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.
## Auxiliary Definitions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Base Mathematical Definitions</td>
<td>33</td>
</tr>
<tr>
<td>A.2 Graphs</td>
<td>33</td>
</tr>
<tr>
<td>A.3 Categories</td>
<td>35</td>
</tr>
<tr>
<td>A.4 Structural Graphs</td>
<td>36</td>
</tr>
<tr>
<td>A.5 Fragments</td>
<td>39</td>
</tr>
<tr>
<td>A.6 Global Fragment Graphs</td>
<td>41</td>
</tr>
<tr>
<td>A.7 Cluster Graphs</td>
<td>42</td>
</tr>
<tr>
<td>A.8 Models</td>
<td>43</td>
</tr>
<tr>
<td>A.9 Category Theory</td>
<td>44</td>
</tr>
<tr>
<td>A.10 Colimit composition</td>
<td>47</td>
</tr>
<tr>
<td>A.11 Typed Structural Graphs</td>
<td>49</td>
</tr>
<tr>
<td>A.12 Typed Fragments</td>
<td>51</td>
</tr>
<tr>
<td>A.13 Typed Models</td>
<td>52</td>
</tr>
</tbody>
</table>

## Z Specification of Fragmenta

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Generics</td>
<td>55</td>
</tr>
<tr>
<td>B.2 Graphs</td>
<td>55</td>
</tr>
<tr>
<td>B.3 Category Theory</td>
<td>58</td>
</tr>
<tr>
<td>B.4 The Graphs Category</td>
<td>60</td>
</tr>
<tr>
<td>B.5 Structural Graphs</td>
<td>61</td>
</tr>
<tr>
<td>B.6 Fragments</td>
<td>63</td>
</tr>
<tr>
<td>B.7 Models</td>
<td>66</td>
</tr>
<tr>
<td>B.8 Typed Structural Graphs</td>
<td>67</td>
</tr>
<tr>
<td>B.9 Typed Fragments</td>
<td>70</td>
</tr>
<tr>
<td>B.10 Typed Models</td>
<td>72</td>
</tr>
<tr>
<td>B.11 Typed Models with Fragmentation Strategies</td>
<td>73</td>
</tr>
<tr>
<td>B.12 Colimit Composition</td>
<td>74</td>
</tr>
</tbody>
</table>
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 [KRM13].

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 [KTPW12]. 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 leaves (continuations).
- A fragment that imports others, on the other hand, gives bottom-up, because defining a root node involves incrementally building upon the leaves (imported fragments). If we see definition as a tree, then top-down involves going down from the root to the leaves; 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. Fig. 2.1(a) shows a simple meta-model of a language to model software

\[^1\text{http://www.mondo-project.org/}\]
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.

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

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 re-
  gion’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 al-
  ternatives: top-down (symbol ↓) and bottom-up (symbol ⊃). Top-down fragmentations are
  realised as continuations; bottom-up as importings. In Fig. 2.1(a), cross-border edges com-
  ing out of WindTurbine and WTS subsystem are top-down; the remaining ones, bottom-up.
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. 4.3) 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, AKMG11, 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. 4.4. 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

...
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 DeclReference_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 DeclReference_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 DeclContract, which represents a contract reference that can be placed in the declarations compartment of a CD. DeclContract specialises DeclReference_AD_CD of fragment F_Decls_AD_CD, which is allowed because DeclReference_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 DeclContract.

- 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 DeclContract 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 model’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.
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 \( P \) to denote a powerset (e.g. \( P\mathbb{N} \)). 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 \( P(\mathbb{N} \times \mathbb{N}) \)).
The symbol \( \rightarrow \) denotes a total function; \( \Rightarrow \) denotes a partial function; and \( \hookrightarrow \) 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. 1). 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 \( s, t: E_G \rightarrow V_G \) for the source and target of edges (see Fig. 3.1(a)). G-morphisms (def. 2) 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. 3). Graphs and their morphisms form category \( \text{Graph} \) (fact 2).

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,
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). 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 $\leq$; SG morphisms may shrink.

---

1In Isabelle, we proved that $src^*$ and $tgt^*$ preserve the information of base source and target functions; see def. 13.
(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 3).

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 Fragmenta’s architecture are as follows:

- **Local Fragment** \((LF_i)\). 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** \((F_i)\). 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** \((C_i)\). 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...
Figures 4.2: Fragments

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.

A fragment (see Fig. 4.2(a)) is a pair \( F = (SG, \text{tr}) \), comprising an SG plus a target function for reference edges (def. 14 defines set \( Fr, F \in Fr \)). Function \( \text{tr} \) is illustrated in Fig. 4.2: in Fragment \( F2 \) of Fig. 4.2(b), for instance, proxy node \( \text{Person} \) (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 \( \text{tr} \) 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.

Fragementa 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

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.
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 5 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. 1.2(c), which puts together fragments $F_0$ and $F_1$ of Fig. 1.2(d). The composition that resolves the references (called *colimit composition*, chapter 5 below) is illustrated in Fig. 1.2(e). The inheritance edges of proxies in Figs. 1.2(d) and 1.2(c) 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 $F_2$ of Fig. 1.2(b), edges *lives* and *owns* pertain to both nodes named *Person*. To support this, fragments extend relations $\prec$, $\ll$, $\text{src}^*$ and $\text{tgt}^*$ of SGs to cover the semantics of references. This extension is based on functions $\text{refs}$, which gives the references relation between proxies and their referred nodes (obtained from a restricted graph that considers reference edges only), and function $\sim\prec$, which yields a relation giving all the representatives of a given node ($\sim\prec = \text{refs}_F \cup (\text{refs}_F)\sim$), and the actual inheritance relation for fragments, which extends the inheritance of SGs with the representatives relation ($\prec = \leq_{\text{sp}} \cup \sim\prec$).

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. 1.2(b) and the one in the inverse direction.

### 4.2 Global Fragment Graphs

*Global fragment graphs* (GFGs) represent fragment relations. A GFG (Fig. 1.3(a)) is a pair $\text{GFG} = (G, ct)$ made of a graph $G$ and an edge colouring function, stating whether the edge is an imports or continues (def. 17 introduces set $\text{GFG}_r$, such that $\text{GFG} \in \text{GFG}_r$). Graph $\text{GFG}_{\text{MONDO}}$ of Fig. 1.3(a) 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. 1.3(b), a CG is a pair $\text{CG} = (G, ct)$ made up of a graph $G$ and an edge colouring function $ct$, stating whether the related clusters are in a relation of imports, continues or contains (see def. 20, which defines set $\text{CG}_r$, such that $\text{CG} \in \text{CG}_r$). Graph $\text{CG}_{\text{MONDO}}$ of Fig. 1.3(b) is an example of a CG; likewise for graph $\text{CG}_{\text{VCL}_\text{AD}_\text{CD}_\text{MM}}$ of Fig. 1.3.

We define two sets of colouring preserving morphisms involving CGs:

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

---

2In Isabelle, we proved that the extensions preserve the information of the corresponding SG relation (e.g. $\leq_p \leq_{\text{sp}}$); see def. 19.
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 \( mc: GFG \rightarrow CG \), and a function \( fd: Ns_{GFG} \rightarrow Fr \) 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 \( \text{UFs} \) def. 23), and from this we can construct a morphism to the model’s GFG (function \( \text{UMToGFG} \), def. 23), and from here the model’s morphism \( mc \) gets to the model’s \( CG \). Figure 4.3(d) illustrates this: \( M_{MONDO} \) at the bottom is the fragment resulting from function \( \text{UFs} \) (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 [EPT06]. 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).
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 function $iet : ES_{SG} \rightarrow SGET$, 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 $SGT_y$, such that $SGT \in SGT_y$).

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)).
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 (Δ - inheritance; ↔ association; ⋆ - composition).

Figure 6.2: Examples of typed structural graphs involving composition and multiplicity.
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 $SG_{TyConf}$ 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 function $iet : EsA_F \rightarrow SGET$, 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 $type : Fr \rightarrow TFr$, mapping the instance fragment to the type one (see def. 33, which defines $FrTy$, such that $FT \in FrTy$).
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.

Figure 6.4: Morphisms of typed Fragmenta models
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 \( \mathcal{FS} = (GFG_S, CG_S, sc, sf) \), 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 \( \mathcal{TFSM} = (TM, FS) \), 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 \( \mathcal{MT} = (M, TM, scg, sgfg, ty) \), 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)).

24
Chapter 7

Discussion

We now discuss the results presented in this report.

**Fragmenta and model design.** Fragmenta aims to fully support separation of concerns effectively. This, however, comes at a complexity cost in terms of 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 all this complexity from the designer to provide a sound environment to enable design of fragmented models that harness separation of concerns. In particular, Fragmenta’s support for both top-down and bottom-up design, means that designers can choose the scheme that best suits their problem and their way of thinking.

To gain the important result of global preservation of inheritance acyclicity checked locally (fact 5), we forbid proxies with supertypes. We do not see this as a serious restriction. It can be seen as a design rule whereby supertypes of some concept must be defined when the concept is first introduced; proxies may then have subtypes, but no supertypes.

**A theory of separation.** Chapter 5 presented the theoretical basis of model colimit-based composition, which amounts to reference resolution through substitution. Fragmenta, however, keeps the models fragmented. The compositions that the theory requires for global purposes are based on the union of all model fragments without reference resolution, a simpler operation. Fragmenta lives, therefore, well with separation; it provides all the required machinery to handle 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 total compositions outlined in chapter 5 as being an aid to designers to get a clean big picture.

**Fragmentation strategies.** FSs complement metalevel definitions of types with a fragmentation structure. This ensures a uniform fragmentation across all model instances that is explicitly declared in the metamodel. Often, the uniformity of such fragmentations is only implicitly agreed among developers with no means to check whether it is enforced or not; in most cases, however, each instance model designer is free to do its own fragmentation, which complicates the processing of collections of such models. The approach presented here enables enforceable FSs whose conformity can be checked by tools. In our theory, the conformance of a model instance fragmentation to its type’s FS is described as a simple commuting of two diagrams, as shown in Fig. 6.4(a).
**Fragmenta and its realisations.** We have implemented Fragmenta in a Eclipse-based tool [GGKdL14] for DSL definition. The theory can be used as the underlying modularity mechanism of other modelling languages with the notions of cluster and fragments realised in its many guises. The visual contract language (VCL) [AKMG10, AG15] provides a packaging mechanism with an underlying mechanism of references that resembles Fragmenta. In VCL, Fragmenta’s clusters are packages and fragments are VCL diagrams. VCL does not provide any support for top-down design. A generic implementation of Fragmenta like the one in [GGKdL14], could greatly simplify the design of a modelling language such as VCL.

<table>
<thead>
<tr>
<th>Verification</th>
<th>268</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>123</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>391</strong></td>
</tr>
</tbody>
</table>

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. The Z specification (very close to the presentation given here), given in appendix B, was then encoded in the Isabelle proof assistant. This step required some meaning-preserving changes to cater to Isabelle’s specificities (e.g., Isabelle has no primitive notion of partial function). Isabelle enabled us to validate and verify Fragmenta; we proved general theorems concerning desired properties (verification) and specific theorems concerning examples (validation). Table 7.1 gives the number of Isabelle proofs that were undertaken.

**Fragmenta and 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 the important concepts of inheritance, composition and multiplicities. Fragmenta’s proxies are an abstraction of EMF proxies [SBPM08] and similar mechanisms used in VCL. Our proved result (fact 5) showing that the well-formedness of an 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.

**Formalisation.** Our formalisation required several design decisions. We formalise inheritance using coloured edges in SGs, as any other edge, unlike similar graphs [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 (differs from [JT12]) also simplifies checking the prescribed edge type to a simple diagram commuting (Fig. 6.3(a)).

References are also treated as coloured edges. Initially, we considered a partial function \( r \) : \( V \rightarrow V \). We opted for the colouring alternative because: it benefits Fragmenta’s uniformity and coherence (all edges are formalised as such), we see such edges reflected in the morphisms from local fragment nodes to GFGs as inter-fragment GFG edges, and because supports both references and merge composition. The drawback of reference edges is that they lie unreferenced.

---

1 The Isabelle theories can be found at [http://www.miso.es/fragmenta/](http://www.miso.es/fragmenta/)
in their SG representation; we need to use the reference target function of fragments to get graphs that are referenced.

To gain the important property of inheritance acyclicity preserved globally (fact 5), we restricted the way proxies make use of inheritance. As discussed above, we do not see this as a serious restriction; given the importance of the result, it is something that is actually worth doing.
Chapter 8

Related Work

MDE’s scalability challenge and the need for modularity has been widely recognised. EMF (a de-facto modelling standard) provides the means to partition models with proxy objects, but lacks support for fragmentation strategies (FSs). To improve this, [SZFK12] proposes a non-formal persistence framework for EMF, where models are fragmented along annotated metamodel composition relations. Our theory is formal, provides a notion of fragmentation regions that is more flexible than the annotated compositions of [SZFK12] and 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 widely used metamodelling language. They provide a formalisation of metamodels with components, comprising an export and an import interface to enable composition. Their definition of metamodel equates to the simple graphs presented here, which, unlike SGs, does not not cover concepts important to metamodelling, 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.

There has been formal work in the context of merge composition [NSC07, SE05], which uses also uses the colimit construction of category theory to explain composition [SE05]. Fragmenta presented here does a more thorough treatment of the proxy mechanism for referencing and incremental definition, which is slightly different from the merge. It also puts forward the simpler union composition, where references are not resolved.

Component graphs [JT12], which build up on distributed graphs, provide two levels of definition: local and network, resembling 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. [JT12] 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, however, it doesn’t provide 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 5) provided some local constraints.
are met.

Hamiaz et al [HPCE14] formalise [HHJZ09] by encoding the model composition operations of [HHJZ09] in the Coq theorem prover. FRAGMENTA is, like [HPCE14], formal and developed with the aid of a proof assistant; it is, however, more abstract, a general approach whose abstract setting tries to mimic common features of MDE; composition is expressed using general mathematical operators, such as colimit and set-union.

Several approaches split monolithic models. Kelsen et al [KMCG11] 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 organization of fragments into clusters.
Chapter 9

Conclusions

This paper presented Fragmenta, a formal theory to fragment MDE models. This paper’s main result (fact 5), 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. 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/1Eq1yZv
References


Appendix A

Auxiliary Definitions

A.1 Base Mathematical Definitions

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

- \(\text{acyclic}[X] = \{ r : X \leftrightarrow X \mid r^+ \cap \text{id}[X] = \emptyset \} \)
- \(\text{connected}[X] = \{ r : X \leftrightarrow X \mid (\forall x : \text{dom } r; \ y : \text{ran } r \bullet x \mapsto y \in r^+) \} \)
- \(\text{tree}[X] = \{ r : X \leftrightarrow X \mid r \in \text{acyclic} \land r \in X \Rightarrow X \} \)
- \(\text{forest}[X] = \{ r : X \leftrightarrow X \mid r \in \text{acyclic} \land (\forall s : X \leftrightarrow X \mid s \subseteq r \land s \in \text{connected} \bullet s \in \text{tree}) \} \)
- \(\text{injrel } Y = \{ r : X \leftrightarrow Y \mid (\forall x : X; \ y_2, y_3 : Y \bullet (x, y_1) \in r \land (x, y_2) \in r \Rightarrow y_1 = y_2 \} \)

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

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.

**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 \(s, t : E_G \rightarrow V_G\).

The set of graphs \(Gr\), such that \(G : Gr\), is defined as:

\[ Gr = \{ (V_G, E_G, s, t) \mid \forall V_G \in \mathbb{P} V \land E_G \in \mathbb{P} E \land s \in E_G \rightarrow V_G \land t \in E_G \rightarrow V_G \} \]

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

- \(N s : Gr \rightarrow \mathbb{P} V\)
- \(E s : Gr \rightarrow \mathbb{P} E\)
- \(s r c : Gr \rightarrow (E \rightarrow V)\)
- \(t g t : Gr \rightarrow (E \rightarrow V)\)
- \(N s(V_G, E_G, s, t) = V_G\)
- \(E s(V_G, E_G, s, t) = E_G\)
- \(s r c(V_G, E_G, s, t) = s\)
- \(t g t(V_G, E_G, s, t) = t\)

In the following, given a graph \(G\), we write \(N s_G\), \(E s_G\), \(s r c_G\) and \(t g t_G\) to yield the components of a graph (nodes, edges, source and target functions), which abbreviates function application (e.g. we write \(N s_G\) to mean \(N s G\)).

We introduce several functions and predicates for graphs: (a) \(E s I d\) gives all self edges, (b) \(a d j a c e n t\) indicates whether any two nodes are adjacent, (c) \(r e l\) yields relation induced by a graph, (d) \(r e s t r i c t\) extracts a sub-graph from the given graph that considers the given set of edges only,
(e) *acyclic* \( G \) says whether a graph is acyclic or not, (f) *disj* says whether two graphs are disjoint, (g) *replaceGfun* does a replacement of nodes on a source or target function, (h) *replaceG* replaces the nodes of a graph given a substitution. 

\[
\begin{align*}
\text{EsId : } G & \rightarrow \mathcal{P} \rightarrow \mathcal{P} E \\
\text{EsId } G = \{ e : \text{EsG} \mid \text{srcG } e = \text{tgtG } e \} \\
\text{rel : } G & \rightarrow (V \rightarrow V) \\
\text{rel } G = \{ (v_1, v_2) \mid \text{adjacent}(v_1, v_2, G) \} \\
\text{acyclicG} : & \mathcal{P}(G) \\
\text{acyclicG } G \Rightarrow \text{rel } G \in \text{acyclic} \\
\text{disj} : & \mathcal{P}(\mathcal{P}(G)) \\
\text{disj } G & \Rightarrow \text{disj } G \in \text{acyclic}
\end{align*}
\]

\[
\text{replaceGfun : } (E \rightarrow V) \rightarrow (V \rightarrow V) \rightarrow (E \rightarrow V) \\
\text{replaceGfun } f \text{ sub } = f \uplus \{ (e, v) \mid e \in \text{dom } f \wedge (f e) \in \text{dom } \text{sub} \wedge v \in V \wedge \text{sub } (f e) = v \} \\
\text{replaceG : } G \rightarrow (V \rightarrow V) \rightarrow G \\
\text{replaceG } G \text{ sub } = (\text{EsG} \setminus \text{dom sub} \cup \text{ran}(\text{EsG} \setminus \text{sub})), \text{EsG}, \text{replaceGfun } \text{srcG } \text{sub}, \text{replaceGfun } \text{tgtG } \text{sub}
\]

Above, symbol \( \prec \) stands for domain restriction; acyclic is set of acyclic relations (definition 4).

**Properties.** The following laws support proof involving graphs:

\[
\begin{align*}
G_1 \in G; G_2 \in G & \vdash \text{disjEs}(G_1, G_2) \Rightarrow \text{disjEs}(G_2, G_1) \\
G_1 \in G; G_2 \in G & \vdash \text{disj}(G_1, G_2) \Rightarrow \text{disj}(G_2, G_1) \\
G_1 \in G; G_2 \in G & \vdash \text{disj}(G_1, G_2) \Rightarrow \text{disj}(\text{restrict } G_1, \text{restrict } G_2) \\
G_1 \in G; G_2 \in G; \text{dom sub} \cup \text{ran sub} & = \emptyset \vdash \text{replaceG } G \text{ sub } \in G \\
G_1 \in G; G_2 \in G; \text{dom sub} \cap \text{ran sub} = \emptyset & \vdash \text{replaceG } G \text{ sub } = G \\
\text{disjEs}(G_1, G_2) & \Rightarrow \text{disjEs}(\text{replaceG } G_1 \text{ sub}, \text{replaceG } G_2 \text{ sub})
\end{align*}
\]

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

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

\[
G_1 \cup G_2 = (\text{EsG} \cup \text{EsG}, \text{EsG}, \cup \text{EsG}, \text{srcG} \cup \text{srcG}, \text{tgtG} \cup \text{tgtG})
\]

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:

\[
\begin{align*}
G_1 \in G; G_2 \in G & \vdash (G_1 \cup G_2) \in G \\
G_1 \in G; G_2 \in G; \text{disjEs}(G_1, G_2) & \Rightarrow G_1 \cup G_2 = G_2 \cup G_1 \\
G_1 \in G; G_2 \in G; \text{disjEs}(G_1, G_2) & \Rightarrow \text{restrict } (G_1 \cup G_2, \text{es}) = \text{restrict } (G_1, \text{es}) \cup \text{restrict } (G_2, \text{es})
\end{align*}
\]

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

**Fact 1** (Graph Acyclicity). The union of graphs \( G_1, G_2 : G \) is acyclic provided: (i) the individual graphs are also acyclic, and (ii) they are mutually disjoint:

\[
G_1 \in G; G_2 \in G \vdash \text{acyclicG } G_1 \cup G_2 \Rightarrow \text{acyclicG } G_1 \wedge \text{acyclicG } G_2 \wedge \text{disj } (G_1, G_2)
\]

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

\[
\begin{align*}
G_1 \in G; G_2 \in G; \text{disjEs}(G_1, G_2) & \vdash \text{adjacent}(x, y, G_1 \cup G_2) \Rightarrow \text{adjacent}(x, y, G_1) \vee \text{adjacent}(x, y, G_2) \\
G_1 \in G; G_2 \in G; \text{adjacent}(x, y, G_1 \cup G_2) & \Rightarrow \text{adjacent}(x, y, G_1) \vee \text{adjacent}(x, y, G_2) \\
G_1 \in G; G_2 \in G; \text{adjacent}(x, y, G_1 \cup G_2) & \Rightarrow \text{adjacent}(x, y, G_1 \cup G_2) \\
G_1 \in G; G_2 \in G; \text{restrict } (G_1 \cup G_2) & \Rightarrow \text{rel } (G_1 \cup G_2) = \text{rel } G_1 \cup \text{rel } G_2 \\
G_1 \in G; G_2 \in G; \text{disjG}(G_1, G_2) & \Rightarrow ((\text{dom } \text{rel } G_1) \cup \text{ran } \text{rel } G_1) \cap ((\text{dom } \text{rel } G_2) \cup \text{ran } \text{rel } G_2) = \emptyset
\end{align*}
\]
Proof. All theorems outlined above were proved in the Isabelle proof assistant. □

Definition 5 (G-Morphisms). A graph morphism \( m : G_1 \to G_2 \) defines a mapping between graphs \( G_1, G_2 : Gr \); it comprises a pair of functions \( m = (f_V, f_E) \), \( f_V : NSG_1 \to NSG_2 \) and \( f_E : ESG_1 \to ESG_2 \), 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 \( G_1 \to G_2 \) (morphisms between two graphs), such that \( G_1 \to G_2 \subseteq GrMorph \), are defined as:

\[
GrMorph = \{ (fe, fe) \mid fe \in V \to V \land fe \in E \to E \}
\]
\[
G_1 \to G_2 = \{ (fv, fe) \mid fv \in NSG_1 \to NSG_2 \land fe \in ESG_1 \to ESG_2 \land fv \circ src_{G_1} = src_{G_2} \circ fe \\
\land fv \circ tgt_{G_1} = tgt_{G_2} \circ fe \}
\]

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:

\[
f_V : GrMorph \to V \to V \land f_E : GrMorph \to E \to E \\
f_V(fv, fe) = fv \land f_E(fv, fe) = fe
\]

□

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

\[
g \circ f = ((fv \circ g) \circ (fv \circ f)).(fv \circ g) \circ (fv \circ f))
\]

□

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. □

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

The base set of all categories is defined as:

\[
Cat_0 = \{ (OC, MC, dm, cd, id_C, \circ) \mid OC \subseteq \mathbb{P} O \land MC \subseteq \mathbb{P} M \land dm \in OC \to MC \\
\land cd \in OC \to MC \land idn \in OC \to MC \land \circ \in MC \times MC \to MC \}
\]

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

\[
obs : Cat_0 \to \mathbb{P} O \\
obs(OC, MC, dm, cd, idn, \circ_C) = OC \\
obs(Morphs(OC, MC, dm, cd, idn, \circ_C)) = MC
\]
\[
dom : Cat_0 \to M \to O \\
dom(OC, MC, dm, cd, idn, \circ_C) = dm \\
\circ(Morphs(OC, MC, dm, cd, idn, \circ_C)) = cd
\]
\[
\text{id} : Cat_0 \to O \to M \\
\text{id}(OC, MC, dm, cd, idn, \circ_C) = \text{idn} \\
\circ(\text{Morphs}(OC, MC, dm, cd, idn, \circ_C)) = \circ
\]

35
In the following, given a category \( C \), we write \( obs_C \), \( morphs_C \), \( dom_C \), \( cod_C \) and \( id_C \) to mean \( obs C \), \( morphs C \), \( dom C \), \( cod C \) and \( id C \), respectively. We write \( g \circ_C f \) to mean \( \circ_C (g, f) \).

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

\[
A \rightarrow_C B = \{ m : morphs_C \mid A \in obs_C \land B \in obs_C \land m = A \land cod_C m = B \}
\]

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

\[
Cat = \{ C : Cat \mid (\forall A : obs_C \bullet id_C A \in A \rightarrow_C A) \\
\land (\forall f, g : morphs_C \mid dom_C g = cod_C f \bullet g \circ_C f \in dom_C f \rightarrow_C cod_C g) \\
\land (\forall A, B, C, D : obs_C \bullet \forall f : A \rightarrow_C B ; g : B \rightarrow_C C ; h : C \rightarrow_C D \bullet \\
h \circ_C (g \circ_C f) = (h \circ_C g) \circ_C f) \\
\land (\forall A, B : obs_C \bullet \forall f : A \rightarrow_C B \bullet id_C B \circ_C f = f \land f \circ id_C A = f) \}
\]

\[\Box\]

**Fact 2** (Category of Graph). We can form the category Graph by taking graphs as category objects (def. 3) and graph morphisms (def. 4) as category morphisms. We define the domain, co-domain, identity and composition of Graph as:

\[
\begin{align*}
domCG : GrMorph & \rightarrow Gr & codCG : GrMorph & \rightarrow Gr \\
domCG m & = G_1 \iff m \in G_1 \rightarrow G_2 & codCG m & = G_2 \iff m \in G_1 \rightarrow G_2 \\
idCG : Gr & \rightarrow GrMorph & \circCG : GrMorph \times GrMorph & \rightarrow GrMorph \\
idCG G_1 & = m \iff m \in G_1 \rightarrow G_2 & gm_1 \circCG m_2 & = m_3 \iff m_3 = m_1 \circ_C m_2
\end{align*}
\]

The category Graph is defined as:

\[\text{Graph} = (Gr, GrMorph, domCG, codCG, idCG, \circCG)\]

**Proof.** All required proofs were done in Isabelle. \[\Box\]

### 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.

\[
SGNT = \{ nnrml, nabst, nprxy \} \quad SGET = \{ einh, ecomp, erel, elnk, eref \}
\]

\[\Box\]

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

\[
\begin{align*}
MultUVal &= N \cup \{ * \} \\
Mult &= \{(lb, ub) \mid lb \in N \land ub \in MultUVal \land (ub = * \lor (ub \in N \land lb \leq ub))\}
\end{align*}
\]

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

\[
\begin{align*}
multOk_{=} : \exists (P \times Mult) \\
multOk_{=} (vs, (lb, ub)) \iff \# vs \geq lb \land (ub = * \lor (ub \in N \land \# vs \leq ub))
\end{align*}
\]

Above, \( \# \) stands for set cardinality. \[\Box\]
Definition 11 (Structural Graphs). A structural graph $SG = (G, nt, ety, sm, tm)$ comprises a graph $G : Gr$, two colouring functions for nodes and edges, $nt : NSG \to SGNT$ and $ety : E_{SG} \to SET$, and source and target multiplicity functions, $sm, tm : E_{SG} \to Mult$ (Fig. 3.1(c)).

Base set $SG0$ of SGs, such that $SG : SG0$, is defined as:

$$SG0 = \{(G, nt, et, sm, tm) \mid G \in Gr \land nt \in NSG \land \exists et \in E_{SG} \land \exists sm, tm \in E_{SG} \land \exists Mult\}$$

The next functions extract the components of a SG:

- $gr : SG0 \to Gr$
- $nty : SG0 \to (V \to SGNT)$
- $ety : SG0 \to (E \to SGNT)$
- $srcm : SG0 \to Mult$
- $tgtrm : SG0 \to Mult$
- $srcm(G, nt, et, sm, tm) = sm$
- $tgtrm(G, nt, et, sm, tm) = tm$

We introduce several functions and predicates to operate upon $SG0$: (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) $<_{G}$ gives SG’s inheritance graph formed as restriction of the SG’s graph to inheritance edges and excluding the dummy self edges, and (g) $\preceq$ is the inheritance relation obtained from the inheritance graph $<_{G}$.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$EsTy$</td>
<td>$SG0 \times P \to SGNT \to P E$</td>
</tr>
<tr>
<td>$EsTy(SG, et)$</td>
<td>$SG0 \to {{esy} \mid nprty}$</td>
</tr>
<tr>
<td>$EsA$</td>
<td>$SG0 \to P E$</td>
</tr>
<tr>
<td>$EsR$</td>
<td>$SG0 \to P E$</td>
</tr>
<tr>
<td>$EsRP$</td>
<td>$SG0 \to P E$</td>
</tr>
<tr>
<td>$EsRYPG$</td>
<td>${ e : EsR SG \mid srcSG e \in NsP SG}$</td>
</tr>
<tr>
<td>$&lt;_{G}$</td>
<td>$SG0 \to Gr$</td>
</tr>
<tr>
<td>$\preceq$</td>
<td>$SG0 \to (V \leftrightarrow V)$</td>
</tr>
</tbody>
</table>

Above, $^*$ 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:

$$SGr = \{ SG : SG0 \mid EsRSG \subseteq EsIdSG \land srcSG \subseteq EsTy(SG, \{ erel, ecomp \}) \to Mult \land tgtSG \subseteq EsTy(SG, \{ ecomp \}) \to Mult \land acyclicG(<_{G} SG) \}$$

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

Auxiliary Definitions. Functions $\preceq$ yields the reflexive transitive closure of $<$.

$$\preceq : SGr \to (V \leftrightarrow V)$$

$\preceq SG = (<SG>)^*$

Here, $^*$ denotes the reflexive transitive closure.

Function $clan$ yields inheritance-family of some SG node using $<^*$. 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$clan$</td>
<td>$V \times SGr \to P V$</td>
</tr>
<tr>
<td>$src^*$</td>
<td>$SGr \to (E \leftrightarrow V)$</td>
</tr>
<tr>
<td>$tgt^*$</td>
<td>$SGr \to (E \leftrightarrow V)$</td>
</tr>
</tbody>
</table>

$$clan(v, SG) = \{ v' \in NSG \mid v' \preceq v \}$$

$$src^* SG = \{ (v, v) \mid v \in EsASG \land v \in NSG \land \exists v_2 : NSG \land v \in clan(v_2, SG) \land srcSG v = v_2 \}$$

$$tgt^* SG = \{ (v, v) \mid v \in EsASG \land v \in NSG \land \exists v_2 : NSG \land v \in clan(v_2, SG) \land tgtSG v = v_2 \}$$
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:

\[
SG \in SGr \vdash EsA_{SG} < src_{SG} \subseteq src^*_{SG} \quad SG \in SGr \vdash EsA_{SG} < tgt_{SG} \subseteq tgt^*_{SG}
\]

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

Definition 12 (Union of Structural Graphs). The union of SGs $SGr_1$, $SGr_2 : SGr$ is defined as:

\[
SGr_1 \cup SGr_2 = (gr SGr_1 \cup gr SGr_2, nty SGr_1 \cup nty SGr_2, ety SGr_1 \cup ety SGr_2, srcm SGr_1 \cup srcm SGr_2, tgtm SGr_1 \cup tgtm SGr_2)
\]

The union of two SGs is defined as the union of the SG’s components, which involves use of graph union (def. 11).

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

\[
\begin{align*}
& \vdash src(SGr_1 \cup SGr_2) = src SGr_1 \cup src SGr_2 \\
& \vdash tgt(SGr_1 \cup SGr_2) = tgt SGr_1 \cup tgt SGr_2 \\
& \vdash EsR(SGr_1 \cup SGr_2) = EsR SGr_1 \cup EsR SGr_2 \\
& \vdash EsId(SGr_1 \cup SGr_2) = EsId SGr_1 \cup EsId SGr_2 \\
& \vdash EsRP(SGr_1 \cup SGr_2) = EsRP SGr_1 \cup EsRP SGr_2 \\
& \vdash \prec (SGr_1 \cup SGr_2) = \prec SGr_1 \cup \prec SGr_2
\end{align*}
\]

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

Fact 3 (Union Composition of Structural Graphs). Given SGs $SGr_1, SGr_2 : SGr$, 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:

\[
SGr_1 \subseteq SGr; SGr_2 \subseteq SGr; disj(SGr_1, SGr_2) \vdash acyclic(<_G (SGr_1 \cup SGr_2)) \Rightarrow acyclic(<_G (SGr_2))
\]

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

\[
SGr_1 \subseteq SGr; SGr_2 \subseteq SGr \vdash (SGr_1 \cup SGr_2) \subseteq SGr
\]

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

\[
\begin{align*}
& SGr_1 \subseteq SGr; SGr_2 \subseteq SGr; disj(SGr_1, SGr_2) \vdash disj(<_G SGr_1,<_G SGr_1) \\
& SGr_1 \subseteq SGr; SGr_2 \subseteq SGr; disj(SGr_1, SGr_2) \vdash (\text{dom } <_{SGr_1} \cup \text{ran } <_{SGr_2}) \cap (\text{dom } <_{SGr_2} \cup \text{ran } <_{SGr_2}) = \emptyset
\end{align*}
\]

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

Definition 13 (SG Morphisms). Given $SGr_1, SGr_2 : SGr$, a SG morphism $m : SGr_1 \rightarrow SGr_2$ is a pair of functions $m = (fv, fe)$ mapping nodes and edges, respectively.

The set of morphisms between two SGs, $SGr_1 \rightarrow SGr_2$, is defined as:

\[
\forall SGr_1, SGr_2 : SGr \bullet \\
SGr_1 \rightarrow SGr_2 = \{(fv, fe) \mid fv \in N_{SGr_1} \rightarrow N_{SGr_2}, fe \in E_{SGr_1} \rightarrow E_{SGr_2}, \\
fv \circ src^*_{SGr_1} \subseteq src^*_{SGr_2}, fe \circ tgt^*_{SGr_1} \subseteq tgt^*_{SGr_2} \circ fe, fv \circ src_{SGr_1} \subseteq src_{SGr_2} \circ fe, fe \circ tgt_{SGr_1} \subseteq tgt_{SGr_2} \circ fv\}
\]

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 4 (Category SGraphs). We can form the category SGraphs by taking SGs as category objects (def. [14]) and SG morphisms (def. [14]) as category morphisms. We define the domain, co-domain, identity and composition of SGraphs as:

\[
\begin{align*}
domCSG & : GrMorph \rightarrow SGr \\
codCSG & : GrMorph \rightarrow SGr \\
dCSG & : SGr \rightarrow GrMorph \\
idCSG & : SGr \rightarrow GrMorph \\
\end{align*}
\]

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

\[
SGMorph = \{m : GrMorph \mid \exists SG_1, SG_2 : SGr \bullet m \in SG_1 \rightarrow SG_2\}
\]

The category SGraphs is defined as:

\[
SGraphs = (SGr, GrrMorph, domCSG, codCSG, idCSG, \circ_{CSG})
\]

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 \( tr : EsRP_{SG} \rightarrow V \), mapping reference edges attached to proxies to referred nodes.

The base set of local fragments \( Fr_0 \), such that \( F : Fr_0 \), is defined as:

\[
Fr_0 = \{(SG, tr) \mid SG \in SGr \land tr \in EsRP_{SG} \rightarrow V \land EsTy(SG,\{einh\}) < src_{SG} \triangleright Ns_{SG} = \emptyset\}
\]

Above, \(<\) and \(\triangleright\) are domain and range restrictions, respectively. Last conjunct says that proxy nodes \( (Ns_{SG}) \) cannot have supertypes.

Several functions extract the components of a fragment:

\[
sg : Fr_0 \rightarrow SGr \\
tgtr : Fr_0 \rightarrow SGr \\
sg(SG, tr) = SG \\
tgtr(SG, tr) = tr
\]

We introduce several functions and predicates to operate upon \( Fr_0 \): (a) \( withRsG \) gives the fragment’s graph with the proxies connected 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.

\[
\begin{align*}
withRsG & : Fr_0 \rightarrow Gr \\
refsG & : Fr_0 \rightarrow Gr \\
refs & : Fr \rightarrow V \leftrightarrow V \\
acyclicIF & : \emptyset Fr_0 \\
withRsG(SG, tr) & = (Ns_{SG} \cup \text{ran} tr, Es_{SG}, src_{SG}, tgt_{SG} \oplus tr) \\
refsG F & = \text{restrict}(withRsG F, \text{EsRP} F) \\
refs F & = \text{rel}(refsG F) \\
acyclicIF F & = (\langle F \ominus refsF \rangle \ominus \emptyset) \in \text{acyclic} \\
refsOF & : Fr_0 \rightarrow V \leftrightarrow P V \\
nonPRefsOF & : Fr_0 \rightarrow V \rightarrow P V \\
refsOF v = (refsF)^\uparrow \{v\} \\
nonPRefsOF v & = \{v : V \mid v \in \text{refsOF} v \land - v \in Ns_{PF}\}
\end{align*}
\]

Here, \(\oplus\) denotes function overriding.

The actual set of fragments \( Fr \) is defined from \( Fr_0 \) as:

\[
Fr = \{F : Fr_0 \mid (\forall v : Ns_{PF} \bullet \nonPRefsOF v \neq \emptyset) \land \text{acyclicIF} F\}
\]

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) $\leadsto$ gives all the representations of some node; (b) $\prec$ extends $\prec$ of SGs (def. 11) with $\leadsto$; (c) $\text{repsOf}$ indicates the representatives of a given node.

\[
\leadsto : Fr \rightarrow (V \leftrightarrow V) \quad \prec : Fr \rightarrow (V \leftrightarrow V) \quad \text{repsOf} : V \times Fr \rightarrow \mathcal{P} V
\]

\[
\leadsto = \text{refSK} \cup (\text{refSK}) \quad \prec = \text{refP} \cup \leadsto \quad \text{repsOf} v F = \{ v' : N_{SP} | v \leadsto v' \}
\]

**Auxiliary Definitions.** Function $\preceq$ is the reflexive transitive closure of $\prec$ relation for fragments, and $\cup_F$ yields the union of two fragments:

\[
\preceq : Fr \rightarrow (V \leftrightarrow V) \quad \cup_F : Fr \times Fr \rightarrow Fr
\]

\[
\preceq = (\prec)^* \quad \text{F}_1 \cup_F F_2 = (\text{sg} F_1 \cup_{SG} \text{sg} F_2, \text{tgtr} F_1 \cup \text{tgtr} F_2) \Rightarrow \text{disj}(F_1, F_2)
\]

Likewise, SG functions $\text{clan}$, $\text{src}^*$ and $\text{tgt}^*$ are extended for fragments by taking references into account:

\[
\text{clan} : V \times Fr \rightarrow \mathcal{P} V \quad \text{src}^* : Fr \rightarrow (E_L \leftrightarrow V_L) \quad \text{tgt}^* : Fr \rightarrow (E_L \leftrightarrow V_L)
\]

\[
\text{clan}(v, F) = \{ v' : N_{SP} | v' \preceq v \}
\]

\[
\text{src}^* F = \{ (e, v) | e \in E_{AF} \land v \in N_{SP} \land (\exists v_2 : N_{SP} \land v \in \text{clan}(v_2, F) \land (e, v_2) \in \text{src}(F)) \}
\]

\[
\text{tgt}^* F = \{ (e, v) | e \in E_{AF} \land v \in N_{SP} \land (\exists v_2 : N_{SP} \land v \in \text{clan}(v_2, F) \land (e, v_2) \in \text{tgtr}(F)) \}
\]

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

\[
\vdash (\text{sg} F) \preceq \preceq_F \quad \vdash (\text{sg} F) \preceq \preceq_F \quad \vdash \text{clan}(v, (\text{sg} F)) \subseteq \text{clan}(v, F)
\]

\[
\vdash \text{src}^* F \subseteq \text{src}^*_F \quad \vdash \text{tgt}^* F \subseteq \text{tgt}^*_F \quad \vdash \text{tgtr} F \subseteq \text{tgtr} F
\]

\[
\vdash v \in \text{clan}(v, F) \quad \vdash \text{src}^* F \quad \vdash \text{tgtr} F
\]

The following laws are related to fragments:

\[
\vdash x \leadsto x \quad x \leadsto y \quad y \leadsto x \quad \vdash x \preceq x
\]

**Proof.** All laws and healthiness conditions outlined above were proved in the Isabelle proof assistant. $\square$

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

\[
F_1 \cup_F F_2 = (\text{sg} F_1 \cup_{SG} \text{sg} F_2, \text{tgtr} F_1 \cup \text{tgtr} F_2)
\]

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

**Properties.** The following laws are related to fragment union:

\[
F_1 \in Fr; F_2 \in Fr \vdash \text{sg}(F_1 \cup_F F_2) = \text{sg} F_1 \cup_{SG} \text{sg} F_2 \quad F_1 \in Fr; F_2 \in Fr \vdash \text{tgtr}(F_1 \cup_F F_2) = \text{tgtr} F_1 \cup \text{tgtr} F_2
\]

\[
F_1 \in Fr; F_2 \in Fr \vdash (F_1 \cup_F F_2) = (\text{sg} F_1 \cup_{SG} \text{sg} F_2)/\text{tgtr} F_1 \cup \text{tgtr} F_2
\]

**Proof.** All laws given above were proved in the Isabelle proof assistant. $\square$

**Fact 5.** Given fragments $F_1, F_2 : Fr$, we have the following:

- The union of two fragments is inheritance acyclic provided that individually the fragments are acyclic ala[1].

\[
F_1 \in Fr; F_2 \in Fr; \text{disj}(\text{sg} F_1, \text{sg} F_2) \Rightarrow \text{acyclicIF} (F_1 \cup_F F_2) \Rightarrow \text{acyclicIF} F_1 \land \text{acyclicIF} F_2
\]

[1] Acyclicity is expressed in terms of relation transitive closure; it is difficult to establish global acyclicity because the equality $(r \cup s)^+ = r^+ \cup s^+$ does not hold in general; composition acyclicity was demonstrated by exploiting the particularities of Frmenta.
• The union of two fragments is well-formed provided the individual fragments are well-formed also (implies the acyclic result):
\[ F_1 \in Fr; \ F_2 \in Fr; \ \text{disj}(sg \ F_1, sg \ F_2) \vdash (F_1 \cup F_2) \in Fr \iff F_1 \in Fr \land F_2 \in Fr \]

• The inheritance graph of every fragment obtained after resolving the references is acyclic\(^2\):
\[ F \in Fr \vdash \text{acyclic}(\text{replace}(\text{inh}(F) \text{ consSubOfFr} F)) \]

Proof. The three big theorems outlined above were proved with the Isabelle proof assistant. □

Definition 16 (Fragment Morphisms). A fragment morphism \( m : F_1 \rightarrow F_2 \) is a mapping from \( F_1 : Fr \) to fragment \( F_2 : Fr \). It consists of a pair of functions \( m = (f_v, f_e) \) mapping nodes and edges, respectively. The set of fragment morphisms is defined as:
\[ \forall F_1, F_2 : Fr \bullet \ F_1 \rightarrow F_2 = \{ (f_v, f_e) \mid f_v \in N_{F_1} \rightarrow N_{F_2} \land f_e \in E_{F_1} \rightarrow E_{F_2} \land f_v \circ \text{src}_{F_1} \subseteq \text{src}_{F_2} \circ f_e \land f_v \circ \text{tgt}_{F_1} \subseteq \text{tgt}_{F_2} \circ f_e \land f_v \circ \leq_{F_1} \subseteq f_v \circ \leq_{F_2} \circ f_e \} \]

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):
\[ \text{ExtEdgeTy} = \{ \text{impo}, \text{conti} \} \]

A GFG is a pair \( \text{GFG} = (G, et) \), where \( G : Gr \) is a graph (definition 3), and \( et : E_{G} \rightarrow \text{ExtEdgeTy} \) is a colouring function mapping edges to extension edge types. 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:
\[ \text{GFG}_{Gr} = \{ (G, et) \mid G \in Gr \land et \in E_{G} \rightarrow \text{ExtEdgeTy} \land N_{G} \subseteq V_{F} \land E_{G} \subseteq E_{F} \land \text{acyclic}(\text{restrict} (E_{G} \setminus \text{ExtId}_{G})) \} \]

Auxiliary Definitions. We introduce functions to extract components of a GFG:
\[ \text{gr} : \text{GFG}_{Gr} \rightarrow Gr \quad \text{fet} : \text{GFG}_{Gr} \rightarrow E \rightarrow \text{ExtEdgeTy} \]
\[ \text{gr}(G, et) = G \quad \text{fet}(G, et) = et \]
\[ \square \]

Definition 18 (Global Fragment Morphisms). A GFG morphism \( m : \text{GFG}_{1} \rightarrow \text{GFG}_{2} \) defines a specific mapping between GFGs (definition 17) from \( \text{GFG}_{1} : \text{GFG}_{Gr} \) to \( \text{GFG}_{2} : \text{GFG}_{Gr} \). The set of GFG-morphisms is defined as:
\[ \forall G_1, G_2 : Gr; \ et_1, et_2 : E \rightarrow \text{ExtEdgeTy} \bullet \ (G_1, et_1) \rightarrow (G_2, et_2) = G_1 \rightarrow G_2 \land \{ (f_v, f_e) \mid et_2 \circ f_e = et_1 \} \]

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

\(^2\)This shows that it is not possible to have indirect cycles, such as the one of \( F4 \) and \( F5 \) in Fig. 41.
Definition 19 (Fragment to GFG Morphisms). A fragment to GFG morphism \( m : Fr \to GFG \) maps local fragment nodes to GFG nodes. The set of such morphisms is defined as:

\[
F \to GFG = \{ (f_v, f_e) \mid f_v \in N_{SF} \to N_{S_{GFG}} \land f_e \in E_{SF} \to E_{S_{GFG}} \\
\land (f_v, f_e) \in (\text{withRsG} \ F) \to (\text{gr} \ GFG) \\
\land \exists v_f, \exists e_f : N_{S_{GFG}} \cup E_{S_{GFG}} \\
\land f_v \in N_{SF} \land f_e \in E_{SF} \land E_{S_{GFG}} \land f_v \in N_{S_{GFG}} \cup E_{S_{GFG}} \land f_e \in E_{S_{GFG}} \land f_v \in N_{S_{GFG}} \cup E_{S_{GFG}} \land f_e \in E_{S_{GFG}} \land (f_v, f_e) \in (\text{withRsG} \ F) \to (\text{gr} \ GFG)
\]

Above, we say that such a morphism is a graph morphism between the graph obtained by considering the fragment’s graph with the actual references (function withRsG, def. 14) 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 \( CG = (G, e_t) \), comprising a graph \( G : Gr \) (definition 13) and a colouring function \( e_t : E_G \to CGEdgeTy \) mapping edges to cluster edge types (set \( CGEdgeTy \)). The set of valid cluster graphs is defined as:

\[
CGr = \{(G, e_t) \mid G \in Gr \land e_t \in E_G \to CGEdgeTy \\
\land \text{acyclic}(\text{restrict}(G, e_t \sim \{(\text{eimpo}, \text{econta})\} \setminus \text{EsId})) \\
\land \text{rel}(\text{restrict}(G, e_t \sim \{(\text{econta})\} \setminus \text{EsId})) \in \text{forest}\}
\]

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:

\[
gr : CGr \to Gr \\
cety : CGr \to E \to ExtEdgeTy
\]

\[
gr(G, e_t) = G \\
cety(G, e_t) = e_t
\]

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

\[
\forall CG_1, CG_2 : CGr \bullet \\
CG_1 \to CG_2 = \{(f_v, f_e) \mid f_v \in N_{S_{CG_1}} \to N_{S_{CG_2}} \land f_e \in E_{S_{CG_1}} \to E_{S_{CG_2}} \\
\land (f_v, f_e) \in (\text{gr} \ CG_1) \to (\text{gr} \ CG_2) \land (\text{cety} \ CG_2) \circ f_e = \text{cety} \ CG_1 \}
\]

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 \( GFG : GFG_r \) (definition 17) to a cluster graph \( CG : CGr \) (definition 20). The set of such morphisms is defined as:

\[
\forall GFG : GFG_r \quad m : GFG \rightarrow CG = \{(fv, fe) \mid \begin{align*}
fv &\in N_{GFG} \rightarrow N_{CG} \land fe \in E_{GFG} \rightarrow E_{CG} \\
fv &\in GFG \rightarrow CG \land (cvty CG) \circ fe = fety GFG
\end{align*}\}
\]

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 \( GFG : GFG_r \), a \( CG : CGr \), a morphism \( mc : GFG \rightarrow CG \), and a mapping from GFG nodes to fragment definitions \( fd : N_{GFG} \rightarrow Fr \).

The base set of all models, such that \( M \in Mdl_0 \), is defined as:

\[
Mdl_0 = \{(GFG, CG, mc, fd) \mid GFG \in GFG_r \land CG \in CGr \land mc \in GFG \rightarrow CG \land fd \in N_{GFG} \rightarrow Fr\}
\]

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

- \( fg : Mdl_0 \rightarrow GFG \)
- \( cg : Mdl_0 \rightarrow CGr \)
- \( mcg : Mdl_0 \rightarrow GrMorph \)
- \( fdef : Mdl_0 \rightarrow (V \rightarrow Fr) \)
- \( mc = mc \) and \( fdef = fd \)

Function \( UF_{Fs} \) returns the fragment that results from the union of all fragments of a model.

\[
UF_{Fs} : Mdl_0 \rightarrow Fr \\
UF_{Fs} M = UF_{Fs}(\text{run}(fdef M)) \\
UF_{Fs}\{F\} = F \\
UF_{Fs}\{F\} \cup F = F \cup_F (UF_{Fs} Fs)
\]

Function \( from_{V} \) indicates to which fragment a local node belongs to:

\[
UFS0 : Mdl_0 \rightarrow Fr \\
UF_{FS0} M = UF_{FS0}(\text{run}(fdef M)) \\
UF_{FS0}\{F\} = F \\
UF_{FS0}\{F\} \cup F = F \cup_F (UF_{FS0} Fs)
\]

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 \(\text{consFToGFG}\)):

\[
\text{consFToGFG} : V_F \times Mdl_0 \to \text{GrMorph}
\]
\[
\text{consFToGFG}(vf, M) = (vf, fe) \iff \exists F : Fr; \ GFG : GFGr \bullet F = \text{fdef} M \ v_f \land GFG = \text{fg} M
\]
\[
\land fe \in N_{SrF} \to N_{EgG} \land fe \in E_{vF} \to E_{GFG} \land vf \in N_{GFG}
\]
\[
\land (\exists ef : E_{SrF} \bullet \text{srcGFG} ef = \text{tgtGFG} ef = vf \land fe = (vf) \cup \text{consFToGFGrefs}(vf, E_{SrF}, M))
\]

\[
\text{consFToGFGrefs}(vf, \{\}, M) = \{\}
\]
\[
\text{consFToGFGrefs}(vf, \{el\} \cup E_v, M) = fe \iff \\
\exists F : Fr; \ GFG : GFGr \bullet F = \text{fdef} M \ v_f \land GFG = \text{fg} M
\]
\[
\land (\exists ef : E_{SrF} \bullet \text{srcGFG} ef = vf \land \text{tgtGFG} ef = \text{frnt}(\text{tgtF} ef, el, M))
\]
\[
\land fe = \{el \mapsto ef\} \cup \text{consFToGFGrefs}(vf, (E_v, M))
\]

\[
mUMFsToGFG(M) = \text{buildUFsToGFG}(\text{fdef} M, M)
\]
\[
\text{buildUFsToGFG} : (V \to Fr) \times M \to \text{GrMorph}
\]
\[
\text{buildUFsToGFG}((vf \to F), M) = \text{consFToGFG}(vf, M)
\]
\[
\cup_{\text{cat}} \text{buildUFsToGFG}(\text{fdef} M, M)
\]

The set of all models \(Mdl\) is defined as:

\[
Mdl = \{M : Mdl_0 \mid MUMFsToGFG M \in UF_{M} M \to (\text{fg} M)\}
\]
\[
\land (\forall vf_1, vf_2 : Ns(\text{fg} M) \mid vf_1 \neq vf_2 \bullet \text{Ns}(\text{fdef} M \ vf_1) \cap \text{Ns}(\text{fdef} M \ vf_2) = \emptyset \land \text{Es}(\text{fdef} M \ vf_1) \cap \text{Es}(\text{fdef} M \ vf_2) = \emptyset)\}
\]

Above, we say that the morphism obtained from \(mUMFsToGFG\) must be a local fragment to GFG morphism, and that all fragments of a model are disjoint. \(\square\)

### A.9 Category Theory

#### Definition 24 (Pushout).

Given a category \(\mathcal{C} : \text{Cat}\) (definition \([\square]\)) 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 \(D \in \text{obs}_{\mathcal{C}}\),
- 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:

\[
\forall C : \text{Cat} \bullet \forall f, g : \text{morph}_{\mathcal{C}} \bullet
\]
\[
\text{PPO}_{\mathcal{C}} (f, g) = \{(D, f', g') \mid D \in \text{obs}_{\mathcal{C}} \land f' \in \text{morph}_{\mathcal{C}} \land g' \in \text{morph}_{\mathcal{C}} \land \text{dom}_{C} f = \text{dom}_{C} g \land \text{dom}_{C} f' = \text{cod}_{C} g \land \text{dom}_{C} g' = \text{cod}_{C} f' \land f' \circ_C g = g' \circ_C f\}
\]

In the following, we write \(\text{PPO}_{\mathcal{C}}(f, g)\) to mean \(\text{PPO}_{\mathcal{C}}^{f, g}\).

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

\[
\forall C : \text{Cat} \bullet \forall f, g : \text{morph}_{\mathcal{C}} \bullet
\]
\[
\text{PO}_{\mathcal{C}} (f, g) = \{(X, k, h) \mid (X, k, h) \in \text{PPO}_{\mathcal{C}}(f, g) \land \exists x : D \to_C X \bullet x \circ_C f' = k \land x \circ_C g' = h))\}
\]

The following diagram defines a pushout:
Definition 25 (Morphisms of Graphs to Categories). A morphism $G \to C$ from a graph $G : Gr$, such that $G = (V_G, E_G, s, t)$ (definition 3), to a category $C : Cat$, such that $C = (O_C, M_C, dm, cd, id_C, \circ)$ (definition 8) is a pair of functions $(mv, me)$ with $mv : Ns G \to obs_C$ and $me : Es G \to morphs_C$, mapping nodes to objects and edges to morphs, 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:

$$\text{MorphGr} \to \text{Cat} = \{(mv, me) \mid mv \in V \to O \land fe \in E \to M\}$$

The set of valid morphisms between a graph and category $m : G \to C$, such that $m \subseteq \text{MorphGr} \to \text{Cat}$, is defined as:

$$\forall G : Gr; C : Cat \bullet G \to C = \{(mv, me) \mid mv \in Ns G \to obs_C \land me \in Es G \to morphs_C \land mv \circ s = dm \circ me \land mv \circ t = cd \circ me\}$$

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

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

$$mv : \text{MorphGr} \to \text{Cat} \to V \to O$$

$$mv(mv, me) = mv$$

$$me : \text{MorphGr} \to \text{Cat} \to E \to M$$

$$me(me, me) = me$$
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 $\mathcal{C} : \text{Cat}$ (definition 8), a graph $G : \text{Gr}$ (definition 1) and a graph to category morphism $m : G \to \mathcal{C}$ (definition 24).

We define the set of diagrams as:

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

Auxiliary Definitions.

We define functions to yield the components of a diagram:

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

The function $\text{catObs}$ and $\text{catMorphs}$ extract, respectively, the set of underlying category objects and the set of underlying category morphisms from a diagram:

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

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

$$\mathcal{C}(D) = \{ (X, ms) \mid \exists \mathcal{C} : \text{cat } D \bullet X \in \text{obs}_C \land ms \in \mathbb{P}(\text{morphs}_C) \land (\forall m : ms \bullet \text{dom}_C m \in \text{catObs } D \land \text{cod}_C m = X) \}$$

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

$$\forall D : \text{Diag}; \ X : O; \ ms : \mathbb{P} M$$
$$\exists (X, ms) \in \text{ValCC } D \iff (X, ms) \in \mathcal{C}(D)$$
$$\land (\forall m : \text{catMorphs } D \bullet \exists f, g : ms; \ C : \text{Cat} \bullet \text{C} = \text{cat } D \land f \in \text{dom}_C m \to_C X \land g \in \text{cod}_C m \to_C X \land g \circ_C m = f)$$
A colimit is then 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:

$$\forall D : \text{Diag} \bullet \text{colimit } D = (q, X : O; ms : \mathbb{P} M | (X, ms) \in \text{ValCC } D$$

$$\wedge (\exists C : \text{Cat} \bullet C = \text{cat } D$$

$$\wedge (\forall X' : \text{obs}_C; ms' : \text{morphs}_C | X \neq X' \wedge (X', ms') \in \text{ValCC } D \bullet$$

$$\exists k : X \to X' \bullet \forall f : ms; g : ms' | \text{dom}_C f = \text{dom}_C g \bullet k \circ_C f = g$$

That is, the underlying diagram commutes:

That is, the underlying diagram commutes:

\[
\begin{array}{c}
X \xrightarrow{k} X' \\
\downarrow f \quad \downarrow g \\
D \xleftarrow{k} D'
\end{array}
\]

\[\square\]

A.10 Colimit composition

Definition 28 (Fragment Composition Diagram). The composition diagram of a fragment is defined through function $\text{compDiag}$. 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 $\text{buildStartDiag}$).

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

3. It adds to the diagram all interface graphs and corresponding morphisms (function $\text{diagMorphisms}$). This involves building the interface graph and morphisms corresponding to merges (function $\text{diagMerges}$) and references (function $\text{diagRefs}$); 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:

$id_G : \text{Gr} \to \text{GrMorph}$

$id_G : G = GM \iff GM \in G \to G \wedge GM = (\text{id(nodesG)}, \text{id(edgesG)})$

The actual category of graphs is defined as:

$\text{GrCat} : \text{Cat}$

$\text{GrCat} = (\text{Gr}, \text{GrMorph}, \text{id}_G, \circ_G)$

Function $\text{compDiag}$ is defined as:

$\text{compDiag} : V_F \times Mdl \to \text{Diag}$

$\text{compDiag}(vf, M) = D' \equiv \exists D_0, D_1, D_2 : \text{Diag} \bullet$

$\text{buildStartDiag}(vf, M) = D_0$

$\wedge \text{diagDepNodes}(\text{importsOf}(vf, (m_{-fG} M)), M, D_0) = D_1$

$\wedge \text{diagDepNodes}(\text{continuesOf}(vf, (m_{-fG} M)), M, D_1) = D_2$

$\wedge \text{diagMorphisms}(vf, M, D_2) = D'$
Function \textit{buildStartDiag} is defined as:

\begin{align*}
\text{buildStartDiag} : V_F \times Mdl &\rightarrow \text{Diag} \\
\text{buildStartDiag}(vf, M) &= \text{addNodeToDiag}(vf, \text{srcGr((m\_fdef M) vf)}, \text{emptyDiag GrCat})
\end{align*}

Function \textit{diagDepNodes} is defined as:

\begin{align*}
\text{diagDepNodes} : \mathbb{P} V_F \times Mdl &\times \text{Diag} \rightarrow \text{Diag} \\
\text{diagDepNodes}(\emptyset, M, D) &= D \\
\text{diagDepNodes}(\{vf_1\} \cup vfs, M, D) &= D' \iff \\
&\exists D_0, D_1, D_2 : \text{Diag} \bullet \\
&\text{addNodeToDiag}(vf_1, gr ((m\_fdef M) vf_1), D) = D_0 \\
&\land \text{diagDepNodes(\text{importsOf}(vf_1, (m\_fg M)), M, D_0)} = D_1 \\
&\land \text{diagDepNodes(\text{continuationsOf}(vf_1, (m\_fg M)), M, D_1)} = D_2 \\
&\land \text{diagDepNodes(vfs, M, D_2)} = D'
\end{align*}

Function \textit{diagMorphisms} is defined as:

\begin{align*}
\text{diagMorphisms} : V_F &\times Mdl \times \text{Diag} \rightarrow \text{Diag} \\
\text{diagMorphisms}_{\emptyset} : (V_F \times Mdl \times \text{Diag} \times \mathbb{P} V_F) &\rightarrow \text{Diag} \times \mathbb{P} V_F \\
\text{diagMorphisms}(vf, M, D) &= \text{diagMorphisms}_{\emptyset}(vf, M, D, \emptyset) \\
\text{diagMorphisms}(vf, M, D, vfs) &= (D', \text{vfs}) \iff \\
&\exists F : Fr; D_1, D_2 : \text{Diag} \bullet F = ((m\_fdef M) vf) \\
&\land \text{diagRefs(vf, \text{importsOf}(vf, m\_fg M) \cup \text{continuationsOf}(vf, m\_fg M), GE, D)} = D_1 \\
&\land \text{diagMorphisms}_{\emptyset}(\text{importsof}(vf, m\_fg M) \\
&\cup \text{continuationsOf}(vf, m\_fg M), GE, D_1, \text{vfs} \cup \{vf\}) = (D', \text{vfs})' \\
\text{diagMorphismsSet} : \mathbb{P} V_F &\times Mdl \times \text{Diag} \times \mathbb{P} V_F \\
\text{diagMorphismsSet}(vf_1, M, D, P) &= (D', P') \iff \text{vf_1} \in P \\
\text{diagMorphismsSet}(vf_1, M, D, P) &= \text{diagMorphismsSet(vfs, M, D, P)} \iff \text{vf_1} \in P \\
&\land \text{diagMorphisms}_{\emptyset}(vf_1, M, D, P) = (D'', P'') \\
&\land \text{diagMorphismsSet(vfs, M, D', P''')} = (D', P''')
\end{align*}

Function \textit{diagRefs} is defined as:

\begin{align*}
\text{HasImpRefs} : \mathbb{P}(V_F \times V_F \times Mdl) &\rightarrow \mathbb{P} F_1, F_2 : Fr \bullet \\
F_1 = (m\_fdef M) vf_1 &\land F_2 = (m\_fdef M) vf_2 \land (\text{refs F_1} \regnode \text{nodes F_2}) \neq \emptyset
\end{align*}

\begin{align*}
\text{diagRefs} : V_F &\times \mathbb{P} V_F \times Mdl \times \text{Diag} \rightarrow \text{Diag} \\
\text{diagRefs}(vf_1, \emptyset, M, D) &= D \\
\text{diagRefs}(vf_1, \{vf_2\} \cup vfs, M, D) &= \text{diagRefs}(vf_1, vfs, M, D) \iff \neg \text{HasImpRefs(vf_1, vfs, M, D)} \\
\text{diagRefs}(vf_1, \{vf_2\} \cup vfs, M, D) &= D' \iff \text{HasImpRefs(vf_1, vfs, M, D)} \bullet \\
&\exists F_1, F_2 : Fr; GI : Gr; vf : VF; m_1, m_2 : GrMorph; e_1, e_2 : E; D_0, D_1, D_2 : \text{Diag} \\
&F_1 = (m\_fdef M) vf_1 \land F_2 = (m\_fdef M) vf_2 \\
&\land GI = (\text{dom(\text{refs F_1}) \regnode \text{nodes F_2}}), \emptyset, \emptyset) \\
&\land m_1 \in GI \rightarrow \text{srcGr F_1} \land m_1 = (\text{Id(\text{dom(\text{refs F_1}) \regnode \text{nodes F_2}})}, \emptyset) \\
&\land m_2 \in GI \rightarrow \text{srcGr F_2} \land m_2 = ((\text{refs F_1} \regnode \text{nodes F_2}), \emptyset) \\
&\land \neg \exists \text{vf_1} \in \text{nodes (gr D)} \land \text{addNodeToDiag(vf_1, GI, D)} = D_0 \\
&\land \neg \{e_1, e_2\} \subseteq \text{edges(gr D_0) \land addEdgeToDiag(e_1 vf_1, vf_1, m_1, D_0)} = D_1 \\
&\land \text{addEdgeToDiag(e_2, vf_1, vfs, D_2)} = D_2 \land D' = \text{diagRefs(vf_1, vfs, GE, D_2)}
\end{align*}
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 11) 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 11).

The set of all type SGs is defined as:

$$TySG = \{(SG, iet) \mid SG \in SGr \land iet \in EsA(SG) \rightarrow SGET\}$$

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

- $sgr : TySGr \rightarrow SGr$
- $sgr(SG, iet) = SG$

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

$$EsOfTy : TySGr \times SGETy \rightarrow \mathcal{P} E$$

$$EsOfTy((SG, iet), ety) = iet^{-1}(\{ety\})$$

Next functions extract functions of SGs to type SGs:

- $NsA : TySGr \rightarrow \mathcal{P} V$
- $NsA(TSG) = N_{sA}(sgr(TSG))$
- $Ns : TySGr \rightarrow \mathcal{P} V$
- $Ns(TSG) = N_{s}(sgr(TSG))$
- $EsC : TySGr \rightarrow \mathcal{P} V$
- $EsC(TSG) = E_{sC}(sgr(TSG))$
- $srcm : TySGr \rightarrow \mathcal{P} V$
- $srcm(TSG) = srcm(sgr(TSG))$
- $tgtn : TySGr \rightarrow \mathcal{P} V$
- $tgtn(TSG) = tgm(tgr(TSG))$

**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 $type : SG \rightarrow (sgr TSG)$ 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:

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

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

- $sgr : SGTy \rightarrow SGr$
- $sgr(SG, TSG, type) = SG$
- $tgyn : SGTy \rightarrow TySGr$
- $tgyn(SG, TSG, type) = TSG$
- $tgymorph : SGTy \rightarrow SGMor$
- $tgymorph(SG, TSG, type) = type$
We extend the functions $Ns$, $Es$, $src$ and $tgt$ of SGs by considering that they yield the nodes and edges of the source SG:

$$Ns : SGTy \to P V$$
$$Ns(SGT) = Ns(srcGr SGT)$$

$$Es : SGTy \to P E$$
$$Es(SGT) = Es(srcGr SGT)$$

$$src : SGTy \to (E \to V)$$
$$src(SGT) = src(srcGr SGT)$$

$$tgt : SGTy \to (E \to V)$$
$$tgt(SGT) = tgt(srcGr SGT)$$

**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:

$$instanceEdgeTypesOk(SG, TSG, type) \iff \text{iet}y_{TSG} \circ (fE type) = \text{iet}y_{SG}$$

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

$$abstractNoDirectInstances(SG, TSG, type) \iff (fV type) \setminus \{\text{nodes}_A(TSG)\} = \emptyset$$

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:

$$containmentNoSharing(SG, TSG, type) \iff (fE type) \setminus \{\text{edge}_A(TSG)\} \subset \text{tgt}^*(SG) \in \text{EinjrelV}$$

This requires that the target function of instances of containment edges is injective (set $\text{injrel}$ of definition §), 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:

$$\forall te : \text{edges}_A(TSG) \bullet$$

$$\exists r : V \leftrightarrow V \bullet r = \text{rel}(\text{restrict(gr SG, (fE type) \setminus \{\{te\}\}))}$$

$$\land \forall v : \text{dom } r \bullet \text{multOk}(r \setminus \{v\}, (srcm TSG)te)$$

$$\land \forall v : \text{ran } r \bullet \text{multOk}(r \setminus \{v\}, (tgtm TSG)te)$$

This predicate obtains the relation that is induced by the edges that are instances of the association edges of the graph (function $\text{rel}$).

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

$$\text{instContainmentAcyclic}(SG, TSG, type) \iff \text{acyclicGr}(\text{restrict(gr SG, (fE type) \setminus \{edges_c(TSG)\}))}$$

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 $\text{acyclic}$ (definition §)
There is a summary predicate that checks that typed structural graph are conformant:

\[
\text{isConformable}(SGT) \iff \text{abstractNoDirectInstances}(SGT) \wedge \text{containmentNoSharing}(SGT) \\
\wedge \text{instMultsOk}(SGT) \wedge \text{instContainmentAcyclic}(SGT)
\]

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

\[
\text{SGTyConf} = \{ SGT : \text{SGTy} \mid \text{isConformable}(SGT) \}
\]

\[\square\]

### A.12 Typed Fragments

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

\[
\text{TFr} = \{(F, \text{iet}) \mid F \in \text{Fr} \wedge \text{iet} \in \text{Es}_A F \rightarrow \text{SGET}\}
\]

**Auxiliary Definitions.** Functions \(\text{Ns}\) and \(\text{Es}\) of \(\text{Fr}\) are extended to \(\text{TFr}\). Functions \(\text{fr}\) and \(\text{iety}\) yield the components of a \(\text{TFr}\):

\[
\begin{align*}
\text{fr} : \text{TFr} &\rightarrow \text{Fr} \\
\text{iety} : \text{TFr} &\rightarrow \text{E} \rightarrow \text{SGET} \\
\text{fr}(F, \text{iet}) &= F \\
\text{iety}(F, \text{iet}) &= \text{iet} \\
\text{fr} \cup_{\text{TFr}} : \text{TFr} \times \text{TFr} &\rightarrow \text{TFr} \\
\text{TFr}_1 \cup_{\text{TFr}} \text{TFr}_2 &= (\text{fr} \text{TFr}_1 \cup_{\text{Fr}} \text{fr} \text{TFr}_2, \text{iety} \text{TFr}_1 \cup \text{iety} \text{TFr}_2)
\end{align*}
\]

\[\square\]

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

\[
\text{FrTy} = \{(F, TF, \text{type}) \mid F \in \text{Fr} \wedge TF \in \text{TFr} \wedge \text{type} \in F \rightarrow TF\}
\]

\[\square\]

**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:

\[
\text{instanceEdgeTypesOk}(F, TF, \text{type}) \iff \text{iety}_{TF} \circ (\text{jet} \text{type}) = \text{et}_{TF}
\]

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

\[
\text{abstractNoDirectInstances}(F, TF, \text{type}) \iff (\text{jet}_{\text{type}}) \sim (\text{NsAbst} TF) = \emptyset
\]

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

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

51
Above, we get all representatives of some abstract node (function \(\text{reps}\)).

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

\[
\text{containmentNoSharing}(F, TF, \text{type}) \equiv \langle ([E \text{type}] \sim \{E \text{Ty}(TF, \{e\text{comp}\})\}) \triangleq \text{tgt}^* F \in \text{injrel} V
\]

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

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

\[
\text{instMultsOk}(F, TF, \text{type}) \equiv \forall te : \text{EsA} TF \bullet
\exists r : V \leftrightarrow V \bullet r = \text{rel}(\text{restrict}(\text{gr} F, (f_E \text{type}) \sim \{\{te\}\})))
\land \forall v : \text{dom} r \bullet \text{multOk}(r \downarrow \text{repsOf}(v, F), \{\text{srcm} (\text{tg}(\text{tg}(\text{tg}(\text{tg}(\{te\}) F)))\})
\land \forall v : \text{ran} r \bullet \text{multOk}(r \downarrow \text{repsOf}(v, F), \{\text{tg}(\text{tg}(\text{tg}(\text{tg}(\{te\}) F)))\})
\]

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

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

\[
\text{instContainmentAcyclic}(F, TF, \text{type}) \equiv \text{rel}(\text{restrict}(\text{gr} SG, (f_E \text{type}) \sim \{E \text{Ty}(TF, \{e\text{comp}\})\}) \in \text{forest}
\]

A summary predicate checks that typed fragments are conformant:

\[
\text{isConformable} FT \equiv \text{instanceEdgeTypesOk} FT \land \text{abstractNoDirectInstances} FT
\land \text{containmentNoSharing} FT \land \text{instMultsOk} FT \land \text{instContainmentAcyclic} FT
\]

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

\[
\text{FrTyConf} = \{ FT : \text{FrTy} | \text{isConformable} FT \}
\]

\[\square\]

### A.13 Typed Models

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

The set of base type models \(\text{TMdl}\), such that \(\text{TM} \in \text{TMdl}\), is defined as:

\[
\text{TMdl} = \{(\text{GFG}, \text{CG}, mc, fd, \text{SGFG}, \text{SCG}, sc, sf) | \text{GFG} \in \text{GFGr} \land \text{CG} \in \text{CGr}
\land mc \in \text{GFG} \rightarrow \text{CG} \land fd \in \text{NSGFG} \rightarrow \text{TFr} \land \text{SGFG} \in \text{GFGr} \land \text{SCG} \in \text{CGr}
\land sc \in \text{SGFG} \rightarrow \text{SCG} \land sf \in \text{GrMorph}\}
\]

We extend the functions to extract the different components of a model (set \(\text{Mdl}\), def 23) to
typed models \((\text{TMdl}_0)\). We define further functions to yield components of \(\text{TMdl}_0\):

\[
\begin{align*}
\text{sgfg} : \text{TMdl}_0 &\to \text{GFG} \\
\text{sgfg}(\text{GFG}, \text{CG}, \text{mc}, \text{fd}, \text{SGFG}, \text{SCG}, \text{sc}, \text{sf}) &\equiv \text{GFG} \\
\text{scg} : \text{TMdl}_0 &\to \text{CG} \\
\text{scg}(\text{GFG}, \text{CG}, \text{mc}, \text{fd}, \text{SGFG}, \text{SCG}, \text{sc}, \text{sf}) &\equiv \text{SCG} \\
\text{smcg} : \text{TMdl}_0 &\to \text{GrMorph} \\
\text{smcg}(\text{GFG}, \text{CG}, \text{mc}, \text{fd}, \text{SGFG}, \text{SCG}, \text{sc}, \text{sf}) &\equiv \text{sc} \\
\text{smfg} : \text{TMdl}_0 &\to (V \to \text{Fr}) \\
\text{smfg}(\text{GFG}, \text{CG}, \text{mc}, \text{fd}, \text{SGFG}, \text{SCG}, \text{sc}, \text{sf}) &\equiv \text{sf}
\end{align*}
\]

Function \(\text{UTFs}\) returns the fragment that results from the union of all fragments of a model.

\(\text{mUTFsToGFG}\) builds a morphism from the union of all typed fragments of a model to the given model’s \(\text{GFG}\).

Function \(\text{UTFs}\) returns the fragment that results from the union of all fragments of a model.

\(\text{from}_\text{V}\) indicates to which fragment a local node belongs to:

\[
\text{UTFs} : \text{TMdl}_0 \to \text{Fr} \\
\text{UTFs} M = \text{UTFs}(\text{ran}(\text{fdef} M)) \\
\text{UTFs}(\text{TF}) = \text{TF} \\
\text{UTFs}(\text{TF}) \cup \text{UTFs} = \text{TF} \cup \text{UTFs}(\text{UTFs} \text{TFs})
\]

Function \(\text{mUTFsToGFG}\) builds a morphism from the union of all type fragments of a type model to the given model’s \(\text{GFG}\), which involves other auxiliary functions (such as \(\text{consTFToGFG}\)):

\[
\begin{align*}
\text{consTFToGFG} : V_F \times \text{TMdl}_0 &\to \text{GrMorph} \\
\text{consTFToGFG}(\text{vf}, \text{TM}) &\equiv (\text{vf}, \text{fc}) \in \exists \text{TF} : \text{TF} \mid \text{GFG} : \text{GFG} \bullet \\
\text{TF} &\equiv \text{fdef} \text{TM} \text{vf} \land \text{GFG} = \text{fdef} \text{TM} \text{vf} \in N_{\text{TF}} \to N_{\text{GFG}} \\
\land \text{fc} &\in \text{Es}_{\text{TF}} \to \text{Es}_{\text{GFG}} \land \text{vf} \in N_{\text{GFG}} \\
\land (\exists \text{ef} : \text{Es}_{\text{GFG}} \bullet \text{src}_{\text{GFG}} \text{ef} = \text{tg}_{\text{GFG}} \text{ef} = \text{vf} \land \text{fc} = N_{\text{TF}} \times \{\text{vf}\} \\
\land \text{fc} &\equiv (\text{Es}_{\text{TF}} \setminus \text{Es}_{\text{TF}}) \times (\{\text{vf}\}) \cup \text{consTFToGFGRef}(\text{vf}, \text{Es}_{\text{TF}} \text{TM})
\end{align*}
\]

The set of all type models \(\text{TMdl}\), such that \(\text{TM} \in \text{TMdl}\), is defined as:

\[
\text{TMdl} = \{\text{TM} : \text{TMdl}_0 \mid \text{smfg} \text{TM} \in (\text{UTFs} \text{TM}) \rightarrow \text{sgfg} \text{TM} \\
\land m\text{UTFsToGFG} \text{TM} \in \text{UTFs} \text{TM} \rightarrow (\text{fdef} \text{TM}) \\
\land (\forall \text{vf}_1, \text{vf}_2 : \text{Ns}(\text{fdef} \text{TM}) \mid \text{vf}_1 \neq \text{vf}_2 \bullet \\
\land \text{Ns}(\text{fdef} \text{TM} \text{vf}_1) \cap \text{Ns}(\text{fdef} \text{TM} \text{vf}_2) = \emptyset \land \text{Es}(\text{fdef} \text{TM} \text{vf}_1) \cap \text{Es}(\text{fdef} \text{TM} \text{vf}_2) = \emptyset)\}
\]
**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:

$$FSs = \{ (CG, GFG, sc, sf) \mid CG \in CGr \land GFG \in GFGr \land sc \in GFG \rightarrow CG \land sf \in GrMorph \}$$

**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:

$$TFSMdl = \{ (TM, FS) \mid TM \in TMdl \land FS \in FSs \land mgfg_{FS} \in UTFs TM \rightarrow gf_{FS} \}$$

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) $smc : cg_M \rightarrow scg_{TM}$ maps $M$’s CG into the FS’s CG of $TM$, (ii) $smf : gf_M \rightarrow sgfg_{TM}$ maps GFG of $M$ into the FS’s GFG of $TM$, and (iii) $ty : UFs M \rightarrow UFs TM$ maps union of model fragments of $M$ into its $TM$ counter-part. Set of typed models is defined as:

$$MdlTy = \{ (M, TM, scg, sgfg, ty) \mid M \in Mdl \land TM \in TFSMdl \land scg \in cg_M \rightarrow scg_{TM} \land sgfg \in gf_M \rightarrow sgfg_{TM} \land (UFs M, UTFs TM, ty) \in FrTyConf \land sgfg \circ UMTGFG M = msf_{TM} \circ ty \land scg \circ mcg_M = ms_{TM} \circ sgfg \}$$

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).

54
Appendix B

Z Specification of FRAGMENTA

B.1 Generics

\textit{section} Fragmenta_Generics \textit{parents} standard_toolkit

\begin{align*}
\text{acyclic}[X] &= \{ r : X \leftrightarrow X \mid r^+ \land \text{id} X = \varnothing \} \\
\text{connected}[X] &= \{ r : X \leftrightarrow X \mid \forall x : \text{dom} r; \ y : \text{ran} r \cdot x \rightarrow y \in r^+ \} \\
\text{tree}[X] &= \{ r : X \leftrightarrow X \mid r \in \text{acyclic} \land r \in X \rightarrow X \} \\
\text{forest}[X] &= \{ r : X \leftrightarrow X \mid r \in \text{acyclic} \land (\forall s : X \leftrightarrow X \mid s \subseteq r \land s \in \text{connected} \cdot s \in \text{tree}) \} \\
\text{injref}[X, Y] &= \{ r : X \leftrightarrow Y \mid (\forall x : X; \ y_1, y_2 : Y \cdot (x, y_1) \in r \land (x, y_2) \in r \Rightarrow y_1 = y_2) \}
\end{align*}

B.2 Graphs

\textit{section} Fragmenta_Graphs \textit{parents} standard_toolkit, Fragmenta_Generics

\[ [V, E] \]
\[ G_r = \{ vs : \text{P} V; \ es : \text{P} E; \ s, t : E \rightarrow V \mid s \in es \rightarrow vs \land t \in es \rightarrow vs \} \]

\begin{align*}
\forall vs : \text{P} V; \ es : \text{P} E; \ s : E \rightarrow V; \ t : E \rightarrow V \cdot N_s(vs, es, s, t) &= vs \\
\forall vs : \text{P} V; \ es : \text{P} E; \ s : E \rightarrow V; \ t : E \rightarrow V \cdot E_s(vs, es, s, t) &= es \\
\forall vs : \text{P} V; \ es : \text{P} E; \ s : E \rightarrow V; \ t : E \rightarrow V \cdot \text{src}(vs, es, s, t) &= s \\
\forall vs : \text{P} V; \ es : \text{P} E; \ s : E \rightarrow V; \ t : E \rightarrow V \cdot \text{tgt}(vs, es, s, t) &= t \\
\forall vs : \text{P} V; \ es : \text{P} E; \ s : E \rightarrow V; \ t : E \rightarrow V \cdot \text{EsId}(vs, es, s, t) &= \{ e : es \mid s e = t e \}
\end{align*}
relation(adjacent _) 

$p_{\text{adjacent}} : \mathbb{P}(V \times V \times \text{Gr})$

$\forall v_1, v_2 : V; \; G : \text{Gr} \bullet (\text{adjacent}(v_1, v_2, G)) \Leftrightarrow (\exists e : \text{Es} G \bullet \text{src} G e = v_1 \land \text{tgt} G e = v_2)$

$\text{successors} : V \times \text{Gr} \to \mathbb{P} V$

$\forall v : V; \; G : \text{Gr} \bullet \text{successors}(v, G) = \{v_1 : \text{Ns} G | \text{adjacent}(v, v_1, G)\}$

$\text{rel} : \text{Gr} \to V \leftrightarrow V$

$\forall G : \text{Gr} \bullet \text{rel} G = \{v_1, v_2 : \text{Ns} G | \text{adjacent}(v_1, v_2, G)\}$

relation(acyclicG _) 

$p_{\text{acyclic}} : \mathbb{P} \text{Gr}$

$\forall G : \text{Gr} \bullet (\text{acyclic}(G) G) \Leftrightarrow \text{rel} G \in \text{acyclic}$

$\text{restrict} : \text{Gr} \times \mathbb{P} E \to \text{Gr}$

$\forall G : \text{Gr}; \; Er : \mathbb{P} E \bullet \text{restrict}(G, Er) = (\text{Ns} G, \text{Es} G \cap \text{Er}, \text{Er} \cup \text{src} G, \text{Er} \cup \text{tgt} G)$

relation(disjGs _) 

$p_{\text{disj}} : \mathbb{P} (\text{Gr} \times \text{Gr})$

$\forall G_1, G_2 : \text{Gr} \bullet (\text{disjGs}(G_1, G_2)) \Leftrightarrow \text{Ns} G_1 \cap \text{Ns} G_2 = \emptyset \land \text{Es} G_1 \cap \text{Es} G_2 = \emptyset$

function 10 leftrailoc( _, _ )

$\text{leftassoc} : \text{Gr} \times \text{Gr} \to \text{Gr}$

$\forall G_1, G_2 : \text{Gr} \bullet G_1 \cup G_2 = (\text{Ns} G_1 \cup \text{Ns} G_2, \text{Es} G_1 \cup \text{Es} G_2, \text{src} G_1 \cup \text{src} G_2, \text{tgt} G_1 \cup \text{tgt} G_2)$

$\Leftrightarrow (\text{disjGs}(G_1, G_2))$
replaceGfun : \( (E \to V) \to (V \to V) \to (E \to V) \)

\[ \forall f : E \to V, \ \text{sub} : V \to V \cdot \]
\[ \text{replaceGfun } f \ \text{sub} = f \uplus \{ e : \text{dom } f ; \ v : V \mid (f e) \in \text{dom sub} \land \text{sub } (f e) = v \} \]

replaceG : \( \text{Gr} \to (V \to V) \to \text{Gr} \)

\[ \forall G : \text{Gr}; \ \text{sub} : V \to V \cdot \text{replaceG } G \ \text{sub} = (\text{Ns } G \setminus \text{dom sub} \cup \text{ran } (\text{Ns } G \circ \text{sub}), \text{Es } G, \]
\[ \text{replaceGfun } (\text{src } G) \ \text{sub}, \text{replaceGfun } (\text{tgt } G) \ \text{sub}) \]

\[ \text{GrMorph } = = (V \to V) \times (E \to E) \]

\[ fV : \text{GrMorph} \to V \to V \]
\[ fE : \text{GrMorph} \to E \to E \]

\[ \forall fe : V \to V; \ fe : E \to E \cdot fV(fe, fe) = fe \]
\[ \forall fe : V \to V; \ fe : E \to E \cdot fE(fe, fe) = fe \]

\text{function 10 leftassoc } (~\circ_{GM} ~) \]

\[ ~\circ_{GM} ~ : \text{GrMorph} \times \text{GrMorph} \to \text{GrMorph} \]

\[ \forall GM_1, GM_2 : \text{GrMorph}; \]
\[ GM_1 \circ_{GM} GM_2 = (fV GM_1 \circ fV GM_2, fE GM_1 \circ fE GM_2) \iff \]
\[ fV GM_1 \cap fV GM_2 = \emptyset \land fE GM_1 \cap fE GM_2 = \emptyset \]

\[ \text{morphG} : \text{Gr} \times \text{Gr} \to \text{GrMorph} \]

\[ \forall G_1, G_2 : \text{Gr}; \ \text{morphG} (G_1, G_2) = \{ fe : \text{Ns } G_1 \to \text{Ns } G_2; \ fe : \text{Es } G_1 \to \text{Es } G_2 \mid \]
\[ \text{src } G_2 \circ fe = fe \circ \text{src } G_1 \land \text{tgt } G_2 \circ fe = fe \circ \text{tgt } G_1 \} \]

\text{function 10 leftassoc } (~\circ_G ~) \]

\[ ~\circ_G ~ : \text{GrMorph} \times \text{GrMorph} \to \text{GrMorph} \]

\[ \forall m_1, m_2 : \text{GrMorph}; \ m_1 \circ_G m_2 = (fV m_1 \circ fV m_2, fE m_1 \circ fE m_2) \]

57
B.3 Category Theory

section Fragmenta_CatTheory parents standard_toolkit, Fragmenta_Graphs

\([O, M]\)

\(\text{Cat0} = \{ os : \mathbb{P} O; ms : \mathbb{P} M; \text{dm}, cd : M \to O; \text{idn} : O \to M; \text{cmp} : M \times M \to M \mid \text{\text{dm}} \in ms \to os \land \text{cd} \in ms \to os \land \text{idn} \in os \to ms \land \text{cmp} \in ms \times ms \to ms\}\)

\begin{align*}
\text{obs} & : \text{Cat0} \to \mathbb{P} O \\
\text{morphs} & : \text{Cat0} \to \mathbb{P} M \\
\text{domC}, \text{codC} & : \text{Cat0} \to M \to O \\
\text{idC} & : \text{Cat0} \to O \to M \\
\text{comp} & : \text{Cat0} \to M \times M \to M
\end{align*}

\(\forall \text{os} : \mathbb{P} O; \text{ms} : \mathbb{P} M; \text{dm}, \text{cd} : M \to O; \text{idn} : O \to M; \text{cmp} : M \times M \to M \bullet \)

\begin{align*}
\text{obs}(\text{os}, \text{ms}, \text{dm}, \text{cd}, \text{idn}, \text{cmp}) & = \text{os} \\
\text{morphs}(\text{os}, \text{ms}, \text{dm}, \text{cd}, \text{idn}, \text{cmp}) & = \text{ms} \\
\text{domC}(\text{os}, \text{ms}, \text{dm}, \text{cd}, \text{idn}, \text{cmp}) & = \text{dm} \\
\text{codC}(\text{os}, \text{ms}, \text{dm}, \text{cd}, \text{idn}, \text{cmp}) & = \text{cd} \\
\text{idC}(\text{os}, \text{ms}, \text{dm}, \text{cd}, \text{idn}, \text{cmp}) & = \text{idn} \\
\text{comp}(\text{os}, \text{ms}, \text{dm}, \text{cd}, \text{idn}, \text{cmp}) & = \text{cmp}
\end{align*}

\(\text{CatMorphs} : \text{Cat0} \to (O \times O) \to \mathbb{P} M\)

\(\forall C : \text{Cat0}; A, B : O \bullet \)

\(\text{CatMorphs}(A, B) = \{ m : \text{morphs} C \mid \text{domC} C m = A \land \text{codC} C m = B\}\)

\(\text{Cat} = = \{ C : \text{Cat0} \mid (\forall A : \text{obs} C \bullet \text{idC} C A \in \text{CatMorphs}(A, A)) \land (\forall f, g : \text{morphs} C \mid \text{domC} C g = \text{codC} C f \bullet \text{comp}(C, g, f) \in \text{CatMorphs} C((\text{domC} C f), (\text{codC} C g))) \land (\forall A, B, C_1, D : \text{obs} C \bullet (\forall f : \text{CatMorphs}(A, B); g : \text{CatMorphs}(B, C_1); h : \text{CatMorphs}(C_1, D) \bullet \text{comp}(h, (\text{comp}(C, g, f))) = \text{comp}(C((\text{comp}(h, g)), f))))) \land (\forall A, B : \text{obs} C \bullet (\forall f : \text{CatMorphs}(A, B) \bullet (\text{comp}(C((\text{idC} C B), f)) = f \land \text{comp}(C(f, (\text{idC} C A)) = f)))\}

\(\text{MorphG2C} = = (V \to O) \times (E \to M)\)
\[ \forall \text{mv} : V \to O; \ \text{me} : E \to M \bullet \text{mv}(\text{me}) = \text{me} \]
\[ \forall \text{mv} : V \to O; \ \text{me} : E \to M \bullet \text{me}(\text{mv}) = \text{me} \]

\[ \text{morphGC} ::= (\lambda G : \text{Gr}; \ C : \text{Cat} \bullet \{\text{mv} : \text{Ns} G \to \text{obs} C; \ \text{me} : \text{Es} G \to \text{morphs} C \mid \text{mv} \circ \text{src} G = \text{dom} C \circ \text{me} \land \text{me} \circ \text{tgt} G = \text{cod} C \circ \text{me}\}) \]

\[ PPO ::= (\lambda C : \text{Cat} \bullet (\lambda f, g : \text{morphs} C \mid \text{dom} C f = \text{dom} C g \bullet \\
\{D : \text{obs} C; f', g' : \text{morphs} C \mid f' \in \text{CatMorphs} C((\text{cod} C g), D) \land \\
g' \in \text{CatMorphs} C((\text{cod} C f), D) \land \text{comp} C(f', g) = \text{comp} C(g', f))\}) \]

\[ PO ::= (\lambda C : \text{Cat} \bullet (\lambda f, g : \text{morphs} C \bullet \\
(\mu D : \text{obs} C; f', g' : \text{morphs} C \mid (D, f', g') \in PPO C(f, g) \\
\land (\forall X : \text{obs} C; h, k : \text{morphs} C \bullet ((X, h, k) \in PPO C(f, g) \\
\land (\exists x : \text{CatMorphs} C(D, X) \bullet (\text{comp} C(x, f') = k \land \text{comp} C(x, g') = h))))))) \]

\[ \text{Diag} ::= \{C : \text{Cat}; \ G : \text{Gr}; \ m : \text{Morph} G2C \mid m \in \text{morphGC}(G, C)\} \]

\[ \text{grD} : \text{Diag} \to \text{Gr} \]
\[ \text{cat} : \text{Diag} \to \text{Cat} \]
\[ \text{morphD} : \text{Diag} \to \text{Morph} G2C \]
\[ \text{NsD} : \text{Diag} \to \mathcal{P} V \]
\[ \text{obsD} : \text{Diag} \to \mathcal{P} O \]
\[ \text{morphsD} : \text{Diag} \to \mathcal{P} M \]

\[ \forall C : \text{Cat}; \ G : \text{Gr}; \ m : \text{Morph} G2C \bullet \text{grD}(C, G, m) = G \]
\[ \forall C : \text{Cat}; \ G : \text{Gr}; \ m : \text{Morph} G2C \bullet \text{cat}(C, G, m) = C \]
\[ \forall C : \text{Cat}; \ G : \text{Gr}; \ m : \text{Morph} G2C \bullet \text{morphD}(C, G, m) = m \]
\[ \forall D : \text{Diag} \bullet \text{NsD} D = \text{Ns}(\text{grD} D) \]
\[ \forall C : \text{Cat}; \ G : \text{Gr}; \ m : \text{Morph} G2C \bullet \text{obsD}(C, G, m) = \text{ran}(\text{mv} m) \]
\[ \forall C : \text{Cat}; \ G : \text{Gr}; \ m : \text{Morph} G2C \bullet \text{morphsD}(C, G, m) = \text{ran}(\text{me} m) \]

\[ CC ::= (\lambda D : \text{Diag} \bullet \{X : \text{obs}(\text{cat} D) ; \ ms : \mathcal{P}(\text{morphs}(\text{cat} D)) \mid \\
\forall m : ms \bullet \text{dom} C(\text{cat} D)m \in \text{obsD} D \land \text{cod} C(\text{cat} D)m = X\}) \]

\[ \text{ValCC} : \text{Diag} \to \mathcal{P}(O \times \mathcal{P} M) \]

\[ \forall D : \text{Diag}; \ X : O; \ ms : \mathcal{P} M \bullet (X, ms) \in \text{ValCC} D \iff (X, ms) \in CC D \\
\land (\forall m : \text{morphsD} D \bullet (\exists f, g : ms \bullet (\text{dom} C(\text{cat} D)m = \text{dom} C(\text{cat} D)f \\
\land \text{cod} C(\text{cat} D)m = \text{dom} C(\text{cat} D)g \land \text{comp}(\text{cat} D)(g, m) = f))) \]

59
\[
\text{Colimit} = \langle \lambda D : \text{Diag} \ast (\mu X : O; \ms : \mathbb{P} M \mid (X, ms) \in \text{ValCC} D \\
\ast (\forall X' : \text{obs}(\text{cat} D); \ms' : \mathbb{P} M \mid X \neq X' \wedge (X', ms') \in \text{ValCC} D) \ast \\
(\exists k : \text{CatMorphs}(\text{cat} D)(X, X') \ast (\forall f : ms; g : ms' \mid \text{dom} C(\text{cat} D)f = \text{dom} C(\text{cat} D)g \ast \\
\text{comp}(\text{cat} D)(k, f) = g))))\rangle
\]

\[
\begin{align*}
\text{obCC} : O \times \mathbb{P} M &\to O \\
\text{morphsCC} : O \times \mathbb{P} M &\to \mathbb{P} M \\
\forall X : O; \ms : \mathbb{P} M &\ast \text{obCC}(X, \ms) = X \\
\forall X : O; \ms : \mathbb{P} M &\ast \text{morphsCC}(X, \ms) = \ms
\end{align*}
\]

### B.4 The Graphs Category

\textit{section Fragmenta\_GraphsCat parents standard\_toolkit, Fragmenta\_Graphs, Fragmenta\_CatTheory}

\[
\begin{align*}
\text{OGr} &: \mathbb{P} O \\
\text{MGr} &: \mathbb{P} M \\
\text{OGrToGr} &: O \to \text{Gr} \\
\text{MGrToGrM} &: M \to \text{GrMorph}
\end{align*}
\]

\[
\begin{align*}
\text{idGr} &: \text{OGr} \to \text{MGr} \\
\text{domGr}, \text{codGr} &: \text{MGr} \to \text{OGr} \\
\forall oG : \text{OGr}; mG : \text{MGr} &\ast \text{idGr} oG = mG \iff (\exists G : \text{Gr}; \text{GM} : \text{GrMorph} \ast \\
&G = \text{OGrToGr} oG \land \text{MGrToGrM} mG = \text{GM} \land \text{GM} = (\text{id}\langle Ns G, \text{id}\langle Es G \rangle) \\
\forall mG : \text{MGr}; oG1 : \text{OGr} &\ast \text{domGr} mG = oG1 \iff (\exists \text{GM} : \text{GrMorph}; G1, G2 : \text{Gr} \ast \\
&\text{GM} = \text{MGrToGrM} mG \land G1 = \text{OGrToGr} oG1 \land \text{GM} \in \text{morphG}(G1, G2)) \\
\forall mG : \text{MGr}; oG2 : \text{OGr} &\ast \\
\text{codGr} mG &\ast oG2 \iff (\exists \text{GM} : \text{GrMorph}; G1, G2 : \text{Gr} \ast \\
&\text{GM} = \text{MGrToGrM} mG \land G2 = \text{OGrToGr} oG2 \land \text{GM} \in \text{morphG}(G1, G2))
\end{align*}
\]

\[
\begin{align*}
\text{cmpGr} &: \text{MGr} \times \text{MGr} \to \text{MGr} \\
\forall mG1, mG2, mG3 : \text{MGr} &\ast \text{cmpGr}(mG1, mG2) = mG3 \iff \\
&\langle \exists GM1, GM2, GM3 : \text{GrMorph} \ast GM1 = \text{MGrToGrM} mG1 \land GM2 = \text{MGrToGrM} mG2 \\
&\land GM3 = \text{MGrToGrM} mG1 \land GM3 = GM1 \circ C GM2 \rangle
\end{align*}
\]

\[
\begin{align*}
\text{GraphsC} &: \text{Cat} \\
\text{GraphsC} &\equiv (\text{OGr}, \text{MGr}, \text{domGr}, \text{codGr}, \text{idGr}, \text{cmpGr})
\end{align*}
\]
B.5 Structural Graphs

\section{Fragmenta\_SGs parents standard\_toolkit, Fragmenta\_Generics, Fragmenta\_Graphs}

\[\text{SGNT} ::= \text{nrmnl} \mid \text{nabst} \mid \text{nprxy}\]
\[\text{SGET} ::= \text{einh} \mid \text{ecomp} \mid \text{cret} \mid \text{elnk} \mid \text{eref}\]
\[\text{MultUVal} ::= \text{val}\langle\mathbb{N}\rangle \mid \text{many}\]
\[\text{MultVal} ::= \text{mr}\langle\mathbb{N} \times \text{MultUVal}\rangle \mid \text{ms}\langle\text{MultUVal}\rangle\]

\[\text{Mult} : \mathbb{P} \text{MultVal}\]

\[\text{Mult} = \{\text{mv} : \text{MultVal} \mid (\exists \text{lb} : \mathbb{N}; \text{ub} : \text{MultUVal} \bullet \text{mv} = \text{mr}(\text{lb}, \text{ub}) \land \text{ub} = \text{many}\]
\[\lor (\exists \text{ubn} : \mathbb{N} \bullet \text{ub} = \text{val} \text{ubn} \land \text{lb} \leq \text{ubn}) \lor (\exists \text{umv} : \text{MultUVal} \bullet \text{mv} = \text{ms} \text{umv})\}\]

\[\text{relation(multOk\_)}\]

\[\\forall \text{vs} : \mathbb{P} V; \text{lb} : \mathbb{N}; \text{ub} : \text{MultUVal} \bullet (\text{multOk}(\text{vs}, \text{mr}(\text{lb}, \text{ub}))) \equiv\]
\[\# \text{vs} \geq \text{lb} \land (\text{ub} = \text{many} \lor (\exists \text{ubn} : \mathbb{N} \bullet \text{ub} = \text{val} \text{ubn} \land \# \text{vs} \leq \text{ubn}))\]
\[\lor (\exists \text{bn} : \mathbb{N} \bullet \text{v} = \text{val} \text{bn} \land \# \text{vs} = \text{bn})\]

\[\text{SGr}_0 =\{\text{G} : \text{Gr}\}; \text{nt} : V \rightarrow \text{SGNT}; \text{et} : E \rightarrow \text{SGET}; \text{sm}, \text{tm} : E \rightarrow \text{Mult} \mid\]
\[\text{nt} \in \text{Ns} \text{G} \rightarrow \text{SGNT} \land \text{et} \in \text{Es} \text{G} \rightarrow \text{SGET}\]

\[\text{gr} : \text{SGr}_0 \rightarrow \text{Gr}\]
\[\text{sgr\_Ns} : \text{SGr}_0 \rightarrow \mathbb{P} V\]
\[\text{sgr\_Es} : \text{SGr}_0 \rightarrow \mathbb{P} E\]
\[\text{sgr\_src} : \text{SGr}_0 \rightarrow E \rightarrow V\]
\[\text{sgr\_tgt} : \text{SGr}_0 \rightarrow E \rightarrow V\]
\[\text{nty} : \text{SGr}_0 \rightarrow V \rightarrow \text{SGNT}\]
\[\text{et} : \text{SGr}_0 \rightarrow E \rightarrow \text{SGET}\]
\[\text{srcm} : \text{SGr}_0 \rightarrow E \rightarrow \text{Mult}\]
\[\text{tgtn} : \text{SGr}_0 \rightarrow E \rightarrow \text{Mult}\]

\[\\forall \text{G} : \text{Gr}; \text{nt} : V \rightarrow \text{SGNT}; \text{et} : E \rightarrow \text{SGET}; \text{sm}, \text{tm} : E \rightarrow \text{Mult} \bullet \text{gr}(\text{G}, \text{nt}, \text{et}, \text{sm}, \text{tm}) = \text{G}\]
\[\\forall \text{SG} : \text{SGr}_0 \bullet \text{sgr\_Ns SG} = \text{Ns(gr SG)}\]
\[\\forall \text{SG} : \text{SGr}_0 \bullet \text{sgr\_Es SG} = \text{Es(gr SG)}\]
\[\\forall \text{SG} : \text{SGr}_0 \bullet \text{sgr\_src SG} = \text{src(gr SG)}\]
\[\\forall \text{SG} : \text{SGr}_0 \bullet \text{sgr\_tgt SG} = \text{tgt(gr SG)}\]
\[\\forall \text{G} : \text{Gr}; \text{nt} : V \rightarrow \text{SGNT}; \text{et} : E \rightarrow \text{SGET}; \text{sm}, \text{tm} : E \rightarrow \text{Mult} \bullet \text{nty}(\text{G}, \text{nt}, \text{et}, \text{sm}, \text{tm}) = \text{nt}\]
\[\\forall \text{G} : \text{Gr}; \text{nt} : V \rightarrow \text{SGNT}; \text{et} : E \rightarrow \text{SGET}; \text{sm}, \text{tm} : E \rightarrow \text{Mult} \bullet \text{et}(\text{G}, \text{nt}, \text{et}, \text{sm}, \text{tm}) = \text{et}\]
\[\\forall \text{G} : \text{Gr}; \text{nt} : V \rightarrow \text{SGNT}; \text{et} : E \rightarrow \text{SGET}; \text{sm}, \text{tm} : E \rightarrow \text{Mult} \bullet \text{srcm}(\text{G}, \text{nt}, \text{et}, \text{sm}, \text{tm}) = \text{sm}\]
\[\\forall \text{G} : \text{Gr}; \text{nt} : V \rightarrow \text{SGNT}; \text{et} : E \rightarrow \text{SGET}; \text{sm}, \text{tm} : E \rightarrow \text{Mult} \bullet \text{tgtn}(\text{G}, \text{nt}, \text{et}, \text{sm}, \text{tm}) = \text{tm}\]
\[ \text{NsTy : } SG_{0} \times \mathbb{P} \text{SGNT} \to \mathbb{P} V \]
\[ \text{EsTy : } SG_{0} \times \mathbb{P} \text{SGET} \to \mathbb{P} E \]
\[ \forall SG : SG_{0}; \ nts : \mathbb{P} \text{SGNT} \bullet \text{NsTy}(SG, nts) = (nty \ SG) \sim \{nts\} \]
\[ \forall SG : SG_{0}; \ ets : \mathbb{P} \text{SGET} \bullet \text{EsTy}(SG, ets) = (ety \ SG) \sim \{ets\} \]

\[ \text{EsA : } SG_{0} \to \mathbb{P} E \]
\[ \text{EsR : } SG_{0} \to \mathbb{P} E \]
\[ \forall SG : SG_{0} \bullet \text{EsA SG} = \text{EsTy}(SG, \{\text{erel}, \text{ecomp}, \text{elnk}\}) \]
\[ \forall SG : SG_{0} \bullet \text{EsR SG} = \text{EsTy}(SG, \{\text{eref}\}) \]

\[ \text{NsP : } SG_{0} \to \mathbb{P} V \]
\[ \forall SG : SG_{0} \bullet \text{NsP SG} = \text{NsTy}(SG, \{\text{nprzy}\}) \]

\[ \text{inhG : } SG_{0} \to Gr \]
\[ \text{inh : } SG_{0} \to V \leftrightarrow V \]
\[ \forall SG : SG_{0} \bullet \text{inhG SG} = \text{restrict}((gr \ SG), (\text{EsTy}(SG, \{\text{einh}\}) \setminus \text{EsId}(gr \ SG))) \]
\[ \forall SG : SG_{0} \bullet \text{inh SG} = \text{rel}(\text{inhG SG}) \]

\[ \text{SGr } = \{SG : SG_{0} \mid \text{EsR SG } \subseteq \text{EsId}(gr \ SG) \land \text{srm SG } \in \text{EsTy}(SG, \{\text{erel}, \text{ecomp}\}) \to \text{Mult} \]
\[ \land \text{tgtn SG } \in \text{EsTy}(SG, \{\text{erel}, \text{ecomp}\}) \to \text{Mult} \]
\[ \land \text{srm SG } \{\text{EsTy}(SG, \{\text{ecomp}\})\} \subseteq \{\text{mr}(0, \text{val } 1), \text{ms } (\text{val } 1)\} \]
\[ \land \text{acyclicG } (\text{inhG SG}) \} \]

\[ \text{EsRP : } SGr \to \mathbb{P} E \]
\[ \forall SG : SGr \bullet \text{EsRP SG} = \{e : \text{EsR SG } \mid \text{sgr_srg SG } e \in \text{NsP SG}\} \]

\[ \text{inhst : } SGr \to V \leftrightarrow V \]
\[ \text{clan : } V \times SGr \to \mathbb{P} V \]
\[ \forall SG : SGr \bullet \text{inhst SG} = (\text{inh SG})^{*} \]
\[ \forall v : V; \ SGr \bullet \text{clan}(v, SG) = \{v' : \text{sgr_srg SG } \mid v' \leftrightarrow v \in \text{inhst SG}\} \]

\[ \text{srcst : } SGr \to E \leftrightarrow V \]
\[ \forall SG : SGr \bullet \text{srcst SG} = \{e : \text{EsA SG } \mid \exists v_{2} : \text{sgr_srg SG } v \in \text{clan}(v_{2}, SG) \land \text{sgr_srg SG } e = v_{2}\} \]
\[
tgst : SGr \rightarrow E \leftrightarrow V
\]
\[
\forall SG : SGr \bullet \tgst SG = \{ e : EsA SG ; v : sgr_Ns SG \mid \\
\exists v_2 : sgr_Ns SG \bullet v \in \text{clan}(v_2, SG) \land sgr_{\tgst} SG e = v_2 \}
\]

**relation** (\text{disjSGs}_-) 

\[
\text{disjSGs}_- : \mathbb{P}(SGr \times SGr)
\]
\[
\forall SG_1, SG_2 : SGr \bullet (\text{disjSGs}(SG_1, SG_2)) \Leftrightarrow (\text{disjGs}(gr\ SG_1, gr\ SG_2))
\]

**function 10** \text{leftassoc} (\_ \cup SG \_)

\[
\_ \cup SG \_ : SGr \times SGr \rightarrow SGr
\]
\[
\forall SG_1, SG_2 : SGr \bullet SG_1 \cup_{SG} SG_2 = (gr\ SG_1 \cup_{SG} gr\ SG_2, nty\ SG_1 \cup_{SG} nty\ SG_2, \\
ety\ SG_1 \cup_{SG} ety\ SG_2, srcm\ SG_1 \cup_{SG} srcm\ SG_2, tgtm\ SG_1 \cup_{SG} tgtm\ SG_2) \Leftrightarrow (\text{disjSGs}(SG_1, SG_2))
\]

\[
morphSG : SGr \times SGr \rightarrow \mathbb{P}\ GrMorph
\]
\[
\forall SG_1, SG_2 : SGr \bullet \\
morphSG(SG_1, SG_2) = \{ f_v : sgr_{Ns} SG_1 \rightarrow sgr_{Ns} SG_2 ; f_e : sgr_{Es} SG_1 \rightarrow sgr_{Es} SG_2 \mid \\
f_v \circ srcst SG_1 \subseteq srcst SG_2 \circ f_e \land f_v \circ tgtst SG_1 \subseteq tgtst SG_2 \circ f_e \\
\land f_v \circ \text{inhst} SG_1 \subseteq \text{inhst} SG_2 \circ f_e \}
\]

**B.6 Fragments**

**section** Fragmenta\_Frs parents standard\_toolkit, Fragmenta\_SGs

\[
Fr_0 \equiv \{ SG : SGr ; tr : E \rightarrow V \mid tr \in EsRP SG \rightarrow V \\
\land (EsRP SG) \cup (sgr\_{src} SG) \in (EsRP SG) \rightarrow NsP\ SG \\
\land EsTy(SG, \{\text{einh}\}) \cup sgr\_{src} SG \triangleright NsP\ SG = \{\} \}
\]

63
fsrcGr : Fr0 → Gr
ftgr : Fr0 → E → V
fNs : Fr0 → P V
fEs : Fr0 → P E
fEsR : Fr0 → P E
fsg : Fr0 → SGr
fsrc : Fr0 → E → V
ftgt : Fr0 → E → V
∀ SG : SGr; tr : E → V • fsrcGr(SG, tr) = gr SG
∀ SG : SGr; tr : E → V • ftgr(SG, tr) = tr
∀ SG : SGr; tr : E → V • fNs(SG, tr) = sgr_Ns SG
∀ SG : SGr; tr : E → V • fEs(SG, tr) = sgr_Es SG
∀ SG : SGr; tr : E → V • fEsR(SG, tr) = EsR SG
∀ SG : SGr; tr : E → V • fsg(SG, tr) = SG
∀ SG : SGr; tr : E → V • fsrc(SG, tr) = sgr_src SG
∀ SG : SGr; tr : E → V • ftgt(SG, tr) = sgr_tgt SG

tgtr : Fr0 → E → V
withRsG : Fr0 → Gr
refsG : Fr0 → Gr
refs : Fr0 → V → V
reps : Fr0 → V → V
referenced : Fr0 → P V
∀ SG : SGr; tr : E → V • tgtr(SG, tr) = sgr_tgt SG ⊕ tr
∀ SG : SGr; tr : E → V •
  withRsG(SG, tr) = (sgr_Ns SG ∪ ran tr, sgr_Es SG, sgr_src SG, tgtr(SG, tr))
∀ F : Fr0 • refsG F = restrict((withRsG F), (EsRP(fsg F)))
∀ F : Fr0 • reps F = rel(refsG F)
∀ F : Fr0 • refs F = refs F ∪ (refs F) ~
∀ SG : SGr; tr : E → V • referenced(SG, tr) = ran tr

inhF : Fr0 → V ↔ V
∀ F : Fr0 • inhF F = inh(fsg F) ⊕ reps F

refsOf : Fr0 → V → P V
∀ F : Fr0; v : V • refsOf F v = (refs F) + {v}

nonPRefsOf : Fr0 → V → P V
∀ F : Fr0; v : V • nonPRefsOf F v = {v2 : V | v2 ∈ refsOf F v ∧ v2 ∈ NsP(fsg F)}

relation(acyclicIF_ )
acyclicIF_ := \forall F : Fr_0 \bullet (\text{acyclicIF } F) \iff (\text{inh}(fsg F) \cup \text{refs } F) \in \text{acyclic}

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

repsOf : \forall V \to Fr \to \mathbb{P} \ V
\forall v : V ; F : Fr \bullet \text{repsOf } v F = \{ v' : fNs F \mid (v', v) \in (\text{reps } F) \}^*

fr\_NsAbst : Fr \to \mathbb{P} \ V
\forall F : Fr \bullet \text{fr\_NsAbst } F = \bigcup \{ v a : \text{NsTy}(fsg F), \{ nabst \} \bullet (\text{repsOf } v a F) \}

relation(disjFs_-)

disjFs_- : \mathbb{P}(Fr \times Fr)
\forall F_1, F_2 : Fr \bullet (\text{disjFs}(F_1, F_2)) \iff (\text{disjSGs}(fsg F_1, fsg F_2))

function 10 leftassoc (_\cup_{Fr} _-)

_\cup_{Fr} _- : Fr \times Fr \to Fr
\forall F_1, F_2 : Fr \bullet F_1 \cup_{Fr} F_2 = (fsg F_1 \cup_{SG} fsg F_2, ftgtr F_1 \cup ftgtr F_2) \iff (\text{disjFs}(F_1, F_2))

inhstF : Fr \to V \leftrightarrow V
\forall F : Fr \bullet \text{inhst } F = (\text{inhP } F)^*

clanF : V \times Fr \to \mathbb{P} \ V
\forall v : V ; F : Fr \bullet \text{clanF}(v, F) = \{ v' : fNs F \mid (v', v) \in \text{inhst } F \}

srcstF : Fr \to E \leftrightarrow V
\forall F : Fr \bullet \text{srcst } F = \{ e : \text{EsA}(fsg F) ; v : fNs F \mid \exists v_2 : fNs F \bullet v \in \text{clanF}(v_2, F) \land (e, v_2) \in \text{srcst } (fsg F) \}
\[ tgtstF : Fr \rightarrow E \leftrightarrow V \]
\[ \forall F : Fr \bullet tgtstF F = \{ e : EsA(fsg F); v : fNs F | \exists v_2 : fNs F \bullet v \in clanF(v_2, F) \land (e, v_2) \in tgtst(fsg F) \} \]

\[ \text{morph} F : Fr \times Fr \rightarrow \mathcal{P} \text{GrMorph} \]
\[ \forall F_1, F_2 : Fr \bullet \text{morph} F(F_1, F_2) = \{ f_o : fNs F_1 \rightarrow fNs F_2; f_e : fEs F_1 \rightarrow fEs F_2 \mid f_o \circ srcst F_1 \subseteq srcst F_2 \circ f_e \land f_o \circ tgtst F_1 \subseteq tgtst F_2 \circ f_e \land f_o \circ instst F_1 \subseteq instst F_2 \circ f_e \} \]

### B.7 Models

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

\[ Mdl_0 = \{ GFG : GFGr; CG : CGr; fcl : GrMorph; fdef : V \rightarrow Fr \mid fcl \in \text{morph} GFG CG(GFG, CG) \land fdef \in gfgNs GFG \rightarrow Fr \} \]

\[ \text{mgfg} : Mdl_0 \rightarrow GFGr \]
\[ \text{mcg} : Mdl_0 \rightarrow CGr \]
\[ \text{mfcl} : Mdl_0 \rightarrow \text{GrMorph} \]
\[ \text{mfdef} : Mdl_0 \rightarrow V \rightarrow Fr \]
\[ \forall GFG : GFGr, CG : CGr; fcl : GrMorph; fdef : V \rightarrow Fr \bullet \text{mgfg}(GFG, CG, fcl, fdef) = GFG \]
\[ \forall GFG : GFGr, CG : CGr; fcl : GrMorph; fdef : V \rightarrow Fr \bullet \text{mcg}(GFG, CG, fcl, fdef) = CG \]
\[ \forall GFG : GFGr, CG : CGr; fcl : GrMorph; fdef : V \rightarrow Fr \bullet \text{mfcl}(GFG, CG, fcl, fdef) = fcl \]
\[ \forall GFG : GFGr; CG : CGr; fcl : GrMorph; fdef : V \rightarrow Fr \bullet \text{mfdef}(GFG, CG, fcl, fdef) = fdef \]

\[ \text{UF} s : Mdl_0 \rightarrow Fr \]
\[ \text{UF}_{S_0} : \mathbb{P}_1 Fr \rightarrow Fr \]
\[ \forall M : Mdl_0 \bullet \text{UF}s M = \text{UF}_{S_0}(\text{ran}(\text{mfdef} M)) \]
\[ \forall F : Fr \bullet \text{UF}_{S_0}\{ F \} = F \]
\[ \forall F : Fr; Fs : \mathbb{P}_1 Fr \bullet \text{UF}_{S_0}\{ F \} \cup Fs = F \cup F (\text{UF}_{S_0} Fs) \]

\[ \text{from} V : V \times Mdl_0 \rightarrow V \]
\[ \forall vl : V; M : Mdl_0; vf : V \bullet \text{from} V(vl, M) = vf \iff vl \in fNs(mfdef M vf) \]
\[\forall v : V; \; M : Mdl_0; \; v : V \to V; \; f : E \to E \mapsto \cdot \]
\[\text{consFToGFG}(v, M) = (v, f) \mapsto\]
\[(\exists F : Fr; \; GFG : GFG_r \cdot F = \text{mfdef M} v f \land GFG' = \text{mgfg M} \land v f \in \text{fNs} F \to \text{fgNs} GFG \]
\[\land f e \in fEs F \to \text{fgEs} GFG \land v f \in \text{fgNs} GFG \]
\[\land (\exists e : \text{fgEs} GFG \cdot (\text{src}(\text{fgfG GFG}) e f = \text{tgt}(\text{fgfG GFG}) e f = v f \land v f = \text{fNs} F \times \{e f\}) \mapsto \text{consFToGFGRef}(v f, (\text{fEsR} F, M)))\]
\[\forall v : V; \; M : Mdl_0; \; f e : E \to E \mapsto \text{consFToGFGRef}(v f, \{\}, M) = \{\}\]
\[\forall v : V; \; M : Mdl_0; \; \epsilon l : E; \; \text{Er} : \exists E; \; f e : E \to E \mapsto \cdot \]
\[\text{consFToGFGRef}(v f, (\{\epsilon l\} \cup \text{Er}), M) = f e \mapsto\]
\[(\exists F : Fr; \; GFG : GFG_r \cdot F = \text{mfdef M} v f \land GFG' = \text{mgfg M} \]
\[\land (\exists e : \text{fgEs} GFG \cdot (\text{src}(\text{fgfG GFG}) e f = v f \land v f = \text{fNs} F \times \{e f\}) \mapsto \text{consFToGFGRef}(v f, (\text{fEsR} F, M)))\]

\[\forall v : V; \; F : Fr; \; M : Mdl_0 \cdot \text{buildUFsToGFG}(((v f \mapsto F), M) = \text{consFToGFG}(v f, M)\]
\[\forall v : V; \; F : Fr; \; \text{def} : V \to V; \; M : Mdl_0 \mapsto\]
\[\text{buildUFsToGFG}(((v f \mapsto F) \cup \text{def}), M) = \text{consFToGFG}(v f, M) \cup \text{buildUFsToGFG}(\text{def}, M)\]

\[\text{Mdl} = \{M : Mdl_0 \mid \text{umToGFG} M \in \text{morphFFGFG}(\{\text{UFs} M\}, \{\text{mgfg M}\})\]
\[\land (\forall v f_1, v f_2 : \text{fgfNs} (\text{mgfg M}) \mid v f_1 \neq v f_2 \cdot \text{disfFs}(\text{mfdef M} v f_1, \text{mfdef M} v f_2))\]
\[ tsgSG : TSGr \rightarrow SGr \]
\[ tsget : TSGr \rightarrow E \rightarrow SGET \]
\[ tsgetA : TSGr \rightarrow E \]
\[ tsgetC : TSGr \rightarrow E \rightarrow \text{Mult} \]
\[ tsgetrcm : TSGr \rightarrow E \rightarrow \text{Mult} \]

\[ \forall SG : SGr; \text{iet} : E \rightarrow SGET \cdot tsgSG(SG, \text{iet}) = SG \]
\[ \forall SG : SGr; \text{iet} : E \rightarrow SGET \cdot tsget(SG, \text{iet}) = \text{iet} \]
\[ \forall TSG : TSGr \cdot tsgEsA_TSG = Esa(tsgSG TSG) \]
\[ \forall TSG : TSGr \cdot tsgEsC_TSG = EsTy((tsgSG TSG), \{\text{ecomp}\}) \]
\[ \forall TSG : TSGr \cdot tsgrcm_TSG = srcm(tsgSG TSG) \]
\[ \forall TSG : TSGr \cdot tsgettm_TSG = tgm(tsgSG TSG) \]

**relation(instanceEdgesOk)_**

\[ \text{instanceEdgesOk}_- : \mathbb{P}(\mathbb{SGr} \times \mathbb{SGr} \times (E \rightarrow \mathbb{SGET}) \times \mathbb{GrMorph}) \]
\[ \forall SG, TSG : SGr; \text{iet} : E \rightarrow \mathbb{SGET}; \text{type} : \mathbb{GrMorph} \cdot \]
\[ \langle \text{instanceEdgesOk}(SG, TSG, \text{iet}, \text{type}) \rangle \Leftrightarrow \text{iet} \circ \text{iet} \circ \text{type} = \text{ety} \ SG \]

\[ SGrTy =\{ SG : SGr; TSG : TSGr; \text{type} : \mathbb{GrMorph} \mid \text{type} \in \text{morph SG}(SG, (tsgSG TSG)) \land \]
\[ \langle \text{instanceEdgesOk}(SG, tsgSG TSG, tsget TSG, \text{type}) \rangle \}\]

\[ sgtSG : SGrTy \rightarrow SGr \]
\[ sgtTSG : SGrTy \rightarrow TSGr \]
\[ sgtType : SGrTy \rightarrow \mathbb{GrMorph} \]
\[ \forall SG : SGr; TSG : TSGr; \text{type} : \mathbb{GrMorph} \cdot sgtSG(SG, TSG, \text{type}) = SG \]
\[ \forall SG : SGr; TSG : TSGr; \text{type} : \mathbb{GrMorph} \cdot sgtTSG(SG, TSG, \text{type}) = TSG \]
\[ \forall SG : SGr; TSG : TSGr; \text{type} : \mathbb{GrMorph} \cdot sgtType(SG, TSG, \text{type}) = \text{type} \]

\[ sgtNs : SGrTy \rightarrow \mathbb{P} \ V \]
\[ sgtEs : SGrTy \rightarrow \mathbb{P} \ E \]
\[ sgtSrc : SGrTy \rightarrow E \rightarrow \mathbb{V} \]
\[ sgtTgt : SGrTy \rightarrow E \rightarrow \mathbb{V} \]
\[ \forall SGT : SGrTy \cdot sgtNs_SGT = sgr_{\text{ns}}(sgtSG SGT) \]
\[ \forall SGT : SGrTy \cdot sgtEs_SGT = sgr_{\text{es}}(sgtSG SGT) \]
\[ \forall SGT : SGrTy \cdot sgtEsI_SGT = EsTy((sgtSG SGT), \{\text{einh}\}) \]
\[ \forall SGT : SGrTy \cdot sgtSrc_SGT = sgr_{\text{src}}(sgtSG SGT) \]
\[ \forall SGT : SGrTy \cdot sgtTgt_SGT = sgr_{\text{tgt}}(sgtSG SGT) \]

**relation(abstractNoDirectInstances)_**

68
\[\text{abstractNoDirectInstances} : \mathcal{P}(\text{SGr} \times \text{SGr} \times \text{GrMorph})\]

\[\forall \text{SG} : \text{SGr}; \text{TSG} : \text{SGr}; \text{type} : \text{GrMorph} \bullet (\text{abstractNoDirectInstances}(\text{SG}, \text{TSG}, \text{type})) \equiv (\text{fV type}) \sim \{\text{NsTy}(\text{TSG}, \text{\{nabst\}})\} = \{\}\]

\text{relation} (\text{containmentNoSharing}_-)

\[\text{containmentNoSharing} : \mathcal{P}(\text{SGr} \times \text{SGr} \times \text{GrMorph})\]

\[\forall \text{SG} : \text{SGr}; \text{TSG} : \text{SGr}; \text{type} : \text{GrMorph} \bullet (\text{containmentNoSharing}(\text{SG}, \text{TSG}, \text{type})) \equiv (\text{fE type}) \sim \{\text{EsTy}(\text{TSG}, \text{\{ecomp\}})\} \triangleleft tgst SG \in \text{injrel}\]

\text{relation} (\text{instMultsOk}_-)

\[\text{instMultsOk} : \mathcal{P}(\text{SGr} \times \text{SGr} \times \text{GrMorph})\]

\[\forall \text{SG} : \text{SGr}; \text{TSG} : \text{SGr}; \text{type} : \text{GrMorph} \bullet (\text{instMultsOk}(\text{SG}, \text{TSG}, \text{type})) \equiv (\forall \text{te} : \text{EsA} \text{TSG} \bullet (\exists \text{V} : \text{V} \Rightarrow \text{V} \bullet \text{r} = \text{rel}\text{\{\text{restrict}((\text{gr SG}), ((\text{fE type}) \sim \{\text{te}\})\}))})\]

\[\land (\forall \text{v} : \text{dom}\text{r} \bullet (\text{multOk}(\text{r} \sim \{\text{v}\}), \text{srcm TSG}\text{te}))\]

\[\land (\forall \text{v} : \text{ran}\text{r} \bullet (\text{multOk}(\text{r} \sim \{\text{v}\}), \text{tgtm TSG}\text{te})))\]

\text{relation} (\text{instContainmentAcyclic}_-)

\[\text{instContainmentAcyclic} : \mathcal{P}(\text{SGr} \times \text{SGr} \times \text{GrMorph})\]

\[\forall \text{SG} : \text{SGr}; \text{TSG} : \text{SGr}; \text{type} : \text{GrMorph} \bullet (\text{instContainmentAcyclic}(\text{SG}, \text{TSG}, \text{type})) \equiv (\text{acyclicG restrict}((\text{gr SG}), ((\text{fE type}) \sim \{\text{EsTy}(\text{TSG}, \text{\{ecomp\}})\}))\]

\text{relation} (\text{isConformable}_-)

\[\text{isConformable} : \mathcal{P}(\text{SGr} \times \text{SGr} \times \text{GrMorph})\]

\[\forall \text{SG}, \text{TSG} : \text{SGr}; \text{type} : \text{GrMorph} \bullet (\text{isConformable}(\text{SG}, \text{TSG}, \text{type})) \equiv (\text{abstractNoDirectInstances}(\text{SG}, \text{TSG}, \text{type})) \land (\text{containmentNoSharing}(\text{SG}, \text{TSG}, \text{type})) \land (\text{instMultsOk}(\text{SG}, \text{TSG}, \text{type})) \land (\text{instContainmentAcyclic}(\text{SG}, \text{TSG}, \text{type}))\]

\[\text{SGTyConf} = \{\text{SG} : \text{SGr}; \text{TSG} : \text{SGr}; \text{type} : \text{GrMorph} \mid |\text{isConformable}(\text{SG}, \text{tgtSG TSG}, \text{type})\}\]

\[\text{morphSGT} = (\lambda \text{SGT}_1, \text{SGT}_2 : \text{SGrTy} \bullet \{m : \text{morphSG}((\text{sgrSG SGTy}_1), (\text{sgrSG SGTy}_2)) \mid \text{sgtType SGTy}_2 \circ_G m = \text{sgtType SGTy}_1\})\]

69
section Fragmenta_TyFrs parents standard_toolkit, Fragmenta_Frs

\[ TFr = \{ F : Fr; \text{iet} : E \rightarrow \text{SGET} | \text{iet} \in \text{EsA}(fsg F) \rightarrow \text{SGET} \} \]

\[
\begin{align*}
& \text{tfG} : TFr \rightarrow Gr \\
& \text{tfNs} : TFr \rightarrow \mathbb{P} V \\
& \text{tfEs} : TFr \rightarrow \mathbb{P} E \\
& \text{tfEsR} : TFr \rightarrow \mathbb{P} E \\
& \text{tfF} : TFr \rightarrow Fr \\
& \text{tfet} : TFr \rightarrow E \rightarrow \text{SGET} \\
& \forall F : Fr; \text{set} : E \rightarrow \text{SGET} \cdot \text{tfG}(F, \text{iet}) = \text{fsrcGr} F \\
& \forall F : Fr; \text{set} : E \rightarrow \text{SGET} \cdot \text{tfNs}(F, \text{iet}) = fN\text{s} F \\
& \forall F : Fr; \text{set} : E \rightarrow \text{SGET} \cdot \text{tfEs}(F, \text{iet}) = f\text{Es} F \\
& \forall F : Fr; \text{set} : E \rightarrow \text{SGET} \cdot \text{tfEsR}(F, \text{iet}) = f\text{EsR} F \\
& \forall F : Fr; \text{set} : E \rightarrow \text{SGET} \cdot \text{tfF}(F, \text{iet}) = F \\
& \forall F : Fr; \text{set} : E \rightarrow \text{SGET} \cdot \text{tfet}(F, \text{iet}) = \text{iet} \\
\end{align*}
\]

function 10 leftassoc (_UTF_)

\[
\_UTF_ : TFr \times TFr \rightarrow TFr \\
\forall TF_1, TF_2 : TFr \cdot TF_1 \_UTF_ TF_2 = (tfF TF_1 \cup_F tfF TF_2, tfet TF_1 \cup tfet TF_2)
\]

\[ FrTy = \{ F : Fr; TF : TFr; \text{type} : \text{GrMorph} | \text{type} \in \text{morphF}(F, (tfF TF)) \} \]

relation(instanceEdgeTypesOkF)

\[
\begin{align*}
& \text{instanceEdgeTypesOkF} : \mathbb{P}(Fr \times TFr \times \text{GrMorph}) \\
& \forall F : Fr; TF : TFr; \text{type} : \text{GrMorph} \cdot (\text{instanceEdgeTypesOkF}(F, TF, \text{type})) \Leftrightarrow \\
& \quad \text{tfet} TF \circ f\text{E type} = \text{ety}(fsg F)
\end{align*}
\]

relation(abstractNoDirectInstancesF)

70
abstractNoDirectInstancesF_ : \mathbb{P} FrTy

\forall F : Fr; TF : TFr; type : GrMorph \bullet (\text{abstractNoDirectInstancesF}(F, TF, type)) \iff (\text{FrTyConf}(\text{FrTyConf} \cap \text{abstractNoDirectInstancesF}(F, TF, type)))

relation(containmentNoSharingF_)

containmentNoSharingF_ : \mathbb{P}(Fr \times Fr \times GrMorph)

\forall F, TF : Fr; type : GrMorph \bullet (\text{containmentNoSharingF}(F, TF, type)) \iff (\text{InstContainmentForest}(\text{FrTyConf} \cap \text{AbstractNoDirectInstancesF}(F, TF, type)) \cap \text{ContainmentNoSharingF}(F, TF, type))

relation(instMultsOkF_)

instMultsOkF_ : \mathbb{P}(Fr \times Fr \times GrMorph)

\forall F, TF : Fr; type : GrMorph \bullet (\text{InstMultsOkF}(F, TF, type)) \iff (\text{InstMultsOkF}(F, TF, type)) \cap (\text{FinMultsOkF}(F, TF, type)) \cap (\text{EsTyConf}(F, TF, type))

relation(instContainmentForest_)

instContainmentForest_ : \mathbb{P}(Fr \times Fr \times GrMorph)

\forall F, TF : Fr; type : GrMorph \bullet (\text{InstContainmentForest}(F, TF, type)) \iff (\text{InstContainmentForest}(F, TF, type)) \cap (\text{FinMultsOkF}(F, TF, type)) \cap (\text{EsTyConf}(F, TF, type))

relation(isConformableF_)

isConformableF_ : \mathbb{P}(Fr \times TFr \times GrMorph)

\forall F : Fr; TF : TFr; type : GrMorph \bullet (\text{isConformableF}(F, TF, type)) \iff (\text{InstanceEdgeTypesOkF}(F, TF, type)) \land (\text{AbstractNoDirectInstancesF}(F, TF, type)) \
\land (\text{ContainmentNoSharingF}(F, TF, type)) \
\land (\text{InstMultsOkF}(F, TF, type)) \
\land (\text{InstContainmentForest}(F, TF, type)) \
\land (\text{FinMultsOkF}(F, TF, type)) \
\land (\text{FinMultsOkF}(F, TF, type)) \
\land (\text{FinMultsOkF}(F, TF, type)) \
\land (\text{FinMultsOkF}(F, TF, type))

FrTyConf \iff \{ FT : FrTy \mid \text{isConformableF}(FT) \}
B.10 Typed Models

**section** Fragmenta_TyMdl

\[ Tm_{\text{def}} \equiv \{ \text{GFG} : \text{GFGr}; \text{CG} : \text{CGr}; \text{fcl} : \text{GrMorph}; \text{fdef} : V \to TFr \mid \text{fcl} \in \text{morphGFGCG(GFG, CG)} \land \text{fdef} \in \text{gfgNs GFG} \to TFr \} \]

\[
\begin{align*}
\forall \text{GFG} : \text{GFGr}; \text{CG} : \text{CGr}; \text{fcl} : \text{GrMorph}; \text{fdef} : V \to TFr & \bullet \text{tmGFG(GFG, CG, fcl, fdef) = GFG} \\
\forall \text{GFG} : \text{GFGr}; \text{CG} : \text{CGr}; \text{fcl} : \text{GrMorph}; \text{fdef} : V \to TFr & \bullet \text{tmCG(GFG, CG, fcl, fdef) = CG} \\
\forall \text{GFG} : \text{GFGr}; \text{CG} : \text{CGr}; \text{fcl} : \text{GrMorph}; \text{fdef} : V \to TFr & \bullet \text{tmfcl(GFG, CG, fcl, fdef) = fcl} \\
\forall \text{GFG} : \text{GFGr}; \text{CG} : \text{CGr}; \text{fcl} : \text{GrMorph}; \text{fdef} : V \to TFr & \bullet \text{tmfdef(GFG, CG, fcl, fdef) = fdef} \\
\end{align*}
\]

\[
\begin{align*}
\forall \text{TM} : Tm_{\text{def}} \bullet \text{UTFs TM = UTFs}(\text{ran}(\text{tmfdef TM})) \\
\forall \text{TF} : TFr \bullet \text{UTFs}_0(\{ \text{TF} \}) = \text{TF} \\
\forall \text{TF} : TFr; \text{TFs} : \mathbb{P}_1 TFr \bullet \text{UTFs}(\{ \text{TF} \} \cup \text{TFs}) = \text{TF} \text{UTF}(\text{UTFs}_0 \text{TFs}) \\
\end{align*}
\]

\[
\begin{align*}
\forall \text{el} : V; \text{TM} : Tm_{\text{def}}; \text{vf} : V \bullet \text{fromVT}(\text{el}, \text{TM}) = \text{vf} \Leftrightarrow \text{el} \in \text{tfNs(}\text{tmfdef TM vf}) \\
\end{align*}
\]

\[
\begin{align*}
\text{consTFToGFG} : V \times Tm_{\text{def}} \to \text{GrMorph} \\
\text{consTFToGFR} : V \times \mathbb{P}_1 E \times Tm_{\text{def}} \to E \to E \\
\forall \text{vf} : V; \text{TM} : Tm_{\text{def}}; \text{fv} : V \to V; \text{fe} : E \to E \bullet \\
\text{consTFToGFG}(\text{vf}, \text{TM}) = (\text{fv, fe}) \Leftrightarrow (\exists \text{TF} : TFr; \text{GFG} : \text{GFGr} \bullet \\
\text{TF} = \text{tmfdef TM vf} \land \text{GFG} = \text{tmGFG TM} \land \text{fv} \in \text{tfNs TF} \to \text{gfgNs GFG} \\
\land \text{fe} \in \text{tfEs TF} \to \text{gfgEs GFG} \land \text{vf} \in \text{gfgNs GFG} \\
\land (\exists \text{ef} : \text{gfgEs GFG} \bullet \\
\text{src}(\text{gfgG GFG})\text{ef} = \text{tgt}(\text{gfgG GFG})\text{ef} = \text{vf} \land \text{fe} = \text{tfNs TF} \times \{ \text{vf} \} \\
\land \text{ef} = (\text{tfEs TF} \setminus \text{tfEsR TF} \times \{ \text{ef} \}) \cup \text{consTFToGFR}(\text{vf}, (\text{tfEsR TF}, \text{TM}))) \\
\forall \text{vf} : V; \text{TM} : Tm_{\text{def}}; \text{fe} : E \to E \bullet \text{consTFToGFR}(\text{vf}, \{ \} \text{, TM}) = \{ \} \\
\forall \text{vf} : V; \text{TM} : Tm_{\text{def}}; \text{el} : E; \text{Er} : \mathbb{P}_1 E; \text{fe} : E \to E \bullet \\
\text{consTFToGFR}(\text{vf}, (\{ \text{el} \} \cup \text{Er}), \text{TM}) = \text{fe} \Leftrightarrow (\exists \text{TF} : TFr; \text{GFG} : \text{GFGr} \bullet \\
\text{TF} = \text{tmfdef TM ef} \land \text{GFG} = \text{tmGFG TM} \\
\land (\exists \text{ef} : \text{gfgEs GFG} \bullet (\text{src}(\text{gfgG GFG})\text{ef} = \text{vf} \\
\land \text{tgt}(\text{gfgG GFG})\text{ef} = \text{fromVT}(\text{tfEsR (tfJF TF el)}, \text{TM}))) \\
\end{align*}
\]

72
\[ m_{\text{UTMToGFG}} : TMdl \rightarrow \text{GrMorph} \]
\[ \text{buildUTFsToGFG} : (V \rightarrow \text{TFr}) \times TMdl \rightarrow \text{GrMorph} \]

\[ \forall TM : TMdl_0; \, fv : V \rightarrow V; \, fe : E \rightarrow E \bullet \]
\[ m_{\text{UTMToGFG}} \, TM = (fv, fe) \iff \]
\[ (\exists \text{TF} : \text{TFr}; \, \text{TF} = \text{UTFs} \, TM \land (fv, fe) = \text{buildUTFsToGFG}(\text{tmfdef} \, TM), TM) \]

\[ \forall \text{vf} : V; \, \text{TF} : \text{TFr}; \, TM : TMdl_0; \bullet \]
\[ \text{buildUTFsToGFG} : \{(\text{vf} \rightarrow \text{TF})\}, TM) = \text{consTFToGFG} (\text{vf}, TM) \]

\[ TMdl = \{ TM : TMdl_0 \mid \exists \, m : \text{GrMorph} \bullet m = m_{\text{UTMToGFG}} \, TM \]
\[ \land \, m \in \text{morphFGFG} ((\text{tf}F(\text{UTFs} \, TM)), (\text{tmFGF} \, TM)) \]
\[ \land \, (\forall \text{vf}_1, \text{vf}_2 : \text{gfgNs}(\text{tmFGF} \, TM) \bullet (\text{vf}_1 \neq \text{vf}_2 \Rightarrow (fv \, m \setminus \{[\text{vf}_1]\}) \land (fv \, m \setminus \{[\text{vf}_2]\}) = \emptyset)) \} \]

\[ MdlTy = \{ M : Mdl; \, TM : TMdl; \, tcg, tgfg, ty : \text{GrMorph} \mid \]
\[ \exists FM : Fr; \, \text{FTM} : \text{TFr}; \, FM = \text{UTFs} \, M \land \text{FTM} = \text{UTFs} \, TM \]
\[ \land \, tcg \in \text{morphCG}(\text{tmcg} \, M), (\text{tmCG} \, TM)) \]
\[ \land \, tgfg \in \text{morphGFG}(\text{tmfclg} \, M), (\text{tmFGF} \, TM)) \]
\[ \land \, (FM, \text{FTM}, ty) \in \text{FrTyConf} \]
\[ \land \, tcg \circ_{tm} m_{\text{UTMToGFG}} \, M = m_{\text{UTMToGFG}} \, TM \circ_{tm} tcg \}

\text{B.11 Typed Models with Fragmentation Strategies}

section Fragmenta_TyFSMdl parents standard_toolkit, Fragmenta_TyFrs, Fragmenta_TyMdl

FSs = \{ SCG : CGr; \, SGFG : GFGr; \, scl, sgfg : \text{GrMorph} \mid \]
\[ \land \, \text{scl} \in \text{morphFGFCG} (\text{SGFG}, \text{SCG}) \} \]

fsCG : FSs \rightarrow CGr
fsGFG : FSs \rightarrow GFGr
\text{fscl} : FSs \rightarrow \text{GrMorph}
\text{fsmgfg} : FSs \rightarrow \text{GrMorph}

\[ \forall \text{SCG} : \text{CGr}; \, \text{SGFG} : \text{GFGr}; \, \text{mcl}, \text{mgfg} : \text{GrMorph} \bullet \]
\[ \text{fsCG} (\text{SCG}, \text{SGFG}, \text{mcl}, \text{mgfg}) = \text{SCG} \]

\[ \forall \text{SCG} : \text{CGr}; \, \text{SGFG} : \text{GFGr}; \, \text{mcl}, \text{mgfg} : \text{GrMorph} \bullet \]
\[ \text{fsGFG} (\text{SCG}, \text{SGFG}, \text{mcl}, \text{mgfg}) = \text{SGFG} \]

\[ \forall \text{SCG} : \text{CGr}; \, \text{SGFG} : \text{GFGr}; \, \text{mcl}, \text{mgfg} : \text{GrMorph} \bullet \]
\[ \text{fscl} (\text{SCG}, \text{SGFG}, \text{mcl}, \text{mgfg}) = \text{mcl} \]

\[ \forall \text{SCG} : \text{CGr}; \, \text{SGFG} : \text{GFGr}; \, \text{mcl}, \text{mgfg} : \text{GrMorph} \bullet \]
\[ \text{fsmgfg} (\text{SCG}, \text{SGFG}, \text{mcl}, \text{mgfg}) = \text{mgfg} \]
TFSMdl == \{ TM : TFSMdl; FS : FSs | (fsmfg FS) ∈ morphFGFG((tfF(UTFs TM)), fsGFG FS) \}

\begin{align*}
tfsmTM & : TFSMdl \rightarrow TM \\
tfsmFS & : TFSMdl \rightarrow FSs \\
tfsmscg & : TFSMdl \rightarrow CGr \\
tfsmshfg & : TFSMdl \rightarrow GFGF
\end{align*}

\forall TM : TFSMdl; FS : FSs \bullet tfsmTM(TM, FS) = TM
\forall TM : TFSMdl; FS : FSs \bullet tfsmFS(TM, FS) = FS
\forall TM : TFSMdl; FS : FSs \bullet tfsmscg(TM, FS) = fsCG FS
\forall TM : TFSMdl; FS : FSs \bullet tfsmshfg(TM, FS) = fsGFG FS

MdlTyFS == \{ M : Mdl; TM : TFSMdl; scg, sgfg, ty : GrMorph | scg ∈ morphCG((mcg M), (tfsmscg TM)) \∧ sgfg ∈ morphFGFG((mfgf M), (tfsmshfg TM)) \∧ (UTFs M, UTFs(tfsmTM TM), ty) ∈ FrTyConf \∧ sgfg ◦ G mUMToGFG M = fsmfg (tfsmFS TM) ◦ G ty \∧ scg ◦ G mfcl M = fsmcl(tfsmFS TM) ◦ G sgfg \}

B.12 Colimit Composition

section Fragmenta_Colimit_Composition parents standard_toolkit, Fragmenta_GraphsCat, Fragmenta_Mdls

emptyG : Gr
emptyG = (∅, ∅, ∅, ∅)

addNodeToGr : V × Gr → Gr
\forall v : V; G, G' : Gr \bullet addNodeToGr(v, G) = G' \iff G' = (Ns G ∪ \{v\}, Es G, src G, tgt G)

addEdgeToGr : E × V × V × Gr → Gr
\forall e : E; v1, v2 : V; G, G' : Gr \bullet addEdgeToGr(e, v1, v2, G) = G' \iff e \in Es G \lor \neg v1 \in Ns G \lor \neg v2 \in Ns G
\forall e : E; v1, v2 : V; G, G' : Gr \bullet addEdgeToGr(e, v1, v2, G) = G' \iff \neg e \in Es G \land v1 \in Ns G \land v2 \in Ns G
\land G' = (Ns G, Es G \cup \{e\}, src G \cup \{(e → v1)\}, tgt G \cup \{(e → v2)\})
emptyDiag : Cat \to \text{Diag}
\forall C : \text{Cat} \bullet \text{emptyDiag} \ C = (C, \text{emptyG}, (\emptyset, \emptyset))

addNodeToDiag : V \times O \times \text{Diag} \to \text{Diag}
\forall \text{vf} : V; \ A : O; \ D, D' : \text{Diag} \bullet \text{addNodeToDiag} (\text{vf}, A, D) = D
\forall \text{vf} : V; \ A : O; \ D, D' : \text{Diag} \bullet \text{addNodeToDiag} (\text{vf}, A, D) = D' \iff (\exists G' : \text{Gr}; \ m' : \text{MorphG2C} \bullet G' = \text{addNodeToGr} (\text{vf}, (\text{grD} D)) \land m' = (mV(\text{morph} D), mE(\text{morphD} D)) \cup \{(e \mapsto m)\} \land D' = (\text{cat} D, G, mD))

addEdgeToDiag : E \times V \times V \times M \times \text{Diag} \to \text{Diag}
\forall e : E; \ \text{vf}_1, \text{vf}_2 : V; \ m : M; \ D, D' : \text{Diag} \bullet
\text{addEdgeToDiag} (e, \text{vf}_1, \text{vf}_2, m, D) = D
\forall e : E; \ \text{vf}_1, \text{vf}_2 : V; \ m : M; \ D, D' : \text{Diag} \bullet \text{addEdgeToDiag} (e, \text{vf}_1, \text{vf}_2, m, D) = D' \iff (\exists G : \text{Gr}; \ mD : \text{MorphG2C} \bullet G = \text{addEdgeToGr} (e, \text{vf}_1, \text{vf}_2, (\text{grD} D)) \land mD = (mV(\text{morph} D), mE(\text{morphD} D) \cup \{(e \mapsto m)\})) \land D' = (\text{cat} D, G, mD))

buildStartDiag : V \times \text{Mdl} \to \text{Diag}
\forall \text{vf} : V; \ M : \text{Mdl}; \ D : \text{Diag} \bullet
\text{buildStartDiag} (\text{vf}, M) = \text{addNodeToDiag} (\text{vf}, (\text{OGrToGr} \ ~)(\text{fsrcGr} (\text{mfdef} M \ \text{vf})), \text{emptyDiag} \ \text{GraphsC})

diagDepNodes : \mathbb{P} V \times \text{Mdl} \times \text{Diag} \to \text{Diag}
\forall M : \text{Mdl}; \ D : \text{Diag} \bullet \text{diagDepNodes} (\{\}, M, D) = D
\forall \text{vfs} : \mathbb{P} V; \ \text{vf}_1 : V; \ M : \text{Mdl}; \ D, D' : \text{Diag} \bullet
\text{diagDepNodes} (\{\text{vf}_1 \cup \text{vfs}\}, M, D) = D' \iff (\exists D_0, D_1, D_2 : \text{Diag} \bullet D_0 = \text{addNodeToDiag} (\text{vf}_1, (\text{OGrToGr} \ ~)(\text{fsrcGr} (\text{mfdef} M \ \text{vf}_1)), D)
\land D_1 = \text{diagDepNodes} ((\text{\text{importsOf} (\text{vf}_1, (\text{\text{mfdef} M}) \cup \text{vfs})), M, D_0)
\land D_2 = \text{diagDepNodes} ((\text{\text{continuationsOf} (\text{vf}_1, (\text{\text{mfdef} M}) \cup \text{vfs})), M, D_1)
\land D' = \text{diagDepNodes} (\text{vfs}, M, D_2))

addMergeMorphisms : \text{Gr} \times \text{Mdl} \times \text{Diag} \times V \times \mathbb{P} V \to V
\forall GI : \text{Gr}; \ M : \text{Mdl}; \ D : \text{Diag}; \ v : V \bullet \text{addMergeMorphisms} (GI, M, D, v, \emptyset) = D
\forall GI : \text{Gr}; \ M : \text{Mdl}; \ D, D' : \text{Diag}; \ vs, vt : V; \ \text{vfs} : \mathbb{P} V \bullet
\text{addMergeMorphisms} (GI, M, D, vs, \{(vt) \cup \text{vfs}\}) = D' \iff (\exists vfs, vt : V; \ F : \text{Fr}; \ m, mM : \text{GrMorph}; \ e : E; \ D_0, D_1 : \text{Diag} \bullet
mM = mU\text{MToFGF} M \land vt = fV mM vt \land vfs = fV mM vs \land e \in \text{Es}(\text{grD} D)
\land F = \text{mfdef} M vft \land D_0 = \text{addNodeToDiag} (vft, (\text{OGrToGr} \ ~)(\text{fsrcGr} F), D)
\land m \in \text{morphG} (GI, (\text{\text{fsrcGr} F})) \land m = (\{\text{vs} \mapsto vt\}, \emptyset)
\land D_1 = \text{addEdgeToDiag} (e, \text{vfs}, \text{vf}, (\text{MGrToGr} M \ ~) m, D_0)
\land D' = \text{addMergeMorphisms} (GI, M, D, vs, \text{vfs})

75
relation(HasImpRefs_)

\[
\text{HasImpRefs}_ = \mathbb{P}(V \times V \times \text{Mdl})
\]
\[
\forall v_1, v_2 : V; \ M : \text{Mdl} \bullet (\text{HasImpRefs}(v_1, v_2, M)) \iff \exists F_1, F_2 : \text{Fr} \bullet F_1 = \text{mfdef } M \ v_1 \land F_2 = \text{mfdef } M \ v_2 \land \text{refs } F_1 \uparrow \text{fNs } F_2 \neq \emptyset
\]

\[
\text{diagRefs} : V \times \mathbb{P}(V \times V \times \text{Mdl}) \to \text{Diag}
\]
\[
\forall v : V; \ M : \text{Mdl}; \ D : \text{Diag} \bullet \text{diagRefs}(v, \emptyset, M, D) = D
\]
\[
\forall v_1, v_2 : V; \ \text{suf} : \mathbb{P}V; \ M : \text{Mdl}; \ D : \text{Diag} \bullet
\text{diagRefs}(v_1, (\{v_2\} \cup \text{suf}), M, D) = \text{diagRefs}(v_1, \text{suf}, M, D) \iff \neg \text{HasImpRefs}(v_1, v_2, M)
\]

\[
\text{diagMorphisms} : V \times \text{Mdl} \times \text{Diag} \to \text{Diag}
\]
\[
\text{diagMorphisms}_ = V \times \text{Mdl} \times \text{Diag} \times \mathbb{P}V \to \text{Diag} \times \mathbb{P}V
\]
\[
\text{diagMorphismsSet} : V \times \mathbb{P}(V \times V \times \text{Mdl}) \times \text{Diag} \times \mathbb{P}V \to \text{Diag} \times \mathbb{P}V
\]

\[
\forall v : V; \ M : \text{Mdl}; \ D, D' : \text{Diag} \bullet
\text{diagMorphisms}(v, M, D) = D' \iff (\exists p_{vfs} : \mathbb{P}V \bullet \text{diagMorphismsSet}(v, M, M, p_{vfs}))
\]
\[
\forall v : V; \ p_{vfs}, p_{vfs}' : \mathbb{P}V; \ M : \text{Mdl}; \ D, D' : \text{Diag} \bullet
\text{diagMorphismsSet}(v, M, D, p_{vfs}) = (D', p_{vfs}') \iff (\exists F : \text{Fr}; \ D_1 : \text{Diag} \bullet
F = \text{mfdef } M \ v_f \\
\land D_1 = \text{diagRefs}(v, \text{importsOf}(v, (\text{mgfg } M)) \cup \text{continuesOf}(v, (\text{mgfg } M)), M, D) \\
\land \text{diagMorphisms}(v, M, D, p_{vfs}) = \text{diagMorphismsSet}(v, M, D, p_{vfs}) \\
(p_{vfs} \cup \{v_f\}) = (D', p_{vfs}')
\]
\[
\forall p_{vfs} : \mathbb{P}V; \ M : \text{Mdl}; \ D : \text{Diag} \bullet \text{diagMorphismsSet}(\emptyset, M, D, p_{vfs}) = (D, p_{vfs})
\]
\[
\forall v : V; \ p_{vfs}, vfs : \mathbb{P}V; \ M : \text{Mdl}; \ D : \text{Diag} \bullet
\text{diagMorphismsSet}(\{v_f\} \cup \text{suf}, M, D, p_{vfs}) = \text{diagMorphismsSet}(vfs, M, D, p_{vfs}) \iff v \in p_{vfs}
\]
\[
\forall v : V; \ p_{vfs}, p_{vfs}', vfs : \mathbb{P}V; \ M : \text{Mdl}; \ D, D' : \text{Diag} \bullet
\text{diagMorphismsSet}(\{v_f\} \cup \text{suf}, M, D, p_{vfs}) = (D, p_{vfs}') \iff v \in p_{vfs} \land (\exists D'' : \text{Diag}; \ p_{vfs}'' : \mathbb{P}V \bullet \text{diagMorphismsSet}(v, M, D, p_{vfs}) = (D'', p_{vfs}'') \\
\land \text{diagMorphismsSet}(v, M, D'', p_{vfs}'') = (D', p_{vfs}''))
\]

76
\[ \text{compDiag} : V \times \text{Mdl} \rightarrow \text{Diag} \]

\[
\forall vf : V; M : \text{Mdl}; D : \text{Diag} \bullet \text{compDiag}(vf, M) = D \iff \\
(\exists D_0, D_1, D_2 : \text{Diag} \bullet D_0 = \text{buildStartDiag}(vf, M) \\
\wedge \text{diagDepNodes}((\text{importsOf}(vf, (\text{mgfg} M))), M, D_0) = D_1 \\
\wedge \text{diagDepNodes}((\text{continuesOf}(vf, (\text{mgfg} M))), M, D_1) = D_2 \\
\wedge \text{diagMorphisms}(vf, M, D_2) = D) \]