr \wedge mc \in GFG \rightarrow CG\\&\quad \wedge fd \in Ns_{GFG} \rightarrow Fr\}\end{aligned}$$

We define functions to extract the different components of a model:

$$
\begin{array}{ll}
gfg : Mdl_0 \to GFGr & cg : Mdl_0 \to CGr \\
gfg(GFG, CG, mc, fd) = GFG & cg(GFG, CG, mc, fd) = CG \\
mcg : Mdl_0 \to GrMorph & fdef : Mdl_0 \to (V \rightarrow\!\!\!\!\!\rightarrow Fr) \\
mcg(GFG, CG, mc, fd) = mc & fdef(GFG, CG, mc, fd) = fd
\end{array}
$$

Function UFs returns the fragment that results from the union of all fragments of a model. $from_V$ indicates to which fragment a local node belongs to:

$$
\begin{array}{ll}
UFs : Mdl_0 \to Fr & UFs_0 : \mathbb{P}_1\, Fr \to Fr \\
UFs\, M = UFs_0(\operatorname{ran}(fdef\, M)) & UFs_0\{F\} = F \\
 & UFs_0\{F\} \cup Fs = F \cup_F (UFs_0\, Fs)
\end{array}
$$

$$
\begin{array}{l}
from_V : V_L \times Mdl \rightarrow\!\!\!\!\!\rightarrow V_F \\
from_V(vl, M) = vf \Leftrightarrow vl \in Ns(fdef\, vf)
\end{array}
$$

Function $mUMFsToGFG$ builds a morphism from the union of all fragments of a model to the given model's GFG, which involves other auxiliary functions (such as $consFToGFG$):

$$
\begin{array}{l}
consFToGFG : V_F \times Mdl_0 \to GrMorph \\
consFToGFG(vf, M) = (fv, fe) \Leftrightarrow \exists\, F : Fr;\ GFG : GFGr \bullet F = fdef\, M\, vf \wedge GFG = gfg\, M \\
\quad \wedge\ fv \in Ns_F \to Ns_{GFG} \wedge fe \in Es_F \to Es_{GFG} \wedge vf \in Ns_{GFG} \\
\quad \wedge\ (\exists\, ef : Es_{GFG} \bullet src_{GFG}\, ef = tgt_{GFG}\, ef = vf \wedge fv = Ns_F \times \{vf\} \\
\qquad \wedge\ fe = (Es_F \setminus EsR_F) \times \{ef\} \cup consFToGFGRefs(vf, EsR_F, M)) \\
consFToGFGRefs : V_F \times \mathbb{P}\, E_L \times Mdl_0 \to E \rightarrow\!\!\!\!\!\rightarrow E \\
consFToGFGRefs(vf, \{\}, M) = \{\} \\
consFToGFGRefs(vf, \{el\} \cup E_r, M) = fe \Leftrightarrow \\
\quad \exists\, F : Fr;\ GFG : GFGr \bullet F = fdef\, M\, vf \wedge GFG = gfg\, M \\
\quad \wedge\ (\exists\, ef : Es_{FG} \bullet src_{GFG}\, ef = vf \wedge tgt_{GFG}\, ef = from_V(tgtr_F\, el, M) \\
\quad \wedge\ fe = \{e_L \mapsto e_f\} \cup consFToGFGRefs(vf, E_r, M)) \\
UMToGFG : Mdl_0 \to GrMorph \\
UMToGFG\, M = buildUFsToGFG(fdef\, M, M) \\
buildUFsToGFG : (V \rightarrow\!\!\!\!\!\rightarrow Fr) \times M \to GrMorph \\
buildUFsToGFG(\{vf \mapsto F\}, M) = consFToGFG(vf, M) \\
buildUFsToGFG(\{vf \mapsto F\} \cup fd, M) = consFToGFG(vf, M) \cup_{GM} buildUFsToGFG(fd, M)
\end{array}
$$

The set of all models Mdl is defined as:

$$
\begin{array}{l}
Mdl = \{M : Mdl_0 \mid UMToGFG\, M \in UFs\, M \to (gfg\, M) \\
\quad \wedge\ (\forall\, vf_1, vf_2 : Ns(gfg\, M) \mid vf_1 \neq vf_2 \bullet disjFs(fdef_M\, vf_1, fdef\, M\, vf_2))\}
\end{array}
$$

Above, we say that the morphism obtained from $UMToGFG$ must be a local fragment to GFG morphism, and that all fragments of a model are disjoint. □

A.9 Category Theory

Definition 24 (Pushout)**.** Given a category $\mathcal{C} : Cat$ (definition 8) and $\mathcal{C}$-morphisms $f : A \to_C B$ and $g : A \to_C C$, a possible pushout (D, f', g') over f and g is defined by:

- A pushout object $D \in obs_C$,
- and morphisms $f' : C \to_C D$ and $g' : B \to_C D$, such that $f' \circ_C g = g' \circ_C f$

Based on this, we define the set of possible pushouts as:

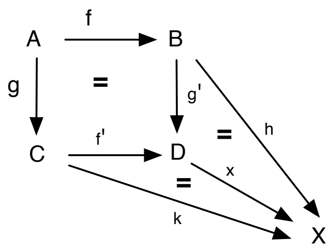
$$\begin{array}{l}\forall\, C : Cat \bullet \forall\, f, g : morphs_C \bullet \\ \quad PPO\ C\,(f, g) = \{(D, f', g') \mid D \in obs_C \wedge f' \in morphs_C \wedge g' \in morphs_C \\ \qquad \wedge\ dom_C\, f = dom_C\, g \wedge dom_C\, f' = cod_C\, g \wedge dom_C\, g' = cod_C\, f \wedge f' \circ_C g = g' \circ_C f\}\end{array}$$

In the following, we write $PPO_C\,(f, g)$ to mean $PPO\ C(f, g)$.

Given a category $\mathcal{C} : Cat$ and $\mathcal{C}$-morphisms $f : A \rightarrow_C B$ and $g : A \rightarrow_C C$, a push out $po = (D, f', g')$ is a unique object from the set of possible pushouts $po : PPO_C(f, g)$, such that for any other push out $po' : PPO_C(f, g)$, where $po' = (X, k, h)$, there is a unique morphism $x : D \rightarrow_C X$; a pushout is defined as:

$$\begin{array}{l}\forall\, C : Cat \bullet \forall\, f, g : morphs_C \bullet \\ \quad PO\ C\,(f, g) = (\mu\, D : obs_C;\ f', g' : morphs_C \mid (D, f', g') \in PPO_C\,(f, g) \\ \qquad \wedge\ (\forall\, X : obs_C;\ k, h : morphs_C \bullet \\ \qquad (X, k, h) \in PPO_C\,(f, g) \wedge \exists\, x : D \rightarrow_C X \bullet x \circ_C f' = k \wedge x \circ_C g' = h)))\end{array}$$

The following diagram defines a pushout:



□

Definition 25 (Morphisms of Graphs to Categories)**.** A morphism $G \rightarrow \mathcal{C}$ from a graph $G : Gr$, such that $G = (V_G, E_G, s, t)$ (definition 3), to a category $\mathcal{C} : Cat$, such that $\mathcal{C} = (O_C, M_C, dm, cd, id_C, \circ)$ (definition 8) is a pair of functions (mv, me) with $mv : Ns\ G \rightarrow obs_C$ and $me : Es\ G \rightarrow morphs_C$, mapping nodes to objects and edges to mo, respectively. We require that the underlying diagram commutes, and so: $mv \circ s = dm \circ me$ and $mv \circ t = cd \circ me$.

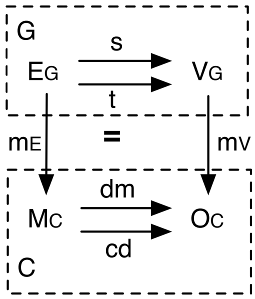
The set of all possible graph to category morphisms is defined as:

$$MorphGr2Cat = \{(mv, me) \mid mv \in V \twoheadrightarrow O \wedge fe \in E \twoheadrightarrow M\}$$

The set of valid morphisms between a graph and category $m : G \rightarrow \mathcal{C}$, such that $m \subseteq MorphGr2Ca$, is defined as:

$$\begin{array}{l}\forall\, G : Gr;\ C : Cat \bullet \\ \quad G \rightarrow C = \{(mv, me) \mid mv \in Ns\ G \rightarrow obs_C \wedge me \in Es\ G \rightarrow morphs_C \\ \quad \wedge\ mv \circ src\ G = dom_C \circ me \wedge mv \circ tgt\ G = cod_C \circ me\}\end{array}$$

Above the last two equations ensure that the underlying diagram commutes:



Auxiliary Definitions. The following functions extract the individual components of a graph morphism:

$m_V : MorphGr2Cat \to V \twoheadrightarrow O$
$m_V(mv, me) = mv$
$m_E : MorphGr2Cat \to E \twoheadrightarrow M$
$m_E(mv, me) = me$

□

Definition 26 (Diagram)**.** For our purposes, a diagram is a collection of vertices and directed edges, that are consistently mapped to the objects and morphisms of the category to which they correspond.

A diagram is, therefore, a triple $D = (\mathcal{C}, G, m_D)$ made up of a category $\mathcal{C} : Cat$ (definition 8), a graph $G : Gr$ (definition 3) and a graph to category morphism $m : G \to C$ (definition 25).

We define the set of diagrams as:

$Diag = \{(\mathcal{C}, G, m) \mid \mathcal{C} \in Cat \wedge G \in Gr \wedge m \in G \to C\}$

Auxiliary Definitions.

We define functions to yield the components of a diagram:

$gr : Diag \to Gr$
$gr(\mathcal{C}, G, m) = G$
$cat : Diag \to Cat$
$cat(\mathcal{C}, G, m) = \mathcal{C}$
$morph : Diag \to GrToCatMorph$
$morph(\mathcal{C}, G, m) = m$

The function *catObs* and *catMorphs* extract, respectively, the set of underlying category objects and the set of underlying category morphisms from a diagram:

$catObs : Diag \to \mathbb{P}\, O$
$catObs(\mathcal{C}, G, m) = \operatorname{ran}(m_V\ m)$
$catMorphs : Diag \to \mathbb{P}\, O$
$catMorphs(\mathcal{C}, G, m) = \operatorname{ran}(m_V\ m)$

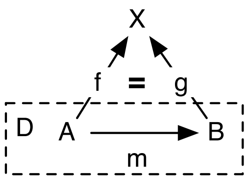
□

Definition 27 (Cocone and colimit)**.** A cocone for a diagram D (definition 26) in a category C is a C-object X and a collection of morphisms that map the objects of the diagram to this object; the set of cocones is defined as:

$$
\begin{array}{l}
\forall\, D : Diag \bullet \\
\quad CC(D) = \{(X, ms) \mid \exists\, \mathcal{C} : cat\ D \bullet X \in obs_C \wedge ms \in \mathbb{P}(morphs_C) \\
\quad \wedge\ (\forall\, m : ms \bullet dom_C\ m \in catObs\ D \wedge cod_C\ m = X)\}
\end{array}
$$

A cocone is valid provided that the morphisms of the diagram and those of the cocone commute:

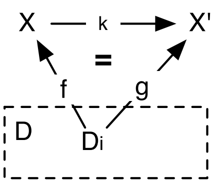
$$
\begin{array}{l}
\forall\, D : Diag;\ X : O;\ ms : \mathbb{P}\, M \\
\quad (X, ms) \in ValCC\ D \Leftrightarrow (X, ms) \in CC\ D \\
\quad\quad \wedge\ (\forall\, m : catMorphs\ D \bullet \exists\, f, g : ms;\ C : Cat \bullet \\
\quad\quad\quad C = cat\ D \wedge f \in dom_C\ m \rightarrow_C X \wedge g \in cod_C\ m \rightarrow_C X \wedge g \circ_C m = f)
\end{array}
$$



A colimit is them a cocone $cc = (X, ms)$ with the universal property that for any other cocone $cc' = (X', ms')$ there is a unique morphism $k : X \to X'$; we define the colimit as:

$$
\begin{array}{l}
\forall\, D : Diag \bullet \\
\quad colimit\ D = (\mu\, X : O;\ ms : \mathbb{P}\, M \mid (X, ms) \in ValCC\ D \\
\quad \wedge\ (\exists\, \mathcal{C} : Cat \bullet C = cat\ D \\
\quad\quad \wedge\ (\forall\, X' : obs_C;\ ms' : morphs_C \mid X \neq X' \wedge (X', ms') \in ValCC\ D \bullet \\
\quad\quad\quad \exists\, k : X \to X' \bullet \forall\, f : ms;\ g : ms' \mid dom_C\, f = dom_C\ g \bullet k \circ_C f = g
\end{array}
$$

That is, the underlying diagram commutes:



□

A.10 Colimit composition

Definition 28 (Fragment Composition Diagram)**.** The composition diagram of a fragment is defined through function *compDiag*. The diagram is built in the following steps:

1. It starts by building a diagram with a node corresponding to the fragment that is being composed (function *buildStartDiag*).

2. It adds to the diagram all the nodes corresponding to the fragments that the fragment to compose is import dependent on (function *diagDepNodes* applied to function *importsOf*) and continues dependent on (function *diagDepNodes* applied to function *continuesOf*).

3. It adds to the diagram all interface graphs and corresponding morphisms (function *diagMorphisms*). This involves building the interface graph and morphisms corresponding to merges (function *diagMerges*) and references (function *diagRefs*); the latter deals with both imports and continuations.

We start by introducing the category of graphs, which is the category on which we perform the colimit-based compositions of fragments. We introduce an identity operator for Graphs:

$$
\begin{aligned}
& id_{Gr} : Gr \to GrMorph \\
& id_{Gr}\ G = GM \Leftrightarrow GM \in G \to G \wedge GM = (\mathrm{id}(nodesG), \mathrm{id}(edgesG))
\end{aligned}
$$

The actual category of graphs is defined as:

$$
\begin{aligned}
& GrCat : Cat \\
& GrCat = (Gr, GrMorph, id_{Gr}, \circ_G)
\end{aligned}
$$

Function *compDiag* is defined as:

$$
\begin{aligned}
& compDiag : V_F \times Mdl \to Diag \\
& compDiag(vf, M) = D' \Leftrightarrow \exists\, D_0, D_1 D_2 : Diag \bullet \\
& \quad buildStartDiag(vf, M) = D_0 \\
& \quad \wedge\ diagDepNodes(importsOf(vf, (m\_fg\ M)), M, D_0) = D_1 \\
& \quad \wedge\ diagDepNodes(continuesOf(vf, (m\_fg\ M)), M, D_1) = D_2 \\
& \quad \wedge\ diagMorphisms(vf, M, D_2) = D'
\end{aligned}
$$

Function *buildStartDiag* is defined as:

$$
\begin{aligned}
& buildStartDiag : V_F \times Mdl \to Diag \\
& buildStartDiag(vf, M) = addNodeToDiag(vf, srcGr((m\_fdef\ M)\ vf), emptyDiag\ GrCat)
\end{aligned}
$$

Function *diagDepNodes* is defined as:

$$
\begin{aligned}
& diagDepNodes : \mathbb{P}\ V_F \times Mdl \times Diag \to Diag \\
& diagDepNodes(\varnothing, M, D) = D \\
& diagDepNodes(\{vf_1\} \cup vfs, M, D) = D' \Leftrightarrow \\
& \quad \exists\, D_0, D_1 D_2 : Diag \bullet \\
& \quad addNodeToDiag(vf_1, gr\ ((m\_fdef\ M) vf_1), D) = D_0 \\
& \quad \wedge\ diagDepNodes(importsOf(vf_1, (m\_fg\ M)), M, D_0) = D_1 \\
& \quad \wedge\ diagDepNodes(continuationsOf(vf_1, (m\_fg\ M)), M, D_1) = D_2 \\
& \quad \wedge\ diagDepNodes(vfs, M, D_2) = D'
\end{aligned}
$$

Function *diagMorphisms* is defined as:

$$
\begin{array}{l}
diagMorphisms : V_F \times Mdl \times Diag \rightarrow Diag \\
diagMorphisms_0 : (V_F \times Mdl \times Diag \times \mathbb{P}\ V_F) \rightarrow Diag \times \mathbb{P}\ V_F) \\
diagMorphisms(vf, M, D) = diagMorphisms_0(vf, M, D, \varnothing) \\
diagMorphisms_0(vf, M, D, pvfs) = (D', pvfs') \Leftrightarrow \\
\quad \exists\, F : Fr;\ D_1 D_2 : Diag \bullet F = ((m\_fdef\ M)vf) \\
\quad \wedge\ diagRefs(vf, importsOf(vf, m\_fg\ M) \cup continuesOf(vf, m\_fg\ M), GE, D) = D_1 \\
\quad \wedge\ diagMorphismsSet(importsOf(vf, m\_fg\ M) \\
\qquad\quad \cup continuesOf(vf, fg\ M), GE, D_1, pvfs \cup \{vf\}) = (D', pvfs') \\
diagMorphismsSet : \mathbb{P}\ V_F \times Mdl \times Diag \times \mathbb{P}\ V_F \rightarrow (Diag \times \mathbb{P}\ V_F) \\
diagMorphismsSet(\varnothing, M, D, P) = (D, P) \\
diagMorphismsSet(\{vf_1\} \cup vfs, M, D, P) = diagMorphismsSet(vfs, M, D, P) \Leftrightarrow vf_1 \in P \\
diagMorphismsSet(\{vf_1\} \cup vfs, M, D, P) = (D', P') \Leftrightarrow \neg\ vf_1 \in P \\
\quad \wedge\ diagMorphisms_0(vf1, M, D, P) = (D'', P'') \\
\quad \wedge\ diagMorphismsSet(vfs, M, D'', P'') = (D', P')
\end{array}
$$

Function *diagRefs* is defined as:

$$
\begin{array}{l}
HasImpRefs\_ : \mathbb{P}(V_F \times V_F \times Mdl) \\
HasImpRefs(vf_1, vf_2, M) \Leftrightarrow \exists\, F_1, F_2 : Fr \bullet \\
\quad F_1 = (m\_fdef\ M)\ vf_1 \wedge F_2 = (m\_fdef\ M)\ vf_2 \wedge (refs\ F_1) \rhd (nodes\ F2) \neq \varnothing
\end{array}
$$

$$
\begin{array}{l}
diagRefs : V_F \times \mathbb{P}\ V_F \times Mdl \times Diag \rightarrow Diag \\
diagRefs(vf_1, \varnothing, M, D) = D \\
diagRefs(vf_1, \{vf_2\} \cup svf, M, D) = diagRefs(vf_1, svf, M, D) \Leftrightarrow \neg\ HasImpRefs(vf_1, vf_2, M) \\
diagRefs(vf_1, \{vf_2\} \cup svf, M, D) = D' \Leftrightarrow HasImpRefs(vf_1, vf_2, M) \\
\quad \wedge\ (\exists\, F_1, F_2 : Fr;\ GI : Gr;\ vfi : VF;\ m_1, m_2 : GrMorph;\ e_1, e_2 : E;\ D_0, D_1, D_2 : Diag \bullet \\
\qquad F_1 = (m\_fdef\ M)\ vf_1 \wedge F_2 = (m\_fdef\ M)\ vf_2 \\
\qquad \wedge\ GI = (\mathrm{dom}((refs\ F_1) \rhd (nodes\ F2)), \varnothing, \varnothing) \\
\qquad \wedge\ m_1 \in GI \rightarrow srcGr\ F_1 \wedge m_1 = (Id(\mathrm{dom}((refs\ F1) \rhd nodesF_2)), \varnothing) \\
\qquad \wedge\ m_2 \in GI \rightarrow srcGr\ F_2 \wedge m_2 = ((refs\ F1) \rhd nodesF_2, \varnothing) \\
\qquad \wedge\ \neg\ vfi \in nodes\ (gr\ D) \wedge addNodeToDiag(vfi, GI, D) = D_0 \\
\qquad \wedge\ \neg\ \{e_1, e_2\} \subseteq edges(gr\ D_0) \wedge addEdgeToDiag(e_1 vfi, vf_1, m_1, D_0) = D_1 \\
\qquad \wedge\ addEdgeToDiag(e_2, vfi, vf_2, m_2, D_1) = D_2 \wedge D' = diagRefs(vf_1, vfs, GE, D_2))
\end{array}
$$

□

A.11 Typed Structural Graphs

Definition 29 (Type Structural Graphs)**.** A type SG is a pair $TSG = (SG, iet)$ made up of a structural graph $SG : SGr$ (definition 9) and a function $iet : edges(SG) \rightarrow SGET$ mapping edges to the instances edge types being prescribed, according to the SG edge types defined by set SGET (definition 9).

The set of all type SGs is defined as:

$$TySGr = \{(SG, iet) \mid SG \in SGr \wedge iet \in EsA(SG) \rightarrow SGETy\}$$

Auxiliary Definitions. Next functions extract different components of a type SG:

$$
\begin{array}{l}
sgr : TySGr \rightarrow SGr \\
sgr(SG, iet) = SG
\end{array}
$$

Next function extracts the set of edges that prescribe a particular edge type:

$EsOfTy : TySGr \times SGETy \to \mathbb{P}\, E$
$EsOfTy((SG, iet), ety) = iet^{\sim} (\!| \{ety\} |\!)$

Next functions extract functions of SGs to type SGs:

$Ns_A : TySGr \to \mathbb{P}\, V$
$Ns_A(TSG) = Ns_A(sgr\ TSG)$
$Es_A : TySGr \to \mathbb{P}\, V$
$Es_A(TSG) = Es_A(sgr\ SGT)$
$Es_C : TySGr \to \mathbb{P}\, V$
$Es_C(TSG) = Es_C(sgr\ SGT)$
$srcm : TySGr \to \mathbb{P}\, V$
$srcm(TSG) = srcm(sgr\ SGT)$
$tgtm : TySGr \to \mathbb{P}\, V$
$tgtm(TSG) = tgtm(sgr\ SGT)$

□

Definition 30 (Typed Structural Graphs)**.** A typed SG is a triple $SGT = (SG, TSG, type)$, consisting of structural graph $SG : SGr$ (definition 11) defining the typed graph, a type structural graph $TSG : TySGr$ defining the type graph, and a structural graph morphism $type : SG \to (sgt\ TSG)$ that maps elements of SG to their types (definition 13), which ensures that the edge types prescribed by the type SG are consistent with the types of the edges in the instance SG.

The set of typed structural graphs is defined as:

$SGTy = \{(SG, TSG, type) \mid SG \in SGr \land TSG \in TySGr \land type \in SG \to (sgr\ TSG)\}$

Auxiliary Definitions. We define functions to extract the different components of a typed structural graph:

$srcGr : SGTy \to SGr$
$srcGr(SG, TSG, type) = SG$
$tyGr : SGTy \to TySGr$
$tyGr(SG, TSG, type) = TSG$
$tymorph : SGTy \to SGMor$
$tymorph(SG, TSG, type) = type$

We extend the functions Ns, Es, src and tgt of SGs by considering that they yield the nodes and edges of the source SG:

$Ns : SGTy \to \mathbb{P}\, V$
$Ns(SGT) = Ns(srcGr\ SGT)$
$Es : SGTy \to \mathbb{P}\, E$
$Es(SGT) = Es(srcGr\ SGT)$
$src : SGTy \to (E \nrightarrow V)$
$src(SGT) = src(srcGr\ SGT)$
$tgt : SGTy \to (E \nrightarrow V)$
$tgt(SGT) = tgt(srcGr\ SGT)$

Remark. Untyped SGs can be represented by considering a trivial type graph, with one node and one edge. All nodes and edges of the untyped graph will have therefore the same type.
□

Definition 31 (SG Conformance)**.** We introduce several predicates to check the conformance of a typed structural graph. First predicate checks that edge types of instance fragment conform with edge types prescribed by type fragment:

$$instanceEdgeTypesOk(SG, TSG, type) \Leftrightarrow iety_{TSG} \circ (fE\ type) = ety_{SG}$$

Second predicate checks that abstract nodes do not have any direct instances:

$$abstractNoDirectInstances(SG, TSG, type) \Leftrightarrow ((f_V\ type)^{\sim} (\!| nodes_A(TSG) |\!)) = \varnothing$$

This says that the set of instances of abstract nodes (obtained from the inverse of the type morphism) must be empty.

Third predicate checks that instances of containment edges do not allow contained nodes to be shared:

$$\begin{aligned}&containmentNoSharing(SG, TSG, type) \Leftrightarrow\\ &\quad ((f_E\ type)^{\sim} (\!| edges_C(TSG) |\!)) \lhd tgt^*(SG) \in EinjrelV\end{aligned}$$

This requires that the target function of instances of containment edges is injective (set *injrel* of definition 1), and so no two edges can have the same target.

Fourth predicate checks that the multiplicity constraints prescribed by the typed structural graph are satisfied in the instances:

$$\begin{aligned}&instMultsOk(SG, TSG, type) \Leftrightarrow \forall\, te : edges_A(TSG) \bullet\\ &\quad \exists\, r : V \leftrightarrow V \bullet r = rel(restrict(gr\ SG, (f_E\ type)^{\sim} (\!| \{te\} |\!)))\\ &\qquad \wedge \forall\, v : \mathrm{dom}\, r \bullet multOk(r (\!| \{v\} |\!), (srcm\ TSG) te)\\ &\qquad \wedge \forall\, v : \mathrm{ran}\, r \bullet multOk(r^{\sim} (\!| \{v\} |\!), (tgtm\ TSG) te)\end{aligned}$$

This predicate obtains the relation that is induced by the edges that are instances of the association edges of the graph (function *rel*).

Fifth predicate checks that the containment relation at the instance level is acyclic:

$$instContainmentAcyclic(SG, TSG, type) \Leftrightarrow acyclicGr(restrict(gr\ SG, (f_E\ type)^{\sim} (\!| edges_C(TSG) |\!)))$$

This says that the relation formed by all edges that are instances of containments must be acyclic. This is expressed by resorting to the predicate *acyclic* (definition 1)

There is a summary predicate that checks that typed structural graph are conformant:

$$\begin{aligned}&isConformable(SGT) \Leftrightarrow abstractNoDirectInstances(SGT) \wedge containmentNoSharing(SGT)\\ &\quad \wedge instMultsOk(SGT) \wedge instContainmentAcyclic(SGT)\end{aligned}$$

The set of all conformable typed SGs is defined from the predicate above as:

$$SGTyConf = \{SGT : SGTy \mid isConformable(SGT)\}$$

□

A.12 Typed Fragments

Definition 32 (Type Fragments)**.** A type fragment is a pair $TF = (F, iet)$ that comprises a fragment $F : Fr$ and a colouring function $iet : EsA_F \rightarrow SGET$ that indicates the instance-level edge types stipulated by the fragment's type-level association edges (relation or composition). The set of type fragments TFr, such that $TF : TFr$, is defined as:

$$TFr = \{(F, iet) \mid F \in Fr \wedge iet \in EsA_F \rightarrow SGET\}$$

Auxiliary Definitions. Functions Ns and Es of Fr are extended to TFr. Functions fr and $iety$ yield the components of a TFr:

$$
\begin{array}{ll}
fr : TFr \rightarrow Fr & iety : TFr \rightarrow E \nrightarrow SGET \\
fr(F, iet) = F & iety(F, iet) = iet \\
\multicolumn{2}{l}{\_ \cup_{TF} \_ : TFr \times TFr \rightarrow TFr} \\
\multicolumn{2}{l}{TF_1 \cup_{TF} TF_2 = (fr\ TF_1 \cup_F fr\ TF_2, iety\ TF_1 \cup iety\ TF_2)}
\end{array}
$$

□

Definition 33 (Typed Fragments)**.** A typed fragment is a triple $FT = (F, TF, type)$, consisting of an instance level fragment $F : Fr$, a type fragment $TF : TFr$ and fragment morphism $type : F \rightarrow TF$, mapping the instance fragment to the type one. The set of typed fragments $FrTy$, such that $FT \in FrTy$, is defined as:

$$FrTy = \{(F, TF, type) \mid F \in Fr \wedge TF \in TFr \wedge type \in F \rightarrow fr\ TF\}$$

□

Definition 34 (Fragment Conformance)**.** We introduce several predicates to check the conformance of a typed fragment. First predicate checks that edge types of instance fragment conform with edge types prescribed by type fragment:

$$instanceEdgeTypesOk(F, TF, type) \Leftrightarrow iety_{TF} \circ (fE\ type) = ety_F$$

Second predicate checks that abstract nodes do not have any direct instances:

$$abstractNoDirectInstances(F, TF, type) \Leftrightarrow ((f_V\ type)^{\sim} (\!| NsAbst\ TF |\!)) = \varnothing$$

This says that the set of instances of abstract nodes (obtained from the inverse of type morphism) must be empty. The function $NsAbst$ is defined to take proxy nodes into account:

$$NsAbst\ F = \bigcup \{va : NsTy(F, \{nabst\}) \bullet reps(F, va)\}$$

Above, we get all representatives of some abstract node (function $reps$).

Third predicate checks that instances of type containment edges do not allow contained nodes to be shared:

$$
\begin{array}{l}
containmentNoSharing(F, TF, type) \Leftrightarrow \\
\quad ((f_E\ type)^{\sim} (\!| EsTy(TF, \{ecomp\}) |\!)) \lhd tgt^*\ F \in E\ injrel\ V
\end{array}
$$

This requires that the target function of instances of containment edges is injective (set definition $injrel$ of def.1).

Fourth predicate checks that the multiplicity constraints prescribed by the type are satisfied in the instances:

$$
\begin{array}{l}
instMultsOk(F, TF, type) \Leftrightarrow \forall\, te : EsA\ TF \bullet \\
\quad \exists\, r : V \leftrightarrow V \bullet r = rel(restrict(gr\ sg\ F, (f_E\ type)^{\sim} (\!| \{te\} |\!))) \\
\qquad \wedge\ \forall\, v : \mathrm{dom}\, r \bullet multOk(r\ (\!| repsOf(v, F) |\!), (srcm\ (sg\ TF) te) \\
\qquad \wedge\ \forall\, v : \mathrm{ran}\, r \bullet multOk(r^{\sim} (\!| repsOf(v, F) |\!), (tgtm\ (sg\ TF) te)
\end{array}
$$

This predicate obtains the relation that is induced by the edges that are instances of the association edges of the graph (function *rel*), and then goes through this relation checking each element in domain and range. This definition takes proxy nodes into account (function *reps*).

Fifth predicate checks that instances of containment relations form a forest:

$$\begin{aligned}&instContainmentAcyclic(F, TF, type) \Leftrightarrow\\&\quad rel(restrict(gr\ SG, (f_E\ type)^{\sim} ⦇EsTy(TF, \{ecomp\})⦈)) \in forest\end{aligned}$$

A summary predicate checks that typed fragments are conformant:

$$\begin{aligned}&isConformable\ FT \Leftrightarrow instanceEdgeTypesOk\ FT \wedge abstractNoDirectInstances\ FT\\&\quad \wedge containmentNoSharing\ FT \wedge instMultsOk\ FT \wedge instContainmentAcyclic\ FT\end{aligned}$$

This way we define the set of conformant typed fragments as:

$$FrTyConf = \{FT : FrTy \mid isConformable\ FT\}$$

□

A.13 Typed Models

Definition 35 (Type Models)**.** A type model with a FS is a tuple $TM = (GFG, CG, mc, fd, SGFG, SCG, sc, sf)$, consisting of a model part and a FS part; the model part comprises a $GFG : GFGr$, a $CG : CGr$, a morphism $mc : GFG \rightarrow CG$, and a function mapping fragment nodes to typed fragments $fd : Ns_{GFG} \rightarrow TFr$; the FS part comprises a $SGFG : GFGr$, a $SCG : CGr$, and two morphism $sc : SGFG \rightarrow SCG$ and $sf : UTFs\ TM \rightarrow SGFG$.

The set of base type models $TMdl$, such that $TM \in TMdl_0$, is defined as:

$$\begin{aligned}&TMdl_0 = \{(GFG, CG, mc, fd, SGFG, SCG, sc, sf) \mid GFG \in GFGr \wedge CG \in CGr\\&\quad \wedge mc \in GFG \rightarrow CG \wedge fd \in Ns_{GFG} \rightarrow TFr \wedge SGFG \in GFGr \wedge SCG \in CGr\\&\quad \wedge sc \in SGFG \rightarrow SCG \wedge sf \in GrMorph\}\end{aligned}$$

We extend the functions to extract the different components of a model (set Mdl, def. 23) to typed models ($TMdl_0$). We define further functions to yield components of $TMdl_0$:

$$\begin{aligned}&sgfg : TMdl_0 \rightarrow GFGr\\&sgfg(GFG, CG, mc, fd, SGFG, SCG, sc, sf) = GFG\\&scg : TMdl_0 \rightarrow CGr\\&scg(GFG, CG, mc, fd, SGFG, SCG, sc, sf) = SCG\\&smcg : TMdl_0 \rightarrow GrMorph\\&smcg(GFG, CG, mc, fd, SGFG, SCG, sc, sf) = sc\\&smfg : TMdl_0 \rightarrow (V \nrightarrow Fr)\\&smfg(GFG, CG, mc, fd, SGFG, SCG, sc, sf) = sf\end{aligned}$$

Function *UTFs* returns the fragment that results from the union of all fragments of a model. *mUTMFsToGFG* builds a morphism from the union of all typed fragments of a model to the given model's FG.

Function *UFs* returns the fragment that results from the union of all fragments of a model. $from_V$ indicates to which fragment a local node belongs to:

$$\begin{aligned}&UTFs : TMdl_0 \rightarrow TFr && UTFs_0 : \mathbb{P}_1\ TFr \rightarrow TFr\\&UTFs\ M = UFs_0(\mathrm{ran}(fdef\ M)) && UTFs_0\{TF\} = TF\\& && UTFs_0\{TF\} \cup TFs = TF \cup_{TF} (UTFs_0\ TFs)\end{aligned}$$

$$\begin{aligned}&from_{VT} : V_L \times TMdl \nrightarrow V_F\\&from_{VT}(vl, TM) = vf \Leftrightarrow vl \in Ns(fdef\ vf)\end{aligned}$$

Function $mUTMFsToGFG$ builds a morphism from the union of all type fragments of a type model to the given model's GFG, which involves other auxiliary functions (such as $consTFToGFG$):

$$
\begin{aligned}
&consTFToGFG : V_F \times TMdl_0 \to GrMorph \\
&consTFToGFG(vf, TM) = (fv, fe) \Leftrightarrow \exists\ TF : TFr;\ GFG : GFGr \bullet \\
&\quad TF = fdef\ TM\ vf \wedge GFG = fg\ TM \wedge fv \in Ns_{TF} \to Ns_{GFG} \\
&\quad \wedge fe \in Es_{TF} \to Es_{GFG} \wedge vf \in Ns_{GFG} \\
&\quad \wedge (\exists\ ef : Es_{GFG} \bullet src_{GFG}\ ef = tgt_{GFG}\ ef = vf \wedge fv = Ns_{TF} \times \{vf\} \\
&\qquad \wedge fe = (Es_{TF} \setminus EsR_{TF}) \times \{ef\} \cup consTFToGFGRefs(vf, EsR_F, TM)) \\
&consTFToGFGRefs : V_F \times \mathbb{P}\, E_L \times TMdl_0 \to E \twoheadrightarrow E \\
&consTFToGFGRefs(vf, \{\}, TM) = \{\} \\
&consTFToGFGRefs(vf, \{el\} \cup E_r, TM) = fe \Leftrightarrow \\
&\quad \exists\ TF : TFr;\ GFG : GFGr \bullet TF = fdef\ TM\ vf \wedge GFG = fg\ TM \\
&\quad \wedge (\exists\ ef : Es_{FG} \bullet src_{GFG} ef = vf \wedge tgt_{GFG}\ ef = from_{VT}(tgtr_F\ el, TM) \\
&\quad \wedge fe = \{e_L \mapsto e_f\} \cup consTFToGFGRefs(vf, E_r, TM)) \\
&mUTMFsToGFG : TMdl_0 \to GrMorph \\
&mUTMFsToGFG\ M = buildUTFsToGFG(fdef\ TM, TM) \\
&buildUTFsToGFG : (V \twoheadrightarrow Fr) \times TM \to GrMorph \\
&buildUTFsToGFG(\{vf \mapsto F\}, TM) = consTFToGFG(vf, TM) \\
&buildUTFsToGFG(\{vf \mapsto F\} \cup fd, TM) = consTFToGFG(vf, TM) \\
&\quad \cup_{GM} buildUTFsToGFG(fd, TM)
\end{aligned}
$$

The set of all type models $TMdl$, such that $TM \in TMdl$, is defined as:

$$
\begin{aligned}
&TMdl = \{TM : TMdl_0 \mid smfg\ TM \in (UTFs\ TM) \to sgfg\ TM \\
&\quad \wedge mUTMFsToGFG\ TM \in UTFs\ TM \to (fg\ TM) \\
&\quad \wedge (\forall\ vf_1, vf_2 : Ns(fg\ TM) \mid vf_1 \neq vf_2 \bullet \\
&\qquad Ns(fdef\ TM\ vf_1) \cap Ns(fdef\ TM\ vf_2) = \varnothing \wedge Es(fdef\ TM\ vf_1) \cap Es(fdef\ TM\ vf_2) = \varnothing)\}
\end{aligned}
$$

□

Definition 36 (Fragmentation Strategies)**.** A fragmentation strategy (FS) is a quadruple $FS = (CG, GFG, sc, sf)$, consisting of two graphs corresponding to the FS's $CG : CGr$ and $GFG : GFGr$, and two morphisms sc, sf, mapping GFG to CG and elements of the model's fragments into the GFG. Set FSs is defined as:

$$
\begin{aligned}
&FSs = \{(CG, GFG, sc, sf) \mid CG \in CGr \wedge GFG \in GFGr \\
&\quad \wedge sc \in GFG \to CG \wedge sf \in GrMorph\}
\end{aligned}
$$

□

Definition 37 (Type model with FS)**.** A type model with a FS is a pair $TFSM = (TM, FS)$, consisting of a type model $TM : TMdl$ (a model containing type fragments, TFr) and a $FS : FSs$. Set of all such models is defined as:

$$TFSMdl = \{(TM, FS) \mid TM \in TMdl \wedge FS \in FSs \wedge mgfg_{FS} \in UTFs\ TM \to gfg_{FS}\}$$

This says that the FS's morphism from fragment elements to the FS's GFG (function $mgfg$) maps elements from union of all the model's fragments. □

Definition 38. A typed model with a FS (Fig. 6.3(c)) is a tuple $MT = (M, TM, scg, sgfg, ty)$, consisting of a model $M : Mdl$, a type model $TM : TFSMdl$ and morphisms: (i) $smc : cg_M \to scg_{TM}$ maps M's CG into the FS's CG of TM, (ii) $smf : gfg_M \to sgfg_{TM}$ maps GFG of M into the FS's GFG of TM, and (iii) $ty : UFs\,M \to UFs\,TM$ maps union of model fragments of M into its TM counter-part. Set of typed models is defined as:

$$
\begin{aligned}
&MdlTy = \{(M, TM, scg, sgfg, ty) \mid M \in Mdl \land TM \in TFSMdl\\
&\quad \land scg \in cg_M \to scg_{TM} \land sgfg \in gfg_M \to sgfg_{TM}\\
&\quad \land (UFs\,M, UTFs\,TM, ty) \in FrTyConf\\
&\quad \land sgfg \circ UMToGFG\,M = msf_{TM} \circ ty \land scg \circ mcg_M = msc_{TM} \circ sgfg\}
\end{aligned}
$$

Here, first four conjuncts state usual membership constraints. Then, we state that the union of M's fragments must conform to its TM counter-part (set $FrTyConf$ of def. 34), and required commutativity constraints as per Fig. 6.4(b). □

Appendix B

Z Specification of Fragmenta

B.1 Generics

section *Fragmenta_Generics* **parents** *standard_toolkit*

$acyclic[X] == \{r : X \leftrightarrow X \mid r^{+} \cap \mathrm{id}\, X = \varnothing\}$

$connected[X] == \{r : X \leftrightarrow X \mid \forall x : \mathrm{dom}\, r;\ y : \mathrm{ran}\, r \bullet x \mapsto y \in r^{+}\}$

$tree[X] == \{r : X \leftrightarrow X \mid r \in acyclic \land r \in X \nrightarrow X\}$

$forest[X] == \{r : X \leftrightarrow X \mid r \in acyclic \land (\forall s : X \leftrightarrow X \mid s \subseteq r \land s \in connected \bullet s \in tree)\}$

$injrel[X, Y] == \{r : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x, y_1) \in r \land (x, y_2) \in r \Rightarrow y_1 = y_2)\}$

B.2 Graphs

section *Fragmenta_Graphs* **parents** *standard_toolkit*, *Fragmenta_Generics*

$[V, E]$

$Gr == \{vs : \mathbb{P}\, V;\ es : \mathbb{P}\, E;\ s, t : E \nrightarrow V \mid s \in es \rightarrow vs \land t \in es \rightarrow vs\}$

$Ns : Gr \rightarrow \mathbb{P}\, V$
$Es, EsId : Gr \rightarrow \mathbb{P}\, E$
$src, tgt : Gr \rightarrow E \nrightarrow V$

$\forall vs : \mathbb{P}\, V;\ es : \mathbb{P}\, E;\ s : E \nrightarrow V;\ t : E \nrightarrow V \bullet Ns(vs, es, s, t) = vs$

$\forall vs : \mathbb{P}\, V;\ es : \mathbb{P}\, E;\ s : E \nrightarrow V;\ t : E \nrightarrow V \bullet Es(vs, es, s, t) = es$

$\forall vs : \mathbb{P}\, V;\ es : \mathbb{P}\, E;\ s : E \nrightarrow V;\ t : E \nrightarrow V \bullet src(vs, es, s, t) = s$

$\forall vs : \mathbb{P}\, V;\ es : \mathbb{P}\, E;\ s : E \nrightarrow V;\ t : E \nrightarrow V \bullet tgt(vs, es, s, t) = t$

$\forall vs : \mathbb{P}\, V;\ es : \mathbb{P}\, E;\ s : E \nrightarrow V;\ t : E \nrightarrow V \bullet EsId(vs, es, s, t) = \{e : es \mid s\, e = t\, e\}$

relation$(adjacent\ \_)$

$$\begin{array}{|l} adjacent\_ : \mathbb{P}(V \times V \times Gr) \\ \hline \forall v_1, v_2 : V;\ G : Gr \bullet (adjacent(v_1, v_2, G)) \Leftrightarrow (\exists e : Es\ G \bullet src\ G\ e = v_1 \land tgt\ G\ e = v_2) \end{array}$$

$$\begin{array}{|l} successors : V \times Gr \rightarrow \mathbb{P}\ V \\ \hline \forall v : V;\ G : Gr \bullet successors(v, G) = \{v_1 : Ns\ G \mid adjacent(v, v_1, G)\} \end{array}$$

$$\begin{array}{|l} rel : Gr \rightarrow V \leftrightarrow V \\ \hline \forall G : Gr \bullet rel\ G = \{v_1, v_2 : Ns\ G \mid adjacent(v_1, v_2, G)\} \end{array}$$

relation$(acyclicG\ \_)$

$$\begin{array}{|l} acyclicG\_ : \mathbb{P}\ Gr \\ \hline \forall G : Gr \bullet (acyclicG\ G) \Leftrightarrow rel\ G \in acyclic \end{array}$$

$$\begin{array}{|l} restrict : Gr \times \mathbb{P}\ E \rightarrow Gr \\ \hline \forall G : Gr;\ Er : \mathbb{P}\ E \bullet restrict(G, Er) = (Ns\ G, Es\ G \cap Er, Er \lhd src\ G, Er \lhd tgt\ G) \end{array}$$

relation$(disjGs\ \_)$

$$\begin{array}{|l} disjGs\_ : \mathbb{P}(Gr \times Gr) \\ \hline \forall G_1, G_2 : Gr \bullet (disjGs(G_1, G_2)) \Leftrightarrow Ns\ G_1 \cap Ns\ G_2 = \varnothing \land Es\ G_1 \cap Es\ G_2 = \varnothing \end{array}$$

function 10 **leftassoc** $(\_ \cup_G \_)$

$$\begin{array}{|l} \_ \cup_G \_ : Gr \times Gr \rightarrow\!\!\!\!\!+ Gr \\ \hline \forall G_1, G_2 : Gr \bullet G_1 \cup_G G_2 = (Ns\ G_1 \cup Ns\ G_2, Es\ G_1 \cup Es\ G_2, src\ G_1 \cup src\ G_2, tgt\ G_1 \cup tgt\ G_2) \\ \quad \Leftrightarrow (disjGs(G_1, G_2)) \end{array}$$

$$
\begin{array}{|l}
replaceGfun : (E \rightarrow\!\!\!\!\!\!+ V) \rightarrow (V \rightarrow\!\!\!\!\!\!+ V) \rightarrow (E \rightarrow\!\!\!\!\!\!+ V) \\
\hline
\forall f : E \rightarrow\!\!\!\!\!\!+ V;\ sub : V \rightarrow\!\!\!\!\!\!+ V \bullet \\
\quad replaceGfun\, f\, sub = f \oplus \{e : \mathrm{dom}\, f;\ v : V \mid (f\ e) \in \mathrm{dom}\, sub \wedge sub\,(f\ e) = v\}
\end{array}
$$

$$
\begin{array}{|l}
replaceG : Gr \rightarrow (V \rightarrow\!\!\!\!\!\!+ V) \rightarrow Gr \\
\hline
\forall G : Gr;\ sub : V \rightarrow\!\!\!\!\!\!+ V \bullet replaceG\ G\ sub = (Ns\ G \setminus \mathrm{dom}\, sub \cup \mathrm{ran}(Ns\ G \lhd sub), Es\ G, \\
\quad replaceGfun\,(src\ G)\, sub, replaceGfun\,(tgt\ G)\, sub)
\end{array}
$$

$$
GrMorph == (V \rightarrow\!\!\!\!\!\!+ V) \times (E \rightarrow\!\!\!\!\!\!+ E)
$$

$$
\begin{array}{|l}
fV : GrMorph \rightarrow V \rightarrow\!\!\!\!\!\!+ V \\
fE : GrMorph \rightarrow E \rightarrow\!\!\!\!\!\!+ E \\
\hline
\forall fv : V \rightarrow\!\!\!\!\!\!+ V;\ fe : E \rightarrow\!\!\!\!\!\!+ E \bullet fV(fv, fe) = fv \\
\forall fv : V \rightarrow\!\!\!\!\!\!+ V;\ fe : E \rightarrow\!\!\!\!\!\!+ E \bullet fE(fv, fe) = fe
\end{array}
$$

function 10 **leftassoc** $(\_ \cup_{GM} \_)$

$$
\begin{array}{|l}
\_ \cup_{GM} \_ : GrMorph \times GrMorph \rightarrow\!\!\!\!\!\!+ GrMorph \\
\hline
\forall GM_1, GM_2 : GrMorph \bullet \\
\quad GM_1 \cup_{GM} GM_2 = (fV\ GM_1 \cup fV\ GM_2, fE\ GM_1 \cup fE\ GM_2) \Leftrightarrow \\
\quad\quad fV\ GM_1 \cap fV\ GM_2 = \varnothing \wedge fE\ GM_1 \cap fE\ GM_2 = \varnothing
\end{array}
$$

$$
\begin{array}{|l}
morphG : Gr \times Gr \rightarrow \mathbb{P}\, GrMorph \\
\hline
\forall G_1, G_2 : Gr \bullet morphG(G_1, G_2) = \{fv : Ns\ G_1 \rightarrow Ns\ G_2;\ fe : Es\ G_1 \rightarrow Es\ G_2 \mid \\
\quad src\ G_2 \circ fe = fv \circ src\ G_1 \wedge tgt\ G_2 \circ fe = fv \circ tgt\ G_1\}
\end{array}
$$

function 10 **leftassoc** $(\_ \circ_G \_)$

$$
\begin{array}{|l}
\_ \circ_G \_ : GrMorph \times GrMorph \rightarrow GrMorph \\
\hline
\forall m_1, m_2 : GrMorph \bullet m_1 \circ_G m_2 = (fV\ m_1 \circ fV\ m_2, fE\ m_1 \circ fE\ m_2)
\end{array}
$$

B.3 Category Theory

section *Fragmenta_CatTheory* **parents** *standard_toolkit*, *Fragmenta_Graphs*

$[O, M]$

$$Cat0 == \{os : \mathbb{P}\, O;\ ms : \mathbb{P}\, M;\ dm, cd : M \to O;\ idn : O \rightarrowtail\!\!\!\to M;\ cmp : M \times M \rightarrowtail\!\!\!\to M \mid \\ dm \in ms \to os \land cd \in ms \to os \land idn \in os \to ms \land cmp \in ms \times ms \to ms\}$$

$$\begin{array}{|l}
obs : Cat0 \to \mathbb{P}\, O \\
morphs : Cat0 \to \mathbb{P}\, M \\
domC, codC : Cat0 \to M \rightarrowtail\!\!\!\to O \\
idC : Cat0 \rightarrowtail\!\!\!\to O \rightarrowtail\!\!\!\to M \\
comp : Cat0 \rightarrowtail\!\!\!\to M \times M \rightarrowtail\!\!\!\to M \\
\hline
\forall os : \mathbb{P}\, O;\ ms : \mathbb{P}\, M;\ dm, cd : M \to O;\ idn : O \rightarrowtail\!\!\!\to M;\ cmp : M \times M \rightarrowtail\!\!\!\to M \bullet \\
\quad obs(os, ms, dm, cd, idn, cmp) = os \\
\forall os : \mathbb{P}\, O;\ ms : \mathbb{P}\, M;\ dm, cd : M \to O;\ idn : O \rightarrowtail\!\!\!\to M;\ cmp : M \times M \rightarrowtail\!\!\!\to M \bullet \\
\quad morphs(os, ms, dm, cd, idn, cmp) = ms \\
\forall os : \mathbb{P}\, O;\ ms : \mathbb{P}\, M;\ dm, cd : M \to O;\ idn : O \rightarrowtail\!\!\!\to M;\ cmp : M \times M \rightarrowtail\!\!\!\to M \bullet \\
\quad domC(os, ms, dm, cd, idn, cmp) = dm \\
\forall os : \mathbb{P}\, O;\ ms : \mathbb{P}\, M;\ dm, cd : M \to O;\ idn : O \rightarrowtail\!\!\!\to M;\ cmp : M \times M \rightarrowtail\!\!\!\to M \bullet \\
\quad codC(os, ms, dm, cd, idn, cmp) = cd \\
\forall os : \mathbb{P}\, O;\ ms : \mathbb{P}\, M;\ dm, cd : M \to O;\ idn : O \rightarrowtail\!\!\!\to M;\ cmp : M \times M \rightarrowtail\!\!\!\to M \bullet \\
\quad idC(os, ms, dm, cd, idn, cmp) = idn \\
\forall os : \mathbb{P}\, O;\ ms : \mathbb{P}\, M;\ dm, cd : M \to O;\ idn : O \rightarrowtail\!\!\!\to M;\ cmp : M \times M \rightarrowtail\!\!\!\to M \bullet \\
\quad comp(os, ms, dm, cd, idn, cmp) = cmp
\end{array}$$

$$\begin{array}{|l}
CatMorphs : Cat0 \to (O \times O) \to \mathbb{P}\, M \\
\hline
\forall C : Cat0;\ A, B : O \bullet \\
\quad CatMorphs\ C(A, B) = \{m : morphs\ C \mid domC\ C\ m = A \land codC\ C\ m = B\}
\end{array}$$

$$\begin{array}{l}
Cat == \{C : Cat0 \mid (\forall A : obs\ C \bullet idC\ C\ A \in CatMorphs\ C(A, A)) \\
\quad \land (\forall f, g : morphs\ C \mid domC\ C\ g = codC\ C\ f \bullet \\
\qquad comp\ C(g, f) \in CatMorphs\ C((domC\ C\ f), (codC\ C\ g))) \\
\quad \land (\forall A, B, C_1, D : obs\ C \bullet \\
\qquad (\forall f : CatMorphs\ C(A, B);\ g : CatMorphs\ C(B, C_1);\ h : CatMorphs\ C(C_1, D) \bullet \\
\qquad\quad comp\ C(h, (comp\ C(g, f))) = comp\ C((comp\ C(h, g)), f))) \\
\quad \land (\forall A, B : obs\ C \bullet (\forall f : CatMorphs\ C(A, B) \bullet \\
\qquad (comp\ C((idC\ C\ B), f) = f \land comp\ C(f, (idC\ C\ A)) = f)))\}
\end{array}$$

$$MorphG2C == (V \rightarrowtail\!\!\!\to O) \times (E \rightarrowtail\!\!\!\to M)$$

$$\begin{array}{|l} mV : MorphG2C \rightarrow V \rightarrow O \\ mE : MorphG2C \rightarrow E \rightarrow M \\ \hline \forall\, mv : V \rightarrow O;\ me : E \rightarrow M \bullet mV(mv, me) = mv \\ \forall\, mv : V \rightarrow O;\ me : E \rightarrow M \bullet mE(mv, me) = me \end{array}$$

$$morphGC == (\lambda\, G : Gr;\ C : Cat \bullet \{mv : Ns\ G \rightarrow obs\ C;\ me : Es\ G \rightarrow morphs\ C \mid \\ mv \circ src\ G = domC\ C \circ me \wedge mv \circ tgt\ G = codC\ C \circ me\})$$

$$PPO == (\lambda\, C : Cat \bullet (\lambda f, g : morphs\ C \mid domC\ C\, f = domC\ C\, g \bullet \\ \{D : obs\ C;\ f', g' : morphs\ C \mid f' \in CatMorphs\ C((codC\ C\ g), D) \wedge \\ g' \in CatMorphs\ C((codC\ C\, f), D) \wedge comp\ C(f', g) = comp\ C(g', f)\}))$$

$$PO == (\lambda\, C : Cat \bullet (\lambda f, g : morphs\ C \bullet \\ (\mu\, D : obs\ C;\ f', g' : morphs\ C \mid (D, f', g') \in PPO\ C(f, g) \\ \wedge (\forall\, X : obs\ C;\ h, k : morphs\ C \bullet ((X, h, k) \in PPO\ C(f, g) \\ \wedge (\exists\, x : CatMorphs\ C(D, X) \bullet (comp\ C(x, f') = k \wedge comp\ C(x, g') = h)))))))$$

$$Diag == \{C : Cat;\ G : Gr;\ m : MorphG2C \mid m \in morphGC(G, C)\}$$

$$\begin{array}{|l} grD : Diag \rightarrow Gr \\ cat : Diag \rightarrow Cat \\ morphD : Diag \rightarrow MorphG2C \\ NsD : Diag \rightarrow \mathbb{P}\, V \\ obsD : Diag \rightarrow \mathbb{P}\, O \\ morphsD : Diag \rightarrow \mathbb{P}\, M \\ \hline \forall\, C : Cat;\ G : Gr;\ m : MorphG2C \bullet grD(C, G, m) = G \\ \forall\, C : Cat;\ G : Gr;\ m : MorphG2C \bullet cat(C, G, m) = C \\ \forall\, C : Cat;\ G : Gr;\ m : MorphG2C \bullet morphD(C, G, m) = m \\ \forall\, D : Diag \bullet NsD\ D = Ns(grD\ D) \\ \forall\, C : Cat;\ G : Gr;\ m : MorphG2C \bullet obsD(C, G, m) = \mathrm{ran}(mV\ m) \\ \forall\, C : Cat;\ G : Gr;\ m : MorphG2C \bullet morphsD(C, G, m) = \mathrm{ran}(mE\ m) \end{array}$$

$$CC == (\lambda\, D : Diag \bullet \{X : obs(cat\ D);\ ms : \mathbb{P}(morphs(cat\ D)) \mid \\ \forall\, m : ms \bullet domC(cat\ D)m \in obsD\ D \wedge codC(cat\ D)m = X\})$$

$$\begin{array}{|l} ValCC : Diag \rightarrow \mathbb{P}(O \times \mathbb{P}\, M) \\ \hline \forall\, D : Diag;\ X : O;\ ms : \mathbb{P}\, M \bullet (X, ms) \in ValCC\ D \Leftrightarrow (X, ms) \in CC\ D \\ \quad \wedge (\forall\, m : morphsD\ D \bullet (\exists f, g : ms \bullet (domC(cat\ D)m = domC(cat\ D)f \\ \qquad \wedge codC(cat\ D)m = domC(cat\ D)g \wedge comp(cat\ D)(g, m) = f))) \end{array}$$

$$Colimit == (\lambda D : Diag \bullet (\mu X : O;\ ms : \mathbb{P}\, M \mid (X, ms) \in ValCC\ D$$
$$\wedge (\forall X' : obs(cat\ D);\ ms' : \mathbb{P}\, M \mid X \neq X' \wedge (X', ms') \in ValCC\ D \bullet$$
$$(\exists k : CatMorphs(cat\ D)(X, X') \bullet (\forall f : ms;\ g : ms' \mid domC(cat\ D)f = domC(cat\ D)g \bullet$$
$$comp(cat\ D)(k, f) = g)))))$$

$$obCC : O \times \mathbb{P}\, M \rightarrow O$$
$$morphsCC : O \times \mathbb{P}\, M \rightarrow \mathbb{P}\, M$$

$$\forall X : O;\ ms : \mathbb{P}\, M \bullet obCC(X, ms) = X$$
$$\forall X : O;\ ms : \mathbb{P}\, M \bullet morphsCC(X, ms) = ms$$

B.4 The Graphs Category

section *Fragmenta_GraphsCat* **parents** *standard_toolkit*, *Fragmenta_Graphs*, *Fragmenta_CatTheory*

$$OGr : \mathbb{P}\, O$$
$$MGr : \mathbb{P}\, M$$
$$OGrToGr : O \rightarrowtail Gr$$
$$MGrToGrM : M \rightarrowtail GrMorph$$

$$idGr : OGr \rightarrow MGr$$
$$domGr, codGr : MGr \rightarrow OGr$$

$$\forall oG : OGr;\ mG : MGr \bullet idGr\ oG = mG \Leftrightarrow (\exists G : Gr;\ GM : GrMorph \bullet$$
$$G = OGrToGr\ oG \wedge MGrToGrM\ mG = GM \wedge GM = (\mathrm{id}(Ns\ G), \mathrm{id}(Es\ G)))$$

$$\forall mG : MGr;\ oG1 : OGr \bullet domGr\ mG = oG1 \Leftrightarrow (\exists GM : GrMorph;\ G_1, G_2 : Gr \bullet$$
$$GM = MGrToGrM\ mG \wedge G_1 = OGrToGr\ oG1 \wedge GM \in morphG(G_1, G_2))$$

$$\forall mG : MGr;\ oG2 : OGr \bullet$$
$$codGr\ mG = oG2 \Leftrightarrow (\exists GM : GrMorph;\ G_1, G_2 : Gr \bullet$$
$$GM = MGrToGrM\ mG \wedge G_2 = OGrToGr\ oG2 \wedge GM \in morphG(G_1, G_2))$$

$$cmpGr : MGr \times MGr \rightarrow MGr$$

$$\forall mG_1, mG_2, mG_3 : MGr \bullet cmpGr(mG_1, mG_2) = mG_3 \Leftrightarrow$$
$$(\exists GM_1, GM_2, GM_3 : GrMorph \bullet GM_1 = MGrToGrM\ mG_1 \wedge GM_2 = MGrToGrM\ mG_2$$
$$\wedge GM_3 = MGrToGrM\ mG_3 \wedge GM_3 = GM_1 \circ_G GM_2)$$

$$GraphsC : Cat$$

$$GraphsC = (OGr, MGr, domGr, codGr, idGr, cmpGr)$$

B.5 Structural Graphs

section *Fragmenta_SGs* **parents** *standard_toolkit*, *Fragmenta_Generics*, *Fragmenta_Graphs*

$SGNT ::= nnrml \mid nabst \mid nprxy$

$SGET ::= einh \mid ecomp \mid erel \mid elnk \mid eref$

$MultUVal ::= val \langle\!\langle \mathbb{N} \rangle\!\rangle \mid many$

$MultVal ::= mr \langle\!\langle \mathbb{N} \times MultUVal \rangle\!\rangle \mid ms \langle\!\langle MultUVal \rangle\!\rangle$

$$
\begin{array}{|l}
Mult : \mathbb{P}\, MultVal \\
\hline
Mult = \{ mv : MultVal \mid (\exists\, lb : \mathbb{N};\ ub : MultUVal \bullet mv = mr(lb, ub) \land ub = many \\
\quad \lor (\exists\, ubn : \mathbb{N} \bullet ub = val\, ubn \land lb \leqslant ubn)) \lor (\exists\, umv : MultUVal \bullet mv = ms\, umv) \}
\end{array}
$$

relation$(multOk\ \_)$

$$
\begin{array}{|l}
multOk\_ : \mathbb{P}(\mathbb{P}\, V \times Mult) \\
\hline
\forall\, vs : \mathbb{P}\, V;\ lb : \mathbb{N};\ ub : MultUVal \bullet (multOk(vs, mr(lb, ub))) \Leftrightarrow \\
\quad \#\, vs \geqslant lb \land (ub = many \lor (\exists\, ubn : \mathbb{N} \bullet ub = val\, ubn \land \#\, vs \leqslant ubn)) \\
\forall\, vs : \mathbb{P}\, V;\ v : MultUVal \bullet (multOk(vs, ms\, v)) \Leftrightarrow v = many \\
\quad \lor (\exists\, bn : \mathbb{N} \bullet v = val\, bn \land \#\, vs = bn)
\end{array}
$$

$$
\begin{array}{l}
SGr_0 == \{ G : Gr;\ nt : V \nrightarrow SGNT;\ et : E \nrightarrow SGET;\ sm, tm : E \nrightarrow Mult \mid \\
\quad nt \in Ns\ G \rightarrow SGNT \land et \in Es\ G \rightarrow SGET \}
\end{array}
$$

$$
\begin{array}{|l}
gr : SGr_0 \rightarrow Gr \\
sgr\_Ns : SGr_0 \rightarrow \mathbb{P}\, V \\
sgr\_Es : SGr_0 \rightarrow \mathbb{P}\, E \\
sgr\_src : SGr_0 \rightarrow E \nrightarrow V \\
sgr\_tgt : SGr_0 \rightarrow E \nrightarrow V \\
nty : SGr_0 \rightarrow V \nrightarrow SGNT \\
ety : SGr_0 \rightarrow E \nrightarrow SGET \\
srcm : SGr_0 \rightarrow E \nrightarrow Mult \\
tgtm : SGr_0 \rightarrow E \nrightarrow Mult \\
\hline
\forall\, G : Gr;\ nt : V \nrightarrow SGNT;\ et : E \nrightarrow SGET;\ sm, tm : E \nrightarrow Mult \bullet gr(G, nt, et, sm, tm) = G \\
\forall\, SG : SGr_0 \bullet sgr\_Ns\ SG = Ns(gr\ SG) \\
\forall\, SG : SGr_0 \bullet sgr\_Es\ SG = Es(gr\ SG) \\
\forall\, SG : SGr_0 \bullet sgr\_src\ SG = src(gr\ SG) \\
\forall\, SG : SGr_0 \bullet sgr\_tgt\ SG = tgt(gr\ SG) \\
\forall\, G : Gr;\ nt : V \nrightarrow SGNT;\ et : E \nrightarrow SGET;\ sm, tm : E \nrightarrow Mult \bullet nty(G, nt, et, sm, tm) = nt \\
\forall\, G : Gr;\ nt : V \nrightarrow SGNT;\ et : E \nrightarrow SGET;\ sm, tm : E \nrightarrow Mult \bullet ety(G, nt, et, sm, tm) = et \\
\forall\, G : Gr;\ nt : V \nrightarrow SGNT;\ et : E \nrightarrow SGET;\ sm, tm : E \nrightarrow Mult \bullet srcm(G, nt, et, sm, tm) = sm \\
\forall\, G : Gr;\ nt : V \nrightarrow SGNT;\ et : E \nrightarrow SGET;\ sm, tm : E \nrightarrow Mult \bullet tgtm(G, nt, et, sm, tm) = tm
\end{array}
$$

$$\begin{array}{|l}
NsTy : SGr_0 \times \mathbb{P}\, SGNT \rightarrow \mathbb{P}\, V \\
EsTy : SGr_0 \times \mathbb{P}\, SGET \rightarrow \mathbb{P}\, E \\
\hline
\forall\, SG : SGr_0;\ nts : \mathbb{P}\, SGNT \bullet NsTy(SG, nts) = (nty\, SG)^{\sim} \limg nts \rimg \\
\forall\, SG : SGr_0;\ ets : \mathbb{P}\, SGET \bullet EsTy(SG, ets) = (ety\, SG)^{\sim} \limg ets \rimg
\end{array}$$

$$\begin{array}{|l}
EsA : SGr_0 \rightarrow \mathbb{P}\, E \\
EsR : SGr_0 \rightarrow \mathbb{P}\, E \\
\hline
\forall\, SG : SGr_0 \bullet EsA\, SG = EsTy(SG, \{erel, ecomp, elnk\}) \\
\forall\, SG : SGr_0 \bullet EsR\, SG = EsTy(SG, \{eref\})
\end{array}$$

$$\begin{array}{|l}
NsP : SGr_0 \rightarrow \mathbb{P}\, V \\
\hline
\forall\, SG : SGr_0 \bullet NsP\, SG = NsTy(SG, \{nprxy\})
\end{array}$$

$$\begin{array}{|l}
inhG : SGr_0 \rightarrow Gr \\
inh : SGr_0 \rightarrow V \leftrightarrow V \\
\hline
\forall\, SG : SGr_0 \bullet inhG\, SG = restrict((gr\, SG), (EsTy(SG, \{einh\}) \setminus EsId(gr\, SG))) \\
\forall\, SG : SGr_0 \bullet inh\, SG = rel(inhG\, SG)
\end{array}$$

$$\begin{array}{l}
SGr == \{SG : SGr_0 \mid EsR\, SG \subseteq EsId(gr\, SG) \wedge srcm\, SG \in EsTy(SG, \{erel, ecomp\}) \rightarrow Mult \\
\quad \wedge\ tgtm\, SG \in EsTy(SG, \{erel, ecomp\}) \rightarrow Mult \\
\quad \wedge\ srcm\, SG \limg EsTy(SG, \{ecomp\}) \rimg \subseteq \{mr(0, val\, 1), ms\, (val\, 1)\} \\
\quad \wedge\ acyclicG\, (inhG\, SG)\}
\end{array}$$

$$\begin{array}{|l}
EsRP : SGr \rightarrow \mathbb{P}\, E \\
\hline
\forall\, SG : SGr \bullet EsRP\, SG = \{e : EsR\, SG \mid sgr_src\, SG\, e \in NsP\, SG\}
\end{array}$$

$$\begin{array}{|l}
inhst : SGr \rightarrow V \leftrightarrow V \\
clan : V \times SGr \rightarrow \mathbb{P}\, V \\
\hline
\forall\, SG : SGr \bullet inhst\, SG = (inh\, SG)^{*} \\
\forall\, v : V;\ SG : SGr \bullet clan(v, SG) = \{v' : sgr_Ns\, SG \mid v' \mapsto v \in inhst\, SG\}
\end{array}$$

$$\begin{array}{|l}
srcst : SGr \rightarrow E \leftrightarrow V \\
\hline
\forall\, SG : SGr \bullet srcst\, SG = \{e : EsA\, SG;\ v : sgr_Ns\, SG \mid \\
\quad \exists\, v_2 : sgr_Ns\, SG \bullet v \in clan(v_2, SG) \wedge sgr_src\, SG\, e = v_2\}
\end{array}$$

$$tgtst : SGr \to E \leftrightarrow V$$

$$\forall SG : SGr \bullet tgtst\ SG = \{e : EsA\ SG;\ v : sgr\_Ns\ SG \mid \\ \exists v_2 : sgr\_Ns\ SG \bullet v \in clan(v_2, SG) \land sgr\_tgt\ SG\ e = v_2\}$$

relation$(disjSGs\ \_)$

$$disjSGs\_ : \mathbb{P}(SGr \times SGr)$$

$$\forall SG_1, SG_2 : SGr \bullet (disjSGs(SG_1, SG_2)) \Leftrightarrow (disjGs(gr\ SG_1, gr\ SG_2))$$

function 10 **leftassoc** $(\_ \cup_{SG} \_)$

$$\_ \cup_{SG} \_ : SGr \times SGr \nrightarrow SGr$$

$$\forall SG_1, SG_2 : SGr \bullet SG_1 \cup_{SG} SG_2 = (gr\ SG_1 \cup_G gr\ SG_2, nty\ SG_1 \cup nty\ SG_2, \\ ety\ SG_1 \cup ety\ SG_2, srcm\ SG_1 \cup srcm\ SG_2, tgtm\ SG_1 \cup tgtm\ SG_2) \Leftrightarrow (disjSGs(SG_1, SG_2))$$

$$morphSG : SGr \times SGr \to \mathbb{P}\ GrMorph$$

$$\forall SG_1, SG_2 : SGr \bullet \\ morphSG(SG_1, SG_2) = \{fv : sgr\_Ns\ SG_1 \to sgr\_Ns\ SG_2;\ fe : sgr\_Es\ SG_1 \to sgr\_Es\ SG_2 \mid \\ fv \circ srcst\ SG_1 \subseteq srcst\ SG_2 \circ fe \land fv \circ tgtst\ SG_1 \subseteq tgtst\ SG_2 \circ fe \\ \land fv \circ inhst\ SG_1 \subseteq inhst\ SG_2 \circ fv\}$$

B.6 Fragments

section *Fragmenta_Frs* **parents** *standard_toolkit*, *Fragmenta_SGs*

$$Fr_0 == \{SG : SGr;\ tr : E \nrightarrow V \mid tr \in EsRP\ SG \to V \\ \land (EsRP\ SG) \lhd (sgr\_src\ SG) \in (EsRP\ SG) \rightarrowtail NsP\ SG \\ \land EsTy(SG, \{einh\}) \lhd sgr\_src\ SG \rhd NsP\ SG = \{\}\}$$

$$
\begin{array}{l}
fsrcGr : Fr_0 \rightarrow Gr \\
ftgtr : Fr_0 \rightarrow E \rightarrowtail V \\
fNs : Fr_0 \rightarrow \mathbb{P}\, V \\
fEs : Fr_0 \rightarrow \mathbb{P}\, E \\
fEsR : Fr_0 \rightarrow \mathbb{P}\, E \\
fsg : Fr_0 \rightarrow SGr \\
fsrc : Fr_0 \rightarrow E \rightarrowtail V \\
ftgt : Fr_0 \rightarrow E \rightarrowtail V \\
\hline
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet fsrcGr(SG, tr) = gr\, SG \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet ftgtr(SG, tr) = tr \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet fNs(SG, tr) = sgr\_Ns\, SG \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet fEs(SG, tr) = sgr\_Es\, SG \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet fEsR(SG, tr) = EsR\, SG \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet fsg(SG, tr) = SG \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet fsrc(SG, tr) = sgr\_src\, SG \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet ftgt(SG, tr) = sgr\_tgt\, SG
\end{array}
$$

$$
\begin{array}{l}
tgtr : Fr_0 \rightarrow E \rightarrowtail V \\
withRsG : Fr_0 \rightarrow Gr \\
refsG : Fr_0 \rightarrow Gr \\
refs : Fr_0 \rightarrow V \leftrightarrow V \\
reps : Fr_0 \rightarrow V \leftrightarrow V \\
referenced : Fr_0 \rightarrow \mathbb{P}\, V \\
\hline
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet tgtr(SG, tr) = sgr\_tgt\, SG \oplus tr \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet \\
\quad withRsG(SG, tr) = (sgr\_Ns\, SG \cup \operatorname{ran} tr, sgr\_Es\, SG, sgr\_src\, SG, tgtr(SG, tr)) \\
\forall F : Fr_0 \bullet refsG\, F = restrict((withRsG\, F), (EsRP(fsg\, F))) \\
\forall F : Fr_0 \bullet refs\, F = rel(refsG\, F) \\
\forall F : Fr_0 \bullet reps\, F = refs\, F \cup (refs\, F)^{\sim} \\
\forall SG : SGr;\ tr : E \rightarrowtail V \bullet referenced(SG, tr) = \operatorname{ran} tr
\end{array}
$$

$$
\begin{array}{l}
inhF : Fr_0 \rightarrow V \leftrightarrow V \\
\hline
\forall F : Fr_0 \bullet inhF\, F = inh(fsg\, F) \cup reps\, F
\end{array}
$$

$$
\begin{array}{l}
refsOf : Fr_0 \rightarrow V \rightarrow \mathbb{P}\, V \\
\hline
\forall F : Fr_0;\ v : V \bullet refsOf\, F\, v = (refs\, F)^{+} (\!| \{v\} |\!)
\end{array}
$$

$$
\begin{array}{l}
nonPRefsOf : Fr_0 \rightarrow V \rightarrow \mathbb{P}\, V \\
\hline
\forall F : Fr_0;\ v : V \bullet nonPRefsOf\, F\, v = \{v2 : V \mid v2 \in refsOf\, F\, v \wedge v2 \in NsP(fsg\, F)\}
\end{array}
$$

relation$(acyclicIF\ \_)$

$$\begin{array}{|l} acyclicIF\_ : \mathbb{P}\, Fr_0 \\ \hline \forall\, F : Fr_0 \bullet (acyclicIF\ F) \Leftrightarrow (inh(fsg\ F) \cup refs\ F) \in acyclic \end{array}$$

$$Fr == \{F : Fr_0 \mid (\forall\ v : NsP(fsg\ F) \bullet nonPRefsOf\ F\ v \neq \varnothing) \land acyclicIF\ F\}$$

$$\begin{array}{|l} repsOf : V \to Fr \to \mathbb{P}\ V \\ \hline \forall\, v : V;\ F : Fr \bullet repsOf\ v\ F = \{v' : fNs\ F \mid (v', v) \in (reps\ F)^*\} \end{array}$$

$$\begin{array}{|l} fr\_NsAbst : Fr \to \mathbb{P}\ V \\ \hline \forall\, F : Fr \bullet fr\_NsAbst\ F = \bigcup\{va : NsTy((fsg\ F), \{nabst\}) \bullet (repsOf\ va\ F)\} \end{array}$$

relation(disjFs _)

$$\begin{array}{|l} \mathrm{disjFs}\_ : \mathbb{P}(Fr \times Fr) \\ \hline \forall\, F_1, F_2 : Fr \bullet (\mathrm{disjFs}(F_1, F_2)) \Leftrightarrow (disjSGs(fsg\ F_1, fsg\ F_2)) \end{array}$$

function 10 **leftassoc** (_ $\cup_F$ _)

$$\begin{array}{|l} \_ \cup_F \_ : Fr \times Fr \rightarrow\!\!\!\!\!\!+\ Fr \\ \hline \forall\, F_1, F_2 : Fr \bullet F_1 \cup_F\ F_2 = (fsg\ F_1 \cup_{SG}\ fsg\ F_2, ftgtr\ F_1 \cup ftgtr\ F_2) \Leftrightarrow (\mathrm{disjFs}(F_1, F_2)) \end{array}$$

$$\begin{array}{|l} inhstF : Fr \to V \leftrightarrow V \\ \hline \forall\, F : Fr \bullet inhstF\ F = (inhF\ F)^* \end{array}$$

$$\begin{array}{|l} clanF : V \times Fr \to \mathbb{P}\ V \\ \hline \forall\, v : V;\ F : Fr \bullet clanF(v, F) = \{v' : fNs\ F \mid (v', v) \in inhstF\ F\} \end{array}$$

$$\begin{array}{|l} srcstF : Fr \to E \leftrightarrow V \\ \hline \forall\, F : Fr \bullet srcstF\ F = \{e : EsA(fsg\ F);\ v : fNs\ F \mid \exists\, v_2 : fNs\ F \bullet \\ \quad v \in clanF(v_2, F) \land (e, v_2) \in srcst(fsg\ F)\} \end{array}$$

$$
\begin{array}{|l}
tgtstF : Fr \to E \leftrightarrow V \\
\hline
\forall F : Fr \bullet tgtstF\ F = \{e : EsA(fsg\ F);\ v : fNs\ F \mid \exists v_2 : fNs\ F \bullet \\
\quad v \in clanF(v_2, F) \land (e, v_2) \in tgtst(fsg\ F)\}
\end{array}
$$

$$
\begin{array}{|l}
morphF : Fr \times Fr \to \mathbb{P}\ GrMorph \\
\hline
\forall F_1, F_2 : Fr \bullet morphF(F_1, F_2) = \{fv : fNs\ F_1 \to fNs\ F_2;\ fe : fEs\ F_1 \to fEs\ F_2 \mid \\
\quad fv \circ srcstF\ F_1 \subseteq srcstF\ F_2 \circ fe \land fv \circ tgtstF\ F_1 \subseteq tgtstF\ F_2 \circ fe \land \\
\quad fv \circ inhstF\ F_1 \subseteq inhstF\ F_2 \circ fv\}
\end{array}
$$

B.7 Global Fragment Graphs

section *Fragmenta_GFGs* **parents** *standard_toolkit*, *Fragmenta_Frs*

$FGCGEdgeTy ::= eimpo \mid econta \mid econti$

$ExtEdgeTy == \{eimpo, econti\}$

$$
\begin{array}{l}
GFGr == \{G : Gr;\ et : E \twoheadrightarrow ExtEdgeTy \mid et \in Es\ G \to ExtEdgeTy \\
\quad \forall v : Ns\ G \bullet \exists e : Es\ G \bullet src\ G\ e = v \land tgt\ G\ e = v \land acyclicG(restrict(G, (Es\ G \setminus EsId\ G)))\}
\end{array}
$$

$$
\begin{array}{|l}
gfgG : GFGr \to Gr \\
fety : GFGr \to E \twoheadrightarrow ExtEdgeTy \\
gfgNs : GFGr \to \mathbb{P}\ V \\
gfgEs : GFGr \to \mathbb{P}\ E \\
gfgEsOfTy : GFGr \times \mathbb{P}\ ExtEdgeTy \to \mathbb{P}\ E \\
importsOf : V \times GFGr \to \mathbb{P}\ V \\
continuationsOf : V \times GFGr \to \mathbb{P}\ V \\
continuesOf : V \times GFGr \to \mathbb{P}\ V \\
\hline
\forall G : Gr;\ et : E \to ExtEdgeTy \bullet gfgG(G, et) = G \\
\forall G : Gr;\ et : E \to ExtEdgeTy \bullet fety(G, et) = et \\
\forall G : Gr;\ et : E \to ExtEdgeTy \bullet gfgNs(G, et) = Ns\ G \\
\forall G : Gr;\ et : E \to ExtEdgeTy \bullet gfgEs(G, et) = Es\ G \\
\forall G : Gr;\ et : E \to ExtEdgeTy;\ fets : \mathbb{P}\ ExtEdgeTy \bullet gfgEsOfTy((G, et), fets) = et^{\sim} (\!| fets |\!) \\
\forall vf : V;\ GFG : GFGr \bullet \\
\quad importsOf(vf, GFG) = successors(vf, (restrict((gfgG\ GFG), (gfgEsOfTy(GFG, \{eimpo\})))) \\
\forall vf : V;\ GFG : GFGr \bullet \\
\quad continuationsOf(vf, GFG) = successors(vf, (restrict((gfgG\ GFG), (gfgEsOfTy(GFG, \{econti\})))) \\
\forall vf : V;\ GFG : GFGr \bullet \\
\quad continuesOf(vf, GFG) = \{vf_2 : V \mid \\
\qquad adjacent(vf_2, vf, restrict((gfgG\ GFG), (gfgEsOfTy(GFG, \{econti\}))))\}
\end{array}
$$

$$
\begin{array}{l}
morphGFG == (\lambda\ GFG_1, GFG_2 : GFGr \bullet \\
\quad \{fV : gfgNs\ GFG_1 \to gfgNs\ GFG_2;\ fE : gfgEs\ GFG_1 \to gfgEs\ GFG_2 \mid \\
\qquad (fV, fE) \in morphG((gfgG\ GFG_1), (gfgG\ GFG_2)) \land fety\ GFG_2 \circ fE = fety\ GFG_1\})
\end{array}
$$

$$
\begin{aligned}
&morphFGFG == (\lambda\, F : Fr;\ GFG : GFGr \bullet \\
&\quad \{fv : fNs\ F \rightarrow gfgNs\ GFG;\ fe : fEs\ F \rightarrow gfgEs\ GFG \mid \\
&\qquad (fv, fe) \in morphG((withRsG\ F), (gfgG\ GFG)) \\
&\qquad \land fNs\ F \neq \varnothing \Rightarrow (\exists\, vfg : gfgNs\ GFG \bullet fv \llparenthesis (fNs\ F) \rrparenthesis = \{vfg\}) \\
&\qquad \land fEs\ F \neq \varnothing \Rightarrow (\exists\, efg : gfgEs\ GFG \bullet \\
&\qquad\quad fe \llparenthesis fEs\ F \setminus EsR(fsg\ F) \rrparenthesis = \{efg\} \land efg \in EsId(gfgG\ GFG)) \\
&\qquad \land fEs\ F \neq \varnothing \land fNs\ F \neq \varnothing \Rightarrow \\
&\qquad\quad (\exists\, vfg : gfgNs\ GFG;\ efg : gfgEs\ GFG \bullet src(gfgG\ GFG) efg = vfg \\
&\qquad\quad \land fv \llparenthesis (ftgt\ F) \llparenthesis EsR(fsg\ F) \rrparenthesis \rrparenthesis = \{vfg\})\})
\end{aligned}
$$

B.8 Cluster Graphs

section *Fragmenta_CGs* **parents** *standard_toolkit*, *Fragmenta_GFGs*

$$
\begin{aligned}
&CGr == \{G : Gr;\ et : E \twoheadrightarrow FGCGEdgeTy \mid et \in Es\ G \rightarrow FGCGEdgeTy \\
&\quad \land (acyclicG\ restrict(G, ((et^{\sim} \llparenthesis \{eimpo, econti\} \rrparenthesis) \setminus EsId\ G))) \\
&\quad \land rel(restrict(G, ((et^{\sim} \llparenthesis \{econta\} \rrparenthesis) \setminus EsId\ G))) \in forest\}
\end{aligned}
$$

$$
\begin{aligned}
&cgG : CGr \rightarrow Gr \\
&cgNs : CGr \rightarrow \mathbb{P}\ V \\
&cgEs : CGr \rightarrow \mathbb{P}\ E \\
&cety : CGr \rightarrow E \twoheadrightarrow FGCGEdgeTy \\
&cgEsTy : CGr \times \mathbb{P}\ FGCGEdgeTy \rightarrow \mathbb{P}\ E \\
\hline
&\forall\, G : Gr;\ et : E \rightarrow FGCGEdgeTy \bullet cgG(G, et) = G \\
&\forall\, G : Gr;\ et : E \rightarrow FGCGEdgeTy \bullet cgNs(G, et) = Ns\ G \\
&\forall\, G : Gr;\ et : E \rightarrow FGCGEdgeTy \bullet cgEs(G, et) = Es\ G \\
&\forall\, G : Gr;\ et : E \rightarrow FGCGEdgeTy \bullet cety(G, et) = et \\
&\forall\, G : Gr;\ et : E \rightarrow FGCGEdgeTy;\ crts : \mathbb{P}\ FGCGEdgeTy \bullet cgEsTy((G, et), crts) = et^{\sim} \llparenthesis crts \rrparenthesis
\end{aligned}
$$

$$
\begin{aligned}
&morphCG == (\lambda\, CG_1, CG_2 : CGr \bullet \\
&\quad \{fV : cgNs\ CG_1 \rightarrow cgNs\ CG_2;\ fE : cgEs\ CG_1 \rightarrow cgEs\ CG_2 \mid \\
&\qquad (fV, fE) \in morphG((cgG\ CG_1), (cgG\ CG_2)) \land cety\ CG_2 \circ fE = cety\ CG_1\}) \\
&morphGFGCG == (\lambda\, GFG : GFGr;\ CG : CGr \bullet \\
&\quad \{fV : gfgNs\ GFG \rightarrow cgNs\ CG;\ fE : gfgEs\ GFG \rightarrow cgEs\ CG \mid \\
&\qquad (fV, fE) \in morphG((gfgG\ GFG), (cgG\ CG)) \land cety\ CG \circ fE = fety\ GFG\})
\end{aligned}
$$

B.9 Models

section *Fragmenta_Mdls* **parents** *standard_toolkit*, *Fragmenta_CGs*

$$
\begin{aligned}
&Mdl_0 == \{GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \twoheadrightarrow Fr \mid \\
&\quad fcl \in morphGFGCG(GFG, CG) \land fdef \in gfgNs\ GFG \rightarrow Fr\}
\end{aligned}
$$

$$
\begin{array}{|l}
mgfg : Mdl_0 \rightarrow GFGr \\
mcg : Mdl_0 \rightarrow CGr \\
mfcl : Mdl_0 \rightarrow GrMorph \\
mfdef : Mdl_0 \rightarrow V \pfun Fr \\
\hline
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \pfun Fr \bullet \\
\quad mgfg(GFG, CG, fcl, fdef) = GFG \\
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \pfun Fr \bullet \\
\quad mcg(GFG, CG, fcl, fdef) = CG \\
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \pfun Fr \bullet \\
\quad mfcl(GFG, CG, fcl, fdef) = fcl \\
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \pfun Fr \bullet \\
\quad mfdef(GFG, CG, fcl, fdef) = fdef
\end{array}
$$

$$
\begin{array}{|l}
UFs : Mdl_0 \rightarrow Fr \\
UFs_0 : \mathbb{P}_1\, Fr \rightarrow Fr \\
\hline
\forall M : Mdl_0 \bullet UFs\, M = UFs_0(\mathrm{ran}(mfdef\, M)) \\
\forall F : Fr \bullet UFs_0\{F\} = F \\
\forall F : Fr;\ Fs : \mathbb{P}_1\, Fr \bullet UFs_0(\{F\} \cup Fs) = F \cup_F (UFs_0\, Fs)
\end{array}
$$

$$
\begin{array}{|l}
fromV : V \times Mdl_0 \rightarrow V \\
\hline
\forall vl : V;\ M : Mdl_0;\ vf : V \bullet fromV(vl, M) = vf \Leftrightarrow vl \in fNs(mfdef\, M\, vf)
\end{array}
$$

$$
\begin{array}{|l}
consFToGFG : V \times Mdl_0 \rightarrow GrMorph \\
consFToGFGRefs : V \times \mathbb{P}\, E \times Mdl_0 \rightarrow E \pfun E \\
\hline
\forall vf : V;\ M : Mdl_0;\ fv : V \pfun V;\ fe : E \pfun E \bullet \\
\quad consFToGFG(vf, M) = (fv, fe) \Leftrightarrow \\
\qquad (\exists F : Fr;\ GFG : GFGr \bullet F = mfdef\, M\, vf \land GFG = mgfg\, M \land fv \in fNs\, F \rightarrow gfgNs\, GFG \\
\qquad \land fe \in fEs\, F \rightarrow gfgEs\, GFG \land vf \in gfgNs\, GFG \\
\qquad \land (\exists ef : gfgEs\, GFG \bullet (src(gfgG\, GFG)ef = tgt(gfgG\, GFG)ef = vf \land fv = fNs\, F \times \{vf\} \\
\qquad\quad \land fe = (fEs\, F \setminus fEsR\, F \times \{ef\}) \cup consFToGFGRefs(vf, (fEsR\, F), M)))) \\
\\
\forall vf : V;\ M : Mdl_0;\ fe : E \pfun E \bullet consFToGFGRefs(vf, \{\}, M) = \{\} \\
\\
\forall vf : V;\ M : Mdl_0;\ el : E;\ Er : \mathbb{P}\, E;\ fe : E \pfun E \bullet \\
\quad consFToGFGRefs(vf, (\{el\} \cup Er), M) = fe \Leftrightarrow \\
\qquad (\exists F : Fr;\ GFG : GFGr \bullet F = mfdef\, M\, vf \land GFG = mgfg\, M \\
\qquad \land (\exists ef : gfgEs\, GFG \bullet (src(gfgG\, GFG)ef = vf \\
\qquad \land tgt(gfgG\, GFG)ef = fromV((ftgtr\, F\, el), M))))
\end{array}
$$

$$
\begin{array}{|l}
mUMToGFG : Mdl_0 \to GrMorph \\
buildUFsToGFG : (V \pfun Fr) \times Mdl_0 \to GrMorph \\
\hline
\forall M : Mdl_0;\ fv : V \pfun V;\ fe : E \pfun E \bullet mUMToGFG\ M = (fv, fe) \Leftrightarrow \\
\quad (\exists F : Fr \bullet F = UFs\ M \land (fv, fe) = buildUFsToGFG((mfdef\ M), M)) \\
\forall vf : V;\ F : Fr;\ M : Mdl_0 \bullet buildUFsToGFG(\{(vf \mapsto F)\}, M) = consFToGFG(vf, M) \\
\forall vf : V;\ F : Fr;\ fdef : V \pfun Fr;\ M : Mdl_0 \bullet \\
\quad buildUFsToGFG((\{(vf \mapsto F)\} \cup fdef), M) = consFToGFG(vf, M) \cup_{GM} buildUFsToGFG(fdef, M)
\end{array}
$$

$$
\begin{array}{|l}
mFrToFG : Mdl_0 \times V \to GrMorph \\
\hline
\forall M : Mdl_0;\ vf : V;\ fv : V \pfun V;\ fe : E \pfun E \bullet \\
\quad mFrToFG(M, vf) = (fv, fe) \Leftrightarrow vf \in gfgNs(mgfg\ M) \\
\quad\quad \land (\exists F : Fr;\ ef : gfgEs(mgfg\ M) \bullet \\
\quad\quad\quad (F = mfdef\ M\ vf \land src(gfgG(mgfg\ M))ef = vf \land tgt(gfgG(mgfg\ M))ef = vf \\
\quad\quad\quad \land fv \in fNs\ F \to gfgNs(mgfg\ M) \\
\quad\quad\quad \land fe \in fEs\ F \to gfgEs(mgfg\ M) \land fv = fNs\ F \times \{vf\} \land fe = fEs\ F \times \{ef\}))
\end{array}
$$

$$
\begin{array}{l}
Mdl == \{M : Mdl_0 \mid mUMToGFG\ M \in morphFGFG((UFs\ M), (mgfg\ M)) \\
\quad \land (\forall vf_1, vf_2 : gfgNs(mgfg\ M) \mid vf_1 \neq vf_2 \bullet \mathrm{disjFs}(mfdef\ M\ vf_1, mfdef\ M\ vf_2))\}
\end{array}
$$

B.10 Typed Structural Graphs

section *Fragmenta_TySGs* **parents** *standard_toolkit*, *Fragmenta_SGs*

$$TSGr == \{SG : SGr;\ iet : E \pfun SGET \mid iet \in EsA\ SG \to SGET\}$$

$$
\begin{array}{|l}
tsgSG : TSGr \to SGr \\
tsgiet : TSGr \to E \pfun SGET \\
tsgEsA : TSGr \to \mathbb{P}\ E \\
tsgEsC : TSGr \to \mathbb{P}\ E \\
tsgsrcm : TSGr \to E \pfun Mult \\
tsgtgtm : TSGr \to E \pfun Mult \\
\hline
\forall SG : SGr;\ iet : E \pfun SGET \bullet tsgSG(SG, iet) = SG \\
\forall SG : SGr;\ iet : E \pfun SGET \bullet tsgiet(SG, iet) = iet \\
\forall TSG : TSGr \bullet tsgEsA\ TSG = EsA(tsgSG\ TSG) \\
\forall TSG : TSGr \bullet tsgEsC\ TSG = EsTy((tsgSG\ TSG), \{ecomp\}) \\
\forall TSG : TSGr \bullet tsgsrcm\ TSG = srcm(tsgSG\ TSG) \\
\forall TSG : TSGr \bullet tsgtgtm\ TSG = tgtm(tsgSG\ TSG)
\end{array}
$$

relation$(instanceEdgesOk\ \_)$

$$instanceEdgesOk\_ : \mathbb{P}(SGr \times SGr \times (E \rightarrow\!\!\!\!\rightarrow SGET) \times GrMorph)$$

$$\forall SG, TSG : SGr;\ iet : E \rightarrow\!\!\!\!\rightarrow SGET;\ type : GrMorph \bullet$$
$$(instanceEdgesOk(SG, TSG, iet, type)) \Leftrightarrow iet \circ fE\ type = ety\ SG$$

$$SGrTy == \{SG : SGr;\ TSG : TSGr;\ type : GrMorph \mid type \in morphSG(SG, (tsgSG\ TSG)) \land (instanceEdgesOk(SG, tsgSG\ TSG, tsgiet\ TSG, type))\}$$

$$sgtSG : SGrTy \rightarrow SGr$$
$$sgtTSG : SGrTy \rightarrow TSGr$$
$$sgtType : SGrTy \rightarrow GrMorph$$

$$\forall SG : SGr;\ TSG : TSGr;\ type : GrMorph \bullet sgtSG(SG, TSG, type) = SG$$
$$\forall SG : SGr;\ TSG : TSGr;\ type : GrMorph \bullet sgtTSG(SG, TSG, type) = TSG$$
$$\forall SG : SGr;\ TSG : TSGr;\ type : GrMorph \bullet sgtType(SG, TSG, type) = type$$

$$sgtNs : SGrTy \rightarrow \mathbb{P}\ V$$
$$sgtEs : SGrTy \rightarrow \mathbb{P}\ E$$
$$sgtEsI : SGrTy \rightarrow \mathbb{P}\ E$$
$$sgtSrc : SGrTy \rightarrow E \rightarrow\!\!\!\!\rightarrow V$$
$$sgtTgt : SGrTy \rightarrow E \rightarrow\!\!\!\!\rightarrow V$$

$$\forall SGT : SGrTy \bullet sgtNs\ SGT = sgr\_Ns(sgtSG\ SGT)$$
$$\forall SGT : SGrTy \bullet sgtEs\ SGT = sgr\_Es(sgtSG\ SGT)$$
$$\forall SGT : SGrTy \bullet sgtEsI\ SGT = EsTy((sgtSG\ SGT), \{einh\})$$
$$\forall SGT : SGrTy \bullet sgtSrc\ SGT = sgr\_src(sgtSG\ SGT)$$
$$\forall SGT : SGrTy \bullet sgtTgt\ SGT = sgr\_tgt(sgtSG\ SGT)$$

relation$(abstractNoDirectInstances\ \_\ )$

$$abstractNoDirectInstances\_ : \mathbb{P}(SGr \times SGr \times GrMorph)$$

$$\forall SG : SGr;\ TSG : SGr;\ type : GrMorph \bullet (abstractNoDirectInstances(SG, TSG, type)) \Leftrightarrow$$
$$(fV\ type)^{\sim} (\!| NsTy(TSG, \{nabst\}) |\!) = \{\}$$

relation$(containmentNoSharing\ \_\ )$

$$containmentNoSharing\_ : \mathbb{P}(SGr \times SGr \times GrMorph)$$

$$\forall SG : SGr;\ TSG : SGr;\ type : GrMorph \bullet (containmentNoSharing(SG, TSG, type)) \Leftrightarrow$$
$$((fE\ type)^{\sim} (\!| EsTy(TSG, \{ecomp\}) |\!)) \lhd tgtst\ SG \in injrel$$

relation$(instMultsOk\_)$

$$
\begin{array}{|l}
instMultsOk\_ : \mathbb{P}(SGr \times SGr \times GrMorph) \\
\hline
\forall SG : SGr;\ TSG : SGr;\ type : GrMorph \bullet (instMultsOk(SG, TSG, type)) \Leftrightarrow \\
\quad (\forall te : EsA\ TSG \bullet (\exists r : V \leftrightarrow V \bullet r = rel(restrict((gr\ SG), ((fE\ type)^{\sim} (\!| \{te\} |\!))))) \\
\quad \land (\forall v : \mathrm{dom}\ r \bullet (multOk(r (\!| \{v\} |\!), srcm\ TSG\ te))) \\
\quad \land (\forall v : \mathrm{ran}\ r \bullet (multOk(r^{\sim} (\!| \{v\} |\!), tgtm\ TSG\ te)))))
\end{array}
$$

relation$(instContainmentAcyclic\_)$

$$
\begin{array}{|l}
instContainmentAcyclic\_ : \mathbb{P}(SGr \times SGr \times GrMorph) \\
\hline
\forall SG : SGr;\ TSG : SGr;\ type : GrMorph \bullet (instContainmentAcyclic(SG, TSG, type)) \Leftrightarrow \\
\quad (acyclicG\ restrict((gr\ SG), ((fE\ type)^{\sim} (\!| EsTy(TSG, \{ecomp\}) |\!))))
\end{array}
$$

relation$(isConformable\_)$

$$
\begin{array}{|l}
isConformable\_ : \mathbb{P}(SGr \times SGr \times GrMorph) \\
\hline
\forall SG, TSG : SGr;\ type : GrMorph \bullet (isConformable(SG, TSG, type)) \Leftrightarrow \\
\quad (abstractNoDirectInstances(SG, TSG, type)) \land (containmentNoSharing(SG, TSG, type)) \\
\quad \land (instMultsOk(SG, TSG, type)) \land (instContainmentAcyclic(SG, TSG, type))
\end{array}
$$

$$SGTyConf == \{SG : SGr;\ TSG : TSGr;\ type : GrMorph \mid isConformable(SG, tsgSG\ TSG, type)\}$$

$$
\begin{array}{l}
morphSGT == (\lambda\ SGT_1, SGT_2 : SGrTy \bullet \\
\quad \{m : morphSG((sgtSG\ SGT_1), (sgtSG\ SGT_2)) \mid sgtType\ SGT_2 \circ_G m = sgtType\ SGT_1\})
\end{array}
$$

B.11 Typed Fragments

section $Fragmenta\_TyFrs$ **parents** $standard\_toolkit, Fragmenta\_Frs$

$$TFr == \{F : Fr;\ iet : E \rightarrow\!\!\!\!\!\rightarrow SGET \mid iet \in EsA(fsg\ F) \rightarrow SGET\}$$

$$
\begin{array}{|l}
tfG : TFr \rightarrow Gr \\
tfNs : TFr \rightarrow \mathbb{P}\ V \\
tfEs : TFr \rightarrow \mathbb{P}\ E \\
tfEsR : TFr \rightarrow \mathbb{P}\ E \\
tfF : TFr \rightarrow Fr \\
tfiet : TFr \rightarrow E \nrightarrow SGET \\
\hline
\forall F : Fr;\ iet : E \nrightarrow SGET \bullet tfG(F, iet) = fsrcGr\ F \\
\forall F : Fr;\ iet : E \nrightarrow SGET \bullet tfNs(F, iet) = fNs\ F \\
\forall F : Fr;\ iet : E \nrightarrow SGET \bullet tfEs(F, iet) = fEs\ F \\
\forall F : Fr;\ iet : E \nrightarrow SGET \bullet tfEsR(F, iet) = fEsR\ F \\
\forall F : Fr;\ iet : E \nrightarrow SGET \bullet tfF(F, iet) = F \\
\forall F : Fr;\ iet : E \nrightarrow SGET \bullet tfiet(F, iet) = iet
\end{array}
$$

function 10 **leftassoc** $(\_UTF\_)$

$$
\begin{array}{|l}
\_UTF\_ : TFr \times TFr \rightarrow TFr \\
\hline
\forall TF_1, TF_2 : TFr \bullet TF_1\, UTF TF_2 = (tfF\ TF_1 \cup_F tfF\ TF_2, tfiet\ TF_1 \cup tfiet\ TF_2)
\end{array}
$$

$$FrTy == \{F : Fr;\ TF : TFr;\ type : GrMorph \mid type \in morphF(F, (tfF\ TF))\}$$

relation$(instanceEdgeTypesOkF\ \_)$

$$
\begin{array}{|l}
instanceEdgeTypesOkF\_ : \mathbb{P}(Fr \times TFr \times GrMorph) \\
\hline
\forall F : Fr;\ TF : TFr;\ type : GrMorph \bullet (instanceEdgeTypesOkF(F, TF, type)) \Leftrightarrow \\
\quad tfiet\ TF \circ fE\ type = ety(fsg\ F)
\end{array}
$$

relation$(abstractNoDirectInstancesF\ \_)$

$$
\begin{array}{|l}
abstractNoDirectInstancesF\_ : \mathbb{P}\ FrTy \\
\hline
\forall F : Fr;\ TF : TFr;\ type : GrMorph \bullet (abstractNoDirectInstancesF(F, TF, type)) \Leftrightarrow \\
\quad (fV\ type)^{\sim} (\!| fr\_NsAbst(tfF\ TF) |\!) = \{\}
\end{array}
$$

relation$(containmentNoSharingF\ \_)$

$$
\begin{array}{|l}
containmentNoSharingF\_ : \mathbb{P}(Fr \times Fr \times GrMorph) \\
\hline
\forall F, TF : Fr;\ type : GrMorph \bullet (containmentNoSharingF(F, TF, type)) \Leftrightarrow \\
\quad ((fE\ type)^{\sim} (\!| EsTy((fsg\ TF), \{ecomp\}) |\!)) \lhd tgtstF\ F \in injrel
\end{array}
$$

relation$(instMultsOkF\ \_)$

$$\begin{array}{|l}
instMultsOkF_ : \mathbb{P}(Fr \times Fr \times GrMorph) \\
\hline
\forall F, TF : Fr;\ type : GrMorph \bullet \\
\quad instMultsOkF(F, TF, type) \Leftrightarrow (\forall te : EsA(fsg\ TF) \bullet \\
\qquad (\exists r : V \leftrightarrow V \bullet r = rel(restrict((fsrcGr\ F), ((fE\ type)^{\sim} (\!|\{te\}|\!))))) \\
\qquad \land (\forall v : \mathrm{dom}\, r \bullet (multOk(r (\!| (repsOf\ v\ F) |\!), srcm(fsg\ TF)te))) \\
\qquad \land (\forall v : \mathrm{ran}\, r \bullet (multOk(r^{\sim} (\!|(repsOf\ v\ F) |\!), tgtm(fsg\ TF)te)))))
\end{array}$$

relation$(instContainmentForest\ \_)$

$$\begin{array}{|l}
instContainmentForest_ : \mathbb{P}(Fr \times Fr \times GrMorph) \\
\hline
\forall F, TF : Fr;\ type : GrMorph \bullet instContainmentForest(F, TF, type) \Leftrightarrow \\
\quad rel(restrict((fsrcGr\ F), ((fE\ type)^{\sim} (\!|EsTy((fsg\ TF), \{ecomp\}) |\!)))) \in forest
\end{array}$$

relation$(isConformableF\ \_)$

$$\begin{array}{|l}
isConformableF_ : \mathbb{P}(Fr \times TFr \times GrMorph) \\
\hline
\forall F : Fr;\ TF : TFr;\ type : GrMorph \bullet (isConformableF(F, TF, type)) \Leftrightarrow \\
\quad (instanceEdgeTypesOkF(F, TF, type)) \land (abstractNoDirectInstancesF(F, TF, type)) \\
\quad \land (containmentNoSharingF(F, tfF\ TF, type)) \\
\quad \land (instMultsOkF(F, tfF\ TF, type)) \land (instContainmentForest(F, tfF\ TF, type))
\end{array}$$

$FrTyConf == \{FT : FrTy \mid isConformableF\ FT\}$

B.12 Typed Models

section $Fragmenta\_TyMdls$ **parents** $standard\_toolkit, Fragmenta\_TyFrs, Fragmenta\_Mdls$

$$\begin{array}{l}
TMdl_0 == \{GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \rightarrow\!\!\!\!\!\!+\ TFr \mid \\
\quad fcl \in morphGFGCG(GFG, CG) \land fdef \in gfgNs\ GFG \rightarrow TFr\}
\end{array}$$

$$\begin{array}{|l}
tmGFG : TMdl_0 \rightarrow GFGr \\
tmCG : TMdl_0 \rightarrow CGr \\
tmfcl : TMdl_0 \rightarrow GrMorph \\
tmfdef : TMdl_0 \rightarrow V \nrightarrow TFr \\
\hline
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \nrightarrow TFr \bullet tmGFG(GFG, CG, fcl, fdef) = GFG \\
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \nrightarrow TFr \bullet tmCG(GFG, CG, fcl, fdef) = CG \\
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \nrightarrow TFr \bullet tmfcl(GFG, CG, fcl, fdef) = fcl \\
\forall GFG : GFGr;\ CG : CGr;\ fcl : GrMorph;\ fdef : V \nrightarrow TFr \bullet tmfdef(GFG, CG, fcl, fdef) = fdef
\end{array}$$

$$\begin{array}{|l}
UTFs : TMdl_0 \rightarrow TFr \\
UTFs_0 : \mathbb{P}_1\ TFr \rightarrow TFr \\
\hline
\forall TM : TMdl_0 \bullet UTFs\ TM = UTFs_0(\mathrm{ran}(tmfdef\ TM)) \\
\forall TF : TFr \bullet UTFs_0\ \{TF\} = TF \\
\forall TF : TFr;\ TFs : \mathbb{P}\ TFr \bullet UTFs_0(\{TF\} \cup TFs) = TF\ UTF(UTFs_0\ TFs)
\end{array}$$

$$\begin{array}{|l}
fromVT : V \times TMdl_0 \rightarrow V \\
\hline
\forall vl : V;\ TM : TMdl_0;\ vf : V \bullet fromVT(vl, TM) = vf \Leftrightarrow vl \in tfNs(tmfdef\ TM\ vf)
\end{array}$$

$$\begin{array}{|l}
consTFToGFG : V \times TMdl_0 \rightarrow GrMorph \\
consTFToGFGRefs : V \times \mathbb{P}\ E \times TMdl_0 \rightarrow E \nrightarrow E \\
\hline
\forall vf : V;\ TM : TMdl_0;\ fv : V \nrightarrow V;\ fe : E \nrightarrow E \bullet \\
\quad consTFToGFG(vf, TM) = (fv, fe) \Leftrightarrow (\exists TF : TFr;\ GFG : GFGr \bullet \\
\qquad TF = tmfdef\ TM\ vf \wedge GFG = tmGFG\ TM \wedge fv \in tfNs\ TF \rightarrow gfgNs\ GFG \\
\qquad \wedge fe \in tfEs\ TF \rightarrow gfgEs\ GFG \wedge vf \in gfgNs\ GFG \\
\qquad \wedge (\exists ef : gfgEs\ GFG \bullet \\
\qquad\quad (src(gfgG\ GFG)ef = tgt(gfgG\ GFG)ef = vf \wedge fv = tfNs\ TF \times \{vf\} \\
\qquad\quad \wedge fe = (tfEs\ TF \setminus tfEsR\ TF \times \{ef\}) \cup consTFToGFGRefs(vf, (tfEsR\ TF), TM)))) \\
\forall vf : V;\ TM : TMdl_0;\ fe : E \nrightarrow E \bullet consTFToGFGRefs(vf, \{\}, TM) = \{\} \\
\forall vf : V;\ TM : TMdl_0;\ el : E;\ Er : \mathbb{P}\ E;\ fe : E \nrightarrow E \bullet \\
\quad consTFToGFGRefs(vf, (\{el\} \cup Er), TM) = fe \Leftrightarrow (\exists TF : TFr;\ GFG : GFGr \bullet \\
\quad TF = tmfdef\ TM\ vf \wedge GFG = tmGFG\ TM \\
\quad \wedge (\exists ef : gfgEs\ GFG \bullet (src(gfgG\ GFG)ef = vf \\
\quad \wedge tgt(gfgG\ GFG)ef = fromVT((ftgtr(tfF\ TF)el), TM))))
\end{array}$$

$$
\begin{array}{|l}
mUTMToGFG : TMdl_0 \rightarrow GrMorph \\
buildUTFsToGFG : (V \pfun TFr) \times TMdl_0 \rightarrow GrMorph \\
\hline
\forall TM : TMdl_0;\ fv : V \pfun V;\ fe : E \pfun E \bullet \\
\quad mUTMToGFG\ TM = (fv, fe) \Leftrightarrow \\
\qquad (\exists TF : TFr \bullet TF = UTFs\ TM \land (fv, fe) = buildUTFsToGFG((tmfdef\ TM), TM)) \\
\forall vf : V;\ TF : TFr;\ TM : TMdl_0 \bullet \\
\quad buildUTFsToGFG(\{(vf \mapsto TF)\}, TM) = consTFToGFG(vf, TM) \\
\forall vf : V;\ TF : TFr;\ fdef : V \pfun TFr;\ TM : TMdl_0 \bullet \\
\quad buildUTFsToGFG((\{(vf \mapsto TF)\} \cup fdef), TM) = \\
\qquad consTFToGFG(vf, TM) \cup_{GM} buildUTFsToGFG(fdef, TM)
\end{array}
$$

$$
\begin{array}{l}
TMdl == \{TM : TMdl_0 \mid \exists m : GrMorph \bullet m = mUTMToGFG\ TM \\
\quad \land m \in morphFGFG((tfF(UTFs\ TM)), (tmGFG\ TM)) \\
\quad \land (\forall vf_1, vf_2 : gfgNs(tmGFG\ TM) \bullet (vf_1 \neq vf_2 \Rightarrow ((fV\ m)^{\sim} (\!| \{vf_1\} |\!)) \cap ((fV\ m)^{\sim} (\!| \{vf_2\} |\!)) = \emptyset))\}
\end{array}
$$

$$
\begin{array}{l}
MdlTy == \{M : Mdl;\ TM : TMdl;\ tcg, tgfg, ty : GrMorph \mid \\
\quad \exists FM : Fr;\ FTM : TFr \bullet FM = UFs\ M \land FTM = UTFs\ TM \\
\qquad \land tcg \in morphCG((mcg\ M), (tmCG\ TM)) \\
\qquad \land tgfg \in morphGFG((mgfg\ M), (tmGFG\ TM)) \\
\qquad \land (FM, FTM, ty) \in FrTyConf \\
\qquad \land tgfg \circ_G mUMToGFG\ M = mUTMToGFG\ TM \circ_G ty \\
\qquad \land tcg \circ_G mfcl\ M = tmfcl\ TM \circ_G tgfg\}
\end{array}
$$

B.13 Typed Models with Fragmentation Strategies

section *Fragmenta_TyFSMdls* **parents** *standard_toolkit*, *Fragmenta_TyFrs*, *Fragmenta_TyMdls*

$$
\begin{array}{l}
FSs == \{SCG : CGr;\ SGFG : GFGr;\ scl, sgfg : GrMorph \mid \\
\quad scl \in morphGFGCG(SGFG, SCG)\}
\end{array}
$$

$$
\begin{array}{|l}
fsCG : FSs \rightarrow CGr \\
fsGFG : FSs \rightarrow GFGr \\
fsmcl : FSs \rightarrow GrMorph \\
fsmgfg : FSs \rightarrow GrMorph \\
\hline
\forall SCG : CGr;\ SGFG : GFGr;\ mcl, mgfg : GrMorph \bullet \\
fsCG(SCG, SGFG, mcl, mgfg) = SCG \\
\forall SCG : CGr;\ SGFG : GFGr;\ mcl, mgfg : GrMorph \bullet \\
fsGFG(SCG, SGFG, mcl, mgfg) = SGFG \\
\forall SCG : CGr;\ SGFG : GFGr;\ mcl, mgfg : GrMorph \bullet \\
fsmcl(SCG, SGFG, mcl, mgfg) = mcl \\
\forall SCG : CGr;\ SGFG : GFGr;\ mcl, mgfg : GrMorph \bullet \\
fsmgfg(SCG, SGFG, mcl, mgfg) = mgfg
\end{array}
$$

$$TFSMdl == \{TM : TMdl;\ FS : FSs \mid$$
$$(fsmgfg\ FS) \in morphFGFG((tfF(UTFs\ TM)), fsGFG\ FS)\}$$

$$
\begin{array}{|l}
tfsmTM : TFSMdl \to TMdl \\
tfsmFS : TFSMdl \to FSs \\
tfsmscg : TFSMdl \to CGr \\
tfsmsgfg : TFSMdl \to GFGr \\
\hline
\forall TM : TMdl;\ FS : FSs \bullet \\
\quad tfsmTM\,(TM, FS) = TM \\
\forall TM : TMdl;\ FS : FSs \bullet \\
\quad tfsmFS\,(TM, FS) = FS \\
\forall TM : TMdl;\ FS : FSs \bullet \\
\quad tfsmscg\,(TM, FS) = fsCG\ FS \\
\forall TM : TMdl;\ FS : FSs \bullet \\
\quad tfsmsgfg\,(TM, FS) = fsGFG\ FS
\end{array}
$$

$$
\begin{array}{l}
MdlTyFS == \{M : Mdl;\ TM : TFSMdl;\ scg, sgfg, ty : GrMorph \mid \\
\quad scg \in morphCG((mcg\ M), (tfsmscg\ TM)) \\
\quad \land\ sgfg \in morphGFG((mgfg\ M), (tfsmsgfg\ TM)) \\
\quad \land\ (UFs\ M, UTFs\ (tfsmTM\ TM), ty) \in FrTyConf \\
\quad\quad \land\ sgfg \circ_G mUMToGFG\ M = fsmgfg\,(tfsmFS\ TM) \circ_G ty \\
\quad \land\ scg \circ_G mfcl\ M = fsmcl(tfsmFS\ TM) \circ_G sgfg\}
\end{array}
$$

B.14 Colimit Composition

section *Fragmenta_Colimit_Composition* **parents** *standard_toolkit*, *Fragmenta_GraphsCat*, *Fragmenta_Mdls*

$$
\begin{array}{|l}
emptyG : Gr \\
\hline
emptyG = (\varnothing, \varnothing, \varnothing, \varnothing)
\end{array}
$$

$$
\begin{array}{|l}
addNodeToGr : V \times Gr \to Gr \\
\hline
\forall v : V;\ G, G' : Gr \bullet addNodeToGr(v, G) = G' \Leftrightarrow G' = (Ns\ G \cup \{v\}, Es\ G, src\ G, tgt\ G)
\end{array}
$$

$$
\begin{array}{|l}
addEdgeToGr : E \times V \times V \times Gr \to Gr \\
\hline
\forall e : E;\ v_1, v_2 : V;\ G, G' : Gr \bullet \\
\quad addEdgeToGr(e, v_1, v_2, G) = G \Leftrightarrow e \in Es\ G \lor \neg\ v_1 \in Ns\ G \lor \neg\ v_2 \in Ns\ G \\
\forall e : E;\ v_1, v_2 : V;\ G, G' : Gr \bullet \\
\quad addEdgeToGr(e, v_1, v_2, G) = G' \Leftrightarrow \neg\ e \in Es\ G \land v_1 \in Ns\ G \land v_2 \in Ns\ G \\
\quad \land\ G' = (Ns\ G, Es\ G \cup \{e\}, src\ G \cup \{(e \mapsto v_1)\}, tgt\ G \cup \{(e \mapsto v_2)\})
\end{array}
$$

$$
\begin{array}{|l}
\mathit{emptyDiag} : \mathit{Cat} \rightarrow \mathit{Diag} \\
\hline
\forall\, C : \mathit{Cat} \bullet \mathit{emptyDiag}\ C = (C, \mathit{emptyG}, (\varnothing, \varnothing))
\end{array}
$$

$$
\begin{array}{|l}
\mathit{addNodeToDiag} : V \times O \times \mathit{Diag} \rightarrow \mathit{Diag} \\
\hline
\forall\, \mathit{vf} : V;\ A : O;\ D, D' : \mathit{Diag} \mid \mathit{vf} \in \mathit{Ns}(\mathit{grD}\ D) \bullet \mathit{addNodeToDiag}(\mathit{vf}, A, D) = D \\
\\
\forall\, \mathit{vf} : V;\ A : O;\ D, D' : \mathit{Diag} \mid \neg\ \mathit{vf} \in \mathit{Ns}(\mathit{grD}\ D) \bullet \\
\quad \mathit{addNodeToDiag}(\mathit{vf}, A, D) = D' \Leftrightarrow (\exists\, G' : \mathit{Gr};\ m' : \mathit{MorphG2C} \bullet G' = \mathit{addNodeToGr}(\mathit{vf}, (\mathit{grD}\ D)) \wedge m' = (mV(\mathit{morph}
\end{array}
$$

$$
\begin{array}{|l}
\mathit{addEdgeToDiag} : E \times V \times V \times M \times \mathit{Diag} \rightarrow \mathit{Diag} \\
\hline
\forall\, e : E;\ \mathit{vf}_1, \mathit{vf}_2 : V;\ m : M;\ D, D' : \mathit{Diag} \mid \\
\quad \neg\ \mathit{vf}_1 \in \mathit{Ns}(\mathit{grD}\ D) \vee \neg\ \mathit{vf}_2 \in \mathit{Ns}(\mathit{grD}\ D) \vee e \in \mathit{Es}(\mathit{grD}\ D) \bullet \\
\quad \mathit{addEdgeToDiag}(e, \mathit{vf}_1, \mathit{vf}_2, m, D) = D \\
\forall\, e : E;\ \mathit{vf}_1, \mathit{vf}_2 : V;\ m : M;\ D, D' : \mathit{Diag} \mid \\
\quad \mathit{vf}_1 \in \mathit{Ns}(\mathit{grD}\ D) \wedge \mathit{vf}_2 \in \mathit{Ns}(\mathit{grD}\ D) \wedge \neg\ e \in \mathit{Es}(\mathit{grD}\ D) \bullet \\
\quad \mathit{addEdgeToDiag}(e, \mathit{vf}_1, \mathit{vf}_2, m, D) = D' \Leftrightarrow (\exists\, G : \mathit{Gr};\ \mathit{mD} : \mathit{MorphG2C} \bullet \\
\qquad G = \mathit{addEdgeToGr}(e, \mathit{vf}_1, \mathit{vf}_2, (\mathit{grD}\ D)) \\
\quad \wedge\ \mathit{mD} = (mV(\mathit{morphD}\ D), mE(\mathit{morphD}\ D) \cup \{(e \mapsto m)\}) \wedge D' = (\mathit{cat}\ D, G, \mathit{mD}))
\end{array}
$$

$$
\begin{array}{|l}
\mathit{buildStartDiag} : V \times \mathit{Mdl} \rightarrow \mathit{Diag} \\
\hline
\forall\, \mathit{vf} : V;\ M : \mathit{Mdl};\ D : \mathit{Diag} \bullet \\
\quad \mathit{buildStartDiag}(\mathit{vf}, M) = \mathit{addNodeToDiag}(\mathit{vf}, (\mathit{OGrToGr}^{\sim})(\mathit{fsrcGr}(\mathit{mfdef}\ M\ \mathit{vf})), \mathit{emptyDiag}\ \mathit{GraphsC})
\end{array}
$$

$$
\begin{array}{|l}
\mathit{diagDepNodes} : \mathbb{P}\, V \times \mathit{Mdl} \times \mathit{Diag} \rightarrow \mathit{Diag} \\
\hline
\forall\, M : \mathit{Mdl};\ D : \mathit{Diag} \bullet \mathit{diagDepNodes}(\{\}, M, D) = D \\
\forall\, \mathit{vfs} : \mathbb{P}\, V;\ \mathit{vf}_1 : V;\ M : \mathit{Mdl};\ D, D' : \mathit{Diag} \bullet \\
\quad \mathit{diagDepNodes}((\{\mathit{vf}_1\} \cup \mathit{vfs}), M, D) = D' \Leftrightarrow \\
\qquad (\exists\, D_0, D_1, D_2 : \mathit{Diag} \bullet D_0 = \mathit{addNodeToDiag}(\mathit{vf}_1, (\mathit{OGrToGr}^{\sim})(\mathit{fsrcGr}(\mathit{mfdef}\ M\ \mathit{vf}_1)), D) \\
\qquad\quad \wedge\ D_1 = \mathit{diagDepNodes}((\mathit{importsOf}(\mathit{vf}_1, (\mathit{mgfg}\ M))), M, D_0) \\
\qquad\quad \wedge\ D_2 = \mathit{diagDepNodes}((\mathit{continuationsOf}(\mathit{vf}_1, (\mathit{mgfg}\ M))), M, D_1) \\
\qquad\quad \wedge\ D' = \mathit{diagDepNodes}(\mathit{vfs}, M, D_2))
\end{array}
$$

$$
\begin{array}{|l}
\mathit{addMergeMorphisms} : \mathit{Gr} \times \mathit{Mdl} \times \mathit{Diag} \times V \times \mathbb{P}\, V \rightarrow \mathit{Diag} \\
\hline
\forall\, \mathit{GI} : \mathit{Gr};\ M : \mathit{Mdl};\ D : \mathit{Diag};\ v : V \bullet \mathit{addMergeMorphisms}(\mathit{GI}, M, D, v, \varnothing) = D \\
\forall\, \mathit{GI} : \mathit{Gr};\ M : \mathit{Mdl};\ D, D' : \mathit{Diag};\ \mathit{vs}, \mathit{vt} : V;\ \mathit{vls} : \mathbb{P}\, V \bullet \\
\quad \mathit{addMergeMorphisms}(\mathit{GI}, M, D, \mathit{vs}, (\{\mathit{vt}\} \cup \mathit{vls})) = D' \Leftrightarrow \\
\qquad (\exists\, \mathit{vfs}, \mathit{vft} : V;\ F : \mathit{Fr};\ m, \mathit{mM} : \mathit{GrMorph};\ e : E;\ D_0, D_1 : \mathit{Diag} \bullet \\
\qquad\quad \mathit{mM} = \mathit{mUMToGFG}\ M \wedge \mathit{vft} = \mathit{fV}\ \mathit{mM}\ \mathit{vt} \wedge \mathit{vfs} = \mathit{fV}\ \mathit{mM}\ \mathit{vs} \wedge \neg\ e \in \mathit{Es}(\mathit{grD}\ D) \\
\qquad\quad \wedge\ F = \mathit{mfdef}\ M\ \mathit{vft} \wedge D_0 = \mathit{addNodeToDiag}(\mathit{vft}, (\mathit{OGrToGr}^{\sim})(\mathit{fsrcGr}\ F), D) \\
\qquad\quad \wedge\ m \in \mathit{morphG}(\mathit{GI}, (\mathit{fsrcGr}\ F)) \wedge m = (\{\mathit{vs} \mapsto \mathit{vt}\}, \varnothing) \\
\qquad\quad \wedge\ D_1 = \mathit{addEdgeToDiag}(e, \mathit{vfs}, \mathit{vft}, (\mathit{MGrToGrM}^{\sim})m, D_0) \\
\qquad \wedge\ D' = \mathit{addMergeMorphisms}(\mathit{GI}, M, D, \mathit{vs}, \mathit{vls}))
\end{array}
$$

relation$(HasImpRefs\ \_)$

$$
\begin{array}{|l}
HasImpRefs\_ : \mathbb{P}(V \times V \times Mdl) \\
\hline
\forall vf_1, vf_2 : V;\ M : Mdl \bullet (HasImpRefs(vf_1, vf_2, M)) \Leftrightarrow \\
\quad (\exists F_1, F_2 : Fr \bullet F_1 = mfdef\ M\ vf_1 \land F_2 = mfdef\ M\ vf_2 \land refs\ F_1 \rhd fNs\ F_2 \neq \varnothing)
\end{array}
$$

$$
\begin{array}{|l}
diagRefs : V \times \mathbb{P}\ V \times Mdl \times Diag \rightarrow Diag \\
\hline
\forall vf : V;\ M : Mdl;\ D : Diag \bullet diagRefs(vf, \varnothing, M, D) = D \\
\\
\forall vf_1, vf_2 : V;\ svf : \mathbb{P}\ V;\ M : Mdl;\ D : Diag \bullet \\
\quad diagRefs(vf_1, (\{vf_2\} \cup svf), M, D) = diagRefs(vf_1, svf, M, D) \Leftrightarrow \neg\ HasImpRefs(vf_1, vf_2, M) \\
\\
\forall vf_1, vf_2 : V;\ svf : \mathbb{P}\ V;\ M : Mdl;\ D, D' : Diag \bullet \\
\quad diagRefs(vf_1, (\{vf_2\} \cup svf), M, D) = D' \Leftrightarrow \\
\qquad HasImpRefs(vf_1, vf_2, M) \\
\qquad\quad \land (\exists F_1, F_2 : Fr;\ GI : Gr;\ vfi : V;\ m_1, m_2 : GrMorph;\ D_0, D_1, D_2 : Diag;\ e_1, e_2 : E \bullet \\
\qquad\qquad (F_1 = mfdef\ M\ vf_1 \land F_2 = mfdef\ M\ vf_2 \\
\qquad\qquad \land GI = (\mathrm{dom}(refs\ F_1 \rhd fNs\ F_2), \varnothing, \varnothing, \varnothing) \land m_1 \in morphG(GI, (fsrcGr\ F_1)) \\
\qquad\qquad \land m_1 = (\mathrm{id}(\mathrm{dom}(refs\ F_1 \rhd fNs\ F_2)), \varnothing) \land m_2 \in morphG(GI, (fsrcGr\ F_2)) \\
\qquad\qquad \land m_2 = (refs\ F_1 \rhd fNs\ F_2, \varnothing) \land \neg\ vfi \in Ns(grD\ D) \\
\qquad\qquad \land D_0 = addNodeToDiag(vfi, (OGrToGr^{\sim})GI, D) \land \neg\ \{e_1, e_2\} \subseteq Es(grD\ D_0) \\
\qquad\qquad \land D_1 = addEdgeToDiag(e_1, vfi, vf_1, (MGrToGrM^{\sim})m_1, D_0) \\
\qquad\qquad \land D_2 = addEdgeToDiag(e_2, vfi, vf_2, (MGrToGrM^{\sim})m_2, D_1) \\
\qquad\qquad \land D' = diagRefs(vf_1, svf, M, D_2)))
\end{array}
$$

$$
\begin{array}{|l}
diagMorphisms : V \times Mdl \times Diag \rightarrow Diag \\
diagMorphisms_0 : V \times Mdl \times Diag \times \mathbb{P}\ V \rightarrow Diag \times \mathbb{P}\ V \\
diagMorphismsSet : \mathbb{P}\ V \times Mdl \times Diag \times \mathbb{P}\ V \rightarrow Diag \times \mathbb{P}\ V \\
\hline
\forall vf : V;\ M : Mdl;\ D, D' : Diag \bullet \\
\quad diagMorphisms(vf, M, D) = D' \Leftrightarrow (\exists p\_vfs : \mathbb{P}\ V \bullet diagMorphisms_0(vf, M, D, \varnothing) = (D', p\_vfs)) \\
\\
\forall vf : V;\ p\_vfs, p\_vfs' : \mathbb{P}\ V;\ M : Mdl;\ D, D' : Diag \bullet \\
\quad diagMorphisms_0(vf, M, D, p\_vfs) = (D', p\_vfs') \Leftrightarrow (\exists F : Fr;\ D_1 : Diag \bullet \\
\qquad F = mfdef\ M\ vf \\
\qquad \land D_1 = diagRefs(vf, (importsOf(vf, (mgfg\ M)) \cup continuesOf(vf, (mgfg\ M))), M, D) \\
\qquad \land diagMorphismsSet((importsOf(vf, (mgfg\ M)) \cup continuesOf(vf, (mgfg\ M))), M, D_1, \\
\qquad\quad (p\_vfs \cup \{vf\})) = (D', p\_vfs')) \\
\\
\forall p\_vfs : \mathbb{P}\ V;\ M : Mdl;\ D : Diag \bullet diagMorphismsSet(\varnothing, M, D, p\_vfs) = (D, p\_vfs) \\
\\
\forall vf : V;\ p\_vfs, vfs : \mathbb{P}\ V;\ M : Mdl;\ D : Diag \bullet \\
\quad diagMorphismsSet((\{vf\} \cup vfs), M, D, p\_vfs) = diagMorphismsSet(vfs, M, D, p\_vfs) \Leftrightarrow vf \in p\_vfs \\
\\
\forall vf : V;\ p\_vfs, p\_vfs', vfs : \mathbb{P}\ V;\ M : Mdl;\ D, D' : Diag \bullet \\
\quad diagMorphismsSet((\{vf\} \cup vfs), M, D, p\_vfs) = (D, p\_vfs') \Leftrightarrow \\
\qquad vf \in p\_vfs \land (\exists D'' : Diag;\ p\_vfs'' : \mathbb{P}\ V \bullet (diagMorphisms_0(vf, M, D, p\_vfs) = (D'', p\_vfs'') \\
\qquad \land diagMorphismsSet(vfs, M, D'', p\_vfs'') = (D', p\_vfs')))
\end{array}
$$

$$
\begin{array}{|l}
compDiag : V \times Mdl \rightarrow Diag \\
\hline
\forall\, vf : V;\ M : Mdl;\ D : Diag \bullet compDiag(vf, M) = D \Leftrightarrow \\
\quad (\exists\, D_0, D_1, D_2 : Diag \bullet D_0 = buildStartDiag(vf, M) \\
\qquad \land\ diagDepNodes((importsOf(vf, (mgfg\ M))), M, D_0) = D_1 \\
\qquad \land\ diagDepNodes((continuesOf(vf, (mgfg\ M))), M, D_1) = D_2 \\
\qquad \land\ diagMorphisms(vf, M, D_2) = D)
\end{array}
$$