Towards a Theory of Views for Feature Models

Dave Clarke, José Proença
IBBT-DistriNet, Dept. Computer Sciences
Katholieke Universiteit Leuven, Belgium
Email: {dave.clarke,jose.proenca}@cs.kuleuven.be

***Abstract*—Variability in a Software Product Line (SPL) is expressed in terms of a feature model. As software development efforts involve increasingly larger feature models, scalable techniques are required to manage their complexity. Furthermore, as many stakeholders have a vested interest in different aspects of a feature model, modularity techniques are required to independently expresses their views of the product line abstracting away from unnecessary details. To address these issues of scalability and modularity this paper introduces a theory of views for features models, encompassing both view compatibility and view reconciliation for plugging together different views of a product line.**

***Keywords*-software product lines; variability; formal methods**

I. Introduction

Variability in a Software Product Lines (SPL) is expressed in terms of a feature model. Feature models are usually depicted using feature diagrams [1, 2], and a variety of increasingly expressive notations have been presented [3, 4, 5, 6, 7, 8, 9, 10]. The semantics of feature diagrams are generally understood in terms of sets of (multi)sets of features [11, 12], where at this level of abstraction a feature is simply a name. As software development efforts involve increasingly larger feature models, scalable techniques are required to manage their complexity, and modularity techniques are required to cater for the different interests of the various stakeholders involved in the project.

One approach to both managing complexity and increasing modularity is using views. A view of a feature model shows the decomposition of a concept up to a certain level of detail. With a view, only part of a feature model is visible to each stakeholder, possibly at with some details abstracted away. For example, one stakeholder may be interested in the user visible functionality, and thus does not need to see the detailed features required by programmers in order to to implement that functionality. Programmers in one team need not see the complete variability of the product line from the perspective of other teams.

A theory of views for features models is introduced, along with notions of view compatibility and view reconciliation, to address issues of scalability and modularity. Our work is presented quite generally, using notions from category theory, to ensure the future applicability of our results to more detailed models of software product lines. Our contributions include: a notion of view for feature models; criteria for determining whether views are compatible for a given SPL; and techniques for reconciling compatible views.

The paper is organised as follows. §II provides some motivating examples. §III introduces some preliminary concepts. §IV defines abstractions and views of feature models. §V presents our key definitions and results: view compatibility and view reconciliation and their characterisation. §VI takes a different angle, revealing some limitations in our approach. §VII applies our approach to express two notions of refinement of feature models. §VIII discusses related work. §IX presents our conclusions and future work.

II. Motivating Examples

We motivate our work with a collection of small examples, where features are simple consumer items:

$$\{tv, book, iPod, newspaper, cd\}. \tag{1}$$

In this paper we investigate on how to combine *views* on product lines. A view consists of a selection of relevant features combined with the abstraction of some other features into more general features. The term *reconciliation* refers to this combination of views, and we use a simple example to guide the intuition behind the reconciliation process. We start by considering views over the two sets:

$$F = \{Electronic, book, newspaper\} \tag{2}$$
$$G = \{tv, iPod, Paper\}. \tag{3}$$

In F we abstract tv and $iPod$ as $Electronic$, and in G we abstract $book$ and $newspaper$ as $Paper$. We do not use cd yet. As a convention, we start the name of a feature with an uppercase letter to denote it is an abstraction. A *product* is a set of features, and a product line is a set of possible products.

Notation. For simplicity, we use only the first letter of each feature using `true type` font, and denote products by the consecutive letters of its features. For example, `tiP` denotes the product $\{tv, iPod, Paper\}$.

This research is partly funded by the EU project FP7-231620 HATS: Highly Adaptable and Trustworthy Software using Formal Methods (http://www.hats-project.eu), and the K.U.Leuven BOF-START project STRT1/09/031 DesignerTypeLab.

Example 1 (Success story)**.** *We reconcile the following products lines, over F and G, respectively.*

$$P = \{\texttt{E}, \texttt{Eb}, \texttt{Ebn}\} \qquad Q = \{\texttt{t}, \texttt{tiP}\}.$$

The reconciliation of two products or product lines x and y is denoted by $x \oplus y$. P and Q can be reconciled only when they are compatible*. Two features a and b are compatible, written $a \frown b$, if one abstracts the other feature. In our example, $\texttt{t} \frown \texttt{E}$ but not $\texttt{t} \frown \texttt{P}$ or $\texttt{P} \frown \texttt{E}$. Two products are compatible if every feature from each product has a compatible feature in the other product. In our case, $\texttt{tiP} \frown \texttt{Eb}$ but not $\texttt{tiP} \frown \texttt{E}$. Finally, two product lines are compatible if every product from each product line has a compatible product in the other product line.*

Our two product lines P and Q are compatible, and the reconciliation $\oplus$ joins every possible combination of compatible products. The reconciliation of these product lines yield:

$$\begin{aligned} P \oplus Q &= \{\texttt{E} \oplus \texttt{t},\ \texttt{Eb} \oplus \texttt{tiP},\ \texttt{Ebn} \oplus \texttt{tiP}\} \\ &= \{\texttt{t},\ \texttt{tib},\ \texttt{tibn}\}. \end{aligned}$$

Example 2 (Partiality)**.** *We now include also the cd feature, which is neither refined or abstracted by any other feature.*

$$G_2 = G \cup \{cd\} \qquad Q_2 = \{\texttt{t}, \underline{\texttt{c}}\texttt{tiP}\}$$

When calculating $P \oplus Q_2$, the compatibility relation ignores the cd feature, while the reconciliation operator preserves it. In our example, $\texttt{ctiP} \frown \texttt{Eb}$, and $\texttt{ctiP} \oplus \texttt{Eb} = \texttt{ctib}$. Thus, $P \frown Q_2$ and $P \oplus Q_2 = \{\texttt{t}, \underline{\texttt{c}}\texttt{tib}, \underline{\texttt{c}}\texttt{tibn}\}$.

Example 3 (Incompatibility)**.** *We now extend Q with $Paper$.*

$$Q_3 = \{\texttt{t}, \texttt{tiP}, \texttt{P}\}$$

In this example, $P \not\frown Q_3$, because there is no product $p \in P$ for which $p \frown Paper$.

III. Preliminaries

We start by introducing basic mathematical models of concepts such as features, product lines, software product lines, following Schobbens et al. [13], along with other relevant formal definitions.

A. Features, Products, and Software Product Lines

A *feature* is a unit of variability in a software product line relevant to some stakeholder [14]. From a formal perspective, features are represented as elements of a set. *Products* and *product lines* are defined as a set of features and as a set of products, respectively. Modelling from the level of abstraction of features, we thus equate a product with a feature configuration and a product line with a feature model.

Typically $F, F', G, G', \ldots$ range over sets of features, $a, b, \ldots$ over features. $p, q, \ldots$ over products, and $P, Q, \ldots$ over product lines. The notation $\mathcal{P}(F)$ denotes the power set of F, namely, the set of subsets of F.

Definition 4 (Product)**.** *Given a set of features F. A* product *is defined as a subset of F. The set of all products for features F is denoted $\overline{F}$ and is defined*

$$\overline{F} \stackrel{\text{def}}{=} \mathcal{P}(F). \tag{4}$$

Definition 5 (Product Line)**.** *Given a set of features F. A* product line *is defined as a set of products for F, that is, a subset of $\overline{F}$. The set of all product lines for features F is denoted $\widehat{F}$ and defined*

$$\widehat{F} \stackrel{\text{def}}{=} \mathcal{P}(\overline{F}) = \mathcal{P}(\mathcal{P}(F)). \tag{5}$$

B. Function Lifting

Our results will be phrased using some basic category theory [15]; we will be working in the categories **Set** of sets and functions, and **PFun** of sets and partial functions.

Given a partial function $f : F \rightharpoonup G$, the domain of f is $\mathrm{dom}(f) = F$, the codomain is $\mathrm{cod}(f) = G$, the pre-image is $\mathrm{def}(f) = \{a \in F \mid f(a) \neq \bot\}$, and the range is $\mathrm{rng}(f) = \{f(a) \mid a \in \mathrm{def}(f)\}$. A function $f : F \to G$ is *total* if $\mathrm{dom}(f) = \mathrm{def}(f)$, and is *onto* if $\mathrm{cod}(f) = \mathrm{rng}(f)$. The composition of two partial functions $u \circ v$ is defined as:

$$(u \circ v)(a) \stackrel{\text{def}}{=} \begin{cases} \bot, & \text{if } v(a) = \bot \\ u(v(a)), & \text{otherwise.} \end{cases}$$

A partial function $f : F \rightharpoonup G$ from one feature set to another can be *lifted* to functions over products and product lines, written as $\overline{f}$ and $\widehat{f}$, respectively.

Definition 6 (Lifting)**.** *Given a partial function $f : F \rightharpoonup G$, we define the lifting of f to products and product lines as $\overline{f}$ and $\widehat{f}$, respectively, defined as follows.*

$$\begin{aligned} \overline{f} &: \overline{F} \to \overline{G} \\ \overline{f}(p) &\stackrel{\text{def}}{=} \{f(a) \mid a \in p \cap \mathrm{def}(f)\} \qquad (6) \\ \widehat{f} &: \widehat{F} \to \widehat{F'} \\ \widehat{f}(P) &\stackrel{\text{def}}{=} \{\overline{f}(p) \mid p \in P\}. \qquad (7) \end{aligned}$$

Another useful notion is the *restriction* of a product or product line to a smaller set of features. Restriction is one of the ingredients of our definition of view.

Definition 7 (Restriction)**.** *Let $G \subseteq F$, $p \in \overline{F}$, and $P \in \widehat{F}$. Define the restriction of p to G, and the restriction of P to G, denoted as $p \restriction G$ and $P \restriction G$, respectively, as follows:*

$$p \restriction G \stackrel{\text{def}}{=} p \cap G \tag{8}$$

$$P \restriction G \stackrel{\text{def}}{=} \{p \restriction G \mid p \in P\}. \tag{9}$$

An equivalent definition of restriction involves lifting a partial function $r : G \rightharpoonup G$, such that $r(a) = a$ for all $a \in \mathrm{def}(r)$, to products and product lines.

IV. Abstractions and Views

Abstraction removes distinctions between features, e.g., mapping all different varieties of television to a single feature TV. Views are then defined on top of notion of abstraction, enabling not only the merging of distinctions, but also the hiding of features. Both concepts can be expressed for features and lifted to products and product lines.

Definition 8 (Abstraction)**.** *Any onto function $f : F \to G$ defines an* abstraction $\widehat{f} : \widehat{F} \to \widehat{G}$*. A product line $Q : \widehat{G}$ can be seen as an abstraction of product line $P : \widehat{F}$ if an $f : F \to G$ exists such that $\widehat{f}(P) = Q$. Similarly, an abstraction of products for such an $f : F \to G$ is defined as $\overline{f} : \overline{F} \to \overline{G}$.*

Define a *view function* as partial abstraction.

Definition 9 (View)**.** *For any partial and onto function $g : F \rightharpoonup G$, we say that g is a* feature view*, $\overline{g}$ is a* product view*, and $\widehat{g}$ is a* product line view*. Furthermore, for any $P \in \widehat{F}$ we say that $\widehat{g}(P) \in \widehat{G}$ is a view on $\widehat{G}$. Similarly, a view of products for such an $g : F \rightharpoonup G$ is defined as $\overline{g} : \overline{F} \to \overline{G}$.*

Example 10. *Consider a product line with features $F = \{sonyTV, philipsTV, sonyDVD, philipsDVD, \ldots\}$. The following function $v : F \to G$, where $G = \{TV, sonyDVD, philipsDVD\}$, defines a view for a stakeholder interested only in the DVD variability and whether a TV is present or not:*

$$f = \begin{cases} sonyTV, philipsTV & \mapsto \; TV \\ sonyDVD & \mapsto \; sonyDVD \\ philipsDVD & \mapsto \; philipsDVD. \end{cases}$$

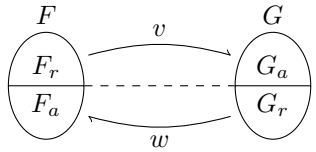
V. View Compatibility and Reconciliation

We now consider the following problem. Two views of a feature model are developed independently by two stakeholders. No globally definitive feature model exists or perhaps it has become out-dated. The stakeholders want to check whether their views are compatible and if so they want to see how the combined or reconciled view looks.

Assume the stakeholders are working with feature sets F and G. If the views are compatible, then it will be the case that some of the features in F are abstractions of features in G, and vice versa. It may also be that some features in F do not appear in G, and vice versa. This means that we can partition F and G each into two parts and construct a view between pairs of partitions, one going in each direction. The desired structure is called a *view partition* and is illustrated below. Here F_r and F_a are a partition of F. Similarly for G. The functions v and w are the view functions, mapping more refined elements in F_r to their abstraction in G_a, and from G_r to F_a, respectively.[1] We depict a view partition below, where v abstract from F_r to G_a, and and w from G_r to F_a.

[1]Subscripts r and a indicate the source set (more refined features) and target set (more abstract features) of view functions of a view partition.

σ =



Definition 11 (View partition)**.** *Define a* view partition *to be a tuple $\sigma = (F_r, F_a, G_r, G_a, v, w)$, where F_r, F_a, G_r, G_a are sets of features such that $F = F_r \cup F_a$, $G = G_r \cup G_a$, $F_r \cap F_a = \emptyset = G_r \cap G_a$, and $v : F_r \rightharpoonup G_a$ and $w : G_r \rightharpoonup F_a$ are partial onto functions.*

The set $F_r + G_r$ consists of the most refined features of F and G ($+$ is categorical coproduct, i.e., disjoint union). Reconciliation of two views will produce a product line in terms of $F_r + G_r$. On the other hand, $F_a + G_a$ is the set of the most abstract features. These are used for checking the compatibility of two views.

Define $v_r : F_r + G_r \rightharpoonup G$ and $v_a : F \rightharpoonup F_a + G_a$ to be the partial onto functions that extend v with the identity for elements in G_r and F_a, respectively. That is, for example,

$$v_r(x) \stackrel{\text{def}}{=} \begin{cases} v(x), & \text{if } x \in F_r \\ x, & \text{otherwise.} \end{cases}$$

Define w_r and w_a analogously, swapping F and G.

These functions play an important role in what follows. For example, function $v_r : F_r + G_r \rightharpoonup G$ presents the view of the reconciled product line in terms of G. Function $v_a : F \rightharpoonup F_a + G_a$ recasts a view based on features F in terms of the most abstract features, namely $F_a + G_a$.

A. View Compatibility

Given a view partition, the following definition captures what it means for the underlying views to be compatible for features, products and product lines.

Definition 12 (Compatibility)**.** *For any $a \in F$, $p \in \overline{F}$, $P \in \widehat{F}$, $b \in G$, $q \in \overline{G}$, and $Q \in \widehat{G}$, and for a view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$, define* compatibility *of features, products, and product lines, denoted by the binary operator $\frown$, as follows:*

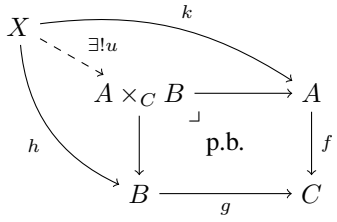
$$\begin{aligned} a \frown b &\stackrel{\text{def}}{=} v_a(a) = w_a(b) \\ p \frown q &\stackrel{\text{def}}{=} \overline{v_a}(p) = \overline{w_a}(q) \\ P \frown Q &\stackrel{\text{def}}{=} \widehat{v_a}(P) = \widehat{w_a}(Q). \end{aligned}$$

Observe that $v_a(a) = w_a(b)$ is equivalent to the more intuitive condition $v(a) = b \vee w(b) = a \vee v(a) = \bot = w(b)$, which states that b is an abstract feature corresponding to a, or vice versa, or both a and b only appear in F_r and G_r, respectively. Now, $p \frown q$ states that every feature in p has a compatible feature in q, and vice versa, and similarly, $P \frown Q$ states that every product in P has a compatible product in Q, and vice versa.

The key role of compatibility is that it provides the conditions for ensuring that two views can be reconciled.

B. Interlude: Pullbacks

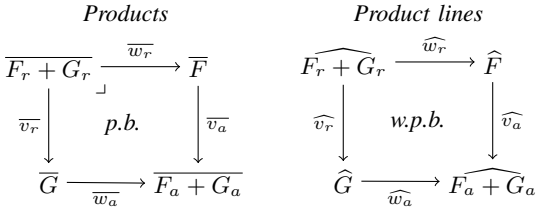
The key machinery to get reconciliation to work is the categorical notion of a pullback [15]. A *pullback* in the category **Set** of a pair of functions $f : A \to C$ and $g : B \to C$ is a the set $A \times_C B = \{(a, b) \in A \times B \mid f(a) = g(b)\}$, in other words, the pairs of elements of A and B that agree (via f and g) in C. This is the canonical definition of pullback—any isomorphic set can also be the basis of a pullback. The key property of a pullback is that if there are functions h and k that make the outer 'square' of the diagram below commute (that is, $k \circ f = h \circ g$), then there exists a unique function u that makes the whole diagram commutes. This captures that $A \times_C B$ is the most detailed of such objects.



A *weak pullback* drops the uniqueness criteria.

Although pullbacks always exist in **Set**, we need to select specific objects as the locus of reconciliation, namely, $\overline{F_r + G_r}$ and $\widehat{F_r + G_r}$, and cannot always work with pullbacks. Theorem 13 describes when these objects are pullbacks (of the appropriate functions) and when we can only work with a weak pullback.

Theorem 13. *Let $\sigma = (F_r, F_a, G_r, G_a, v, w)$ be a view partition. The first diagram is a pullback, and the second is a weak pullback.*



We now analyse how to combine two views $P \in \widehat{F}$ and $Q \in \widehat{G}$ for two feature sets F and G. We start at the level of features and build up to product lines. In the following, let $\sigma = (F_r, F_a, G_r, G_a, v, w)$ be a view partition.

C. Feature reconciliation

We define reconciliation of features by splitting into the cases where $a \in F_r$ and $b \in G_r$.

Definition 14 (Feature reconciliation)**.** *Given $a \in F$ and $b \in G$, define:*

$$\oplus_\sigma \;:\; F \times G \rightharpoonup F_r + G_r$$

$$a \oplus_\sigma b \stackrel{\text{def}}{=} \begin{cases} a & \text{if } a \in F_r \text{ and } b \notin G_r \\ b & \text{if } a \notin F_r \text{ and } b \in G_r \\ \bot & \text{otherwise.} \end{cases}$$

From now on we drop references to the view partition σ. When a and b are compatible, and at least one is in the domain of v or w, then a refines b or vice-versa. The reconciliation choses the most refined feature. The split into these three cases is justified by the following lemma.

Lemma 15. *Let $a \in F$ and $b \in G$, such that $a \in \text{def}(v)$ or $b \in \text{def}(w)$. If $a \frown b$ then $a \in F_r \Leftrightarrow b \notin G_r$.*

D. Product Reconciliation

Reconciliation for features can be lifted to products, by selecting the most refined features and discarding the rest.

Definition 16 (Product reconciliation)**.** *Given a view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$, and products $p \in \overline{F}$ and $q \in \overline{G}$ such that $p \frown q$, define:*

$$\oplus \;:\; \overline{F} \times \overline{G} \to \overline{F_r + G_r}$$

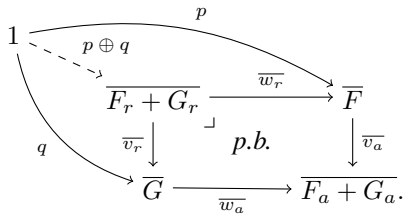
$$p \oplus q \stackrel{\text{def}}{=} (p \upharpoonright F_r) \cup (q \upharpoonright G_r).$$

The following is used to prove our characterisation property.

Lemma 17. $p \frown q \;\Rightarrow\; \overline{w_r}(p \oplus q) = p$ *and* $\overline{v_r}(p \oplus q) = q$.

The following corollary is justified by Lemma 17 and from the fact that $\overline{F_r + G_r}$ is the pullback. This corollary characterises product reconciliation (Definition 16). The 'outer square' corresponds precisely to the compatibility condition (Definition 12). The fact that the inner square is a pullback guarantees the uniqueness of the function $1 \to \overline{F_r + G_r}$, which can be shown to be $p \oplus q$, the uniquely selected element of $\overline{F_r + G_r}$.

Corollary 18. *Given view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$. If $p \frown q$, then $p \oplus q$ is the unique morphism making the following diagram commute:*



E. Product Line Reconciliation

We now lift the previous results for products and define the reconciliation of product lines as follows.

Definition 19 (Product line reconciliation)**.** *Given a view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$, $P \in \widehat{F}$, and $Q \in \widehat{G}$,define:*

$$\oplus \;:\; \widehat{F} \times \widehat{G} \to \widehat{F_r + G_r}$$

$$P \oplus Q \stackrel{\text{def}}{=} \{p \oplus q \mid p \in P, q \in Q, p \frown q\}.$$

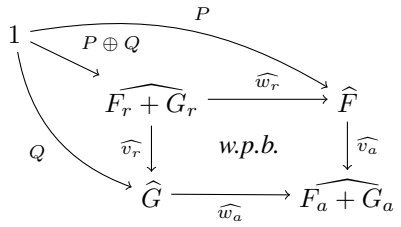
This operation takes all compatible pairs of products from each product line and combines them with the product

reconciliation operation. The imposition that $P \frown Q$ is a sufficient condition to guarantee that, after reconciling P and Q, the original values of P and Q can still be recovered, as stated by the following lemma.

Lemma 20. *$P \frown Q \Rightarrow \widehat{w_r}(P \oplus Q) = P$ and $\widehat{v_r}(P \oplus Q) = Q$.*

Furthermore, when the above condition does not hold, it is not possible in general to recover the values of P and Q from $P \oplus Q$, meaning that the P and Q would not be views on $P \oplus Q$. The following corollary is justified by Lemma 20 and that $\widehat{Fs + Gs}$ is a weak pullback.

Corollary 21. *Given view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$. If $P \frown Q$, then $P \oplus Q$ makes the diagram commute:*



This construction is somewhat unsatisfying, as it does not say conclusively that operation for product line view reconciliation is the best we can hope for. Before presenting an alternative characterisation, we take a step back to consider how our constructions look for views of a feature, product or product line, based on some given view partition.

Define the following functions:

$$c : F_r + G_r \to F_r + G_r \qquad a \mapsto w_r(a) \oplus v_r(a) \tag{10}$$

$$\overline{c} : \overline{F_r + G_r} \to \overline{F_r + G_r} \qquad p \mapsto \overline{w_r}(p) \oplus \overline{v_r}(p) \tag{11}$$

$$\widehat{c} : \widehat{F_r + G_r} \to \widehat{F_r + G_r} \qquad P \mapsto \widehat{w_r}(P) \oplus \widehat{v_r}(P) \tag{12}$$

The following state the relationship between a feature, product or product line and the views on it for a given view partition. Firstly, the views are compatible and secondly, reconciliation works on the nose for features and products but provides an over approximation for product lines.

Lemma 22. *Given $a \in F_r + G_r$, $p \in \overline{F_r + G_r}$ and $P \in \widehat{F_r + G_r}$. Then:*

1) $w_r(a) \frown v_r(a)$
2) $\overline{w_r}(p) \frown \overline{v_r}(p)$
3) $\widehat{w_r}(P) \frown \widehat{v_r}(P)$
4) $c(a) = a$
5) $\overline{c}(p) = p$
6) $\widehat{c}(P) \supseteq P$.

The following properties capture the nature of $\widehat{c}$, and thus provide some intuition about product lines reconciliation .

Lemma 23. *$\widehat{c}$ is a closure operator.*[2]

The following definition captures the equivalence class of product lines that have the same views in $\widehat{F}$ and in $\widehat{G}$.

[2](1) $P \subseteq \widehat{c}(P)$. (2) $P \subseteq Q$ implies $\widehat{c}(P) \subseteq \widehat{c}(Q)$. (3) $\widehat{c}(\widehat{c}(P)) = \widehat{c}(P)$.

Definition 24. *Define equivalence $\sim \subseteq \widehat{F_r + G_r} \times \widehat{F_r + G_r}$ as $P \sim Q \iff \widehat{w_r}(P) = \widehat{w_r}(Q) \wedge \widehat{v_r}(P) = \widehat{v_r}(Q)$.*

The following lemma characterises $\widehat{c}$ as selecting the maximum of all possible equivalent product lines with the same views in $\widehat{F}$ and $\widehat{G}$.

Lemma 25. *$\widehat{c}(P)$ is the maximum (wrt set inclusion) of $[P]$, the equivalence class containing P.*

VI. Constructing a View Partition

In the previous section, we assumed a view partition was given, though not necessarily the product line. Now we start from the other direction and address whether it is possible to find an appropriate view partition given a product line and two views. Let $P \in \widehat{E}$ be a product line, and $\widehat{f} : \widehat{E} \to \widehat{F}$ and $\widehat{g} : \widehat{E} \to \widehat{G}$ define two views in $\widehat{F}$ and $\widehat{G}$. The following condition guarantees the existence of a view partition.

Definition 26 (View compatibility)**.** *Two view functions $f : E \rightharpoonup F$ and $g : E \rightharpoonup G$ are compatible, denoted $f \frown g$,* iff *for every $a \in F$ and $b \in G$*

$$f^{-1}(a) \ \# \ g^{-1}(b),$$

where $X \# Y$ iff *$X \cap Y = \emptyset$, $X \subseteq Y$, or $Y \subseteq X$.*

Intuitively, Definition 26 states that f and g treat the features in E in terms of compatible abstractions. That is, two feature sets $E_a = f^{-1}(a)$ and $E_b = f^{-1}(b)$ are either incompatible, that is, $f^{-1}(a) \cap g^{-1}(b) = \emptyset$, or that one is an abstraction of the other $f^{-1}(a) \subseteq g^{-1}(b)$, or vice versa.

The following theorem states that when two views functions are compatible, then a view partition exists to make the two views of the original product line compatible.

Theorem 27. *Let $\widehat{f} : \widehat{E} \to \widehat{F}$ and $\widehat{g} : \widehat{E} \to \widehat{G}$ be two views. If $f \frown g$, then there exists a view partition σ such that $\forall P \in \widehat{E} \cdot \widehat{f}(P) \frown_\sigma \widehat{g}(P)$. Specifically, the following gives such a view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$, where $R = g \circ f^{-1}$:*

$$F_a \stackrel{\text{def}}{=} \{x \in F \mid |R(x)| > 1\} \qquad F_r \stackrel{\text{def}}{=} F \setminus F_a$$
$$G_a \stackrel{\text{def}}{=} R(F_r) \qquad G_r \stackrel{\text{def}}{=} G \setminus G_a$$
$$v : F_r \rightharpoonup G_a \stackrel{\text{def}}{=} \{R(x) \mid x \in F_r\}$$
$$w : G_r \rightharpoonup F_a \stackrel{\text{def}}{=} \{R^{-1}(x) \mid x \in G_r\}.$$

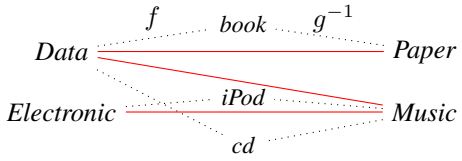
The relationship between the original product line $P \in \widehat{E}$ and the reconciled views $\widehat{c}(P)$ has already been described in Lemmas 22, 23 and 25.

Ultimately, one would expect all pairs of views to be compatible. That they are not points to a weakness in our theory. The problem stems from the fact that the two incompatible views have different 'vocabularies'.

Example 28 (Overlapping views)**.** *Given features $\{tv, book, iPod, newspaper, cd\}$. Consider view functions:*

$$f = \{cd, book \mapsto Data\,;\ iPod \mapsto Electronic\}$$
$$g = \{cd, iPod \mapsto Music\,;\ book \mapsto Paper\}$$

Observe that $f \pitchfork g$, because $f^{-1}(Data) \# g^{-1}(Music)$ does not hold. Consequently, there is no view partition guaranteeing the compatibility of product lines obtained by f and g. The cause is the following conflict:



$Data$ is neither an abstraction of $Music$ nor vice versa, thus the calculations in Theorem 27 fail to yield a view partition.

VII. Applications

Two applications demonstrate the expressiveness of our approach. Although the key ingredient is an abstraction function, view reconciliation can express notions of refinement:

- **Uniform refinement**: replace a feature by a collection of products.
- **Refinement in context**: replaces a feature a by a collection of products based on the other features occurring with a in a product.

These could allow feature models to be developed in a more scalable and modular fashion.

A. Uniform refinement

First we describe this operation based on sets. We will define only a simple variant which refines only one feature, and conjecture that it generalises in a straightforward fashion.

Uniform refinement of a feature f by a set of products Q, consists of taking each product containing f and replacing f by product p, for each $p \in Q$. In feature diagrams this corresponds to replacing a leaf feature f by a whole feature diagram whose underlying feature model is Q.

Definition 29 (Uniform Refinement)**.** *Given product line $P \in \widehat{F}$, $f \in F$, and $Q \in \widehat{G}$, where $F \cap G = \emptyset$. Define:*

$$UniR(P, f, Q) \stackrel{\text{def}}{=} \{p[f \mapsto q] \mid p \in P, q \in Q\} \tag{13}$$

$$p[f \mapsto q] \stackrel{\text{def}}{=} \begin{cases} p \setminus \{f\} \cup q, & \text{if } f \in p \\ p, & \text{otherwise.} \end{cases} \tag{14}$$

The resulting refinement is a product line over features $F \backslash \{f\} \cup G$. This refinement can be achieved when $\emptyset \notin Q$ using reconciliation, as follows.

$$UniR'(P, f, Q) \stackrel{\text{def}}{=} P \oplus_\sigma Q_\emptyset \tag{15}$$

$$Q_\emptyset = Q \cup \{\emptyset\} \tag{16}$$

$$\sigma = (F_r, F_a, G_r, G_a, v, w) \tag{17}$$

where $F_r = F \backslash \{f\}$, $G_r = \bigcup_{q \in Q} q$, $F_a = \{f\}$, $G_a = \emptyset$, $v = \emptyset$, $w = \{g \mapsto f \mid g \in G_r\}$.

The view partition σ is defined such that the to-be-replaced feature f is in F_a as the abstraction of the features in Q. The remaining features from F are kept in F_r and are not abstracted by v. Consequently, the features in F_r are preserved during the reconciliation. Correctness is given by the following lemma.

Lemma 30. $UniR(P, f, Q) = UniR'(P, f, Q)$.

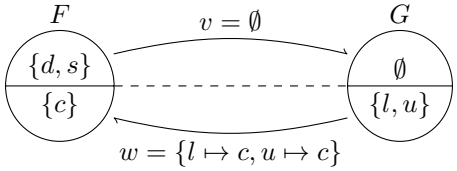
Example 31. *Consider the following product line, where we write abc to represent the product $\{a, b, c\}$:*

$$Pwd = \{c, cd, s\}$$

Pwd describes the features of a password field. A password can have characters (feature c), digits (feature d), and symbols (s), according to the combinations in Pwd. We apply the uniform refinement

$$(c, \{u, lu\})$$

which refines characters as lowercase and uppercase letters (features l and u, respectively), and requires uppercase letters. The view partition used to calculate this refinement is:



Finally, the refinement is calculated as follows.

$$\begin{aligned} & UniR(Pwd, c, \{l, u, lu\}) \\ = \; & \{c, cd, s\} \oplus \{u, lu, \emptyset\} \\ = \; & \{c \oplus u, c \oplus lu, cd \oplus u, cd \oplus lu, s \oplus \emptyset\} \\ = \; & \{u, lu, du, dlu, s\}. \end{aligned}$$

B. Refinement in Context

We now describe a more complex notion of refinement, where a feature f is replaced by a set of possible products depending on the context, namely the features in the same product as f, in which f appears. Refinement in context is given by an operation $RiC(P, f, k)$, where $k : \overline{C} \to \widehat{G}$, can be understood as taking the product line P and refining it by replacing f by the product $k(c)$, whenever f appears together with context $c \subseteq \overline{C}$.

Definition 32 (Refinement in Context)**.** *Given a product line $P \in \widehat{F}$, define* refinement in context *wrt $f \in F$ and $k : \overline{C} \to \widehat{G}$) by:*

$$RiC(P, f, k) = \{p[f \mapsto q] \mid p \in P, q \in k(p \cap C)\} \tag{18}$$

where $f \in F$, $f \notin C \subseteq F$, and $F \cap G = \emptyset$.

We encode the same refinement as a reconciliation as follows, for the cases where $\emptyset \notin \mathrm{rng}(k)$:

$$RiC'(P, f, k) \stackrel{\text{def}}{=} P \oplus_\sigma Q_k \tag{19}$$

$$Q_k = \{c \cup g \mid (c, g) \in k\} \cup \widehat{C} \tag{20}$$

$$\sigma = (F_r, F_a, G_r, G_a, v, w) \tag{21}$$

where $F_r = F \backslash F_a$ $\quad G_r = G \cup C$
$F_a = \{f\} \cup C$ $\quad G_a = \emptyset$
$v = \emptyset$ $\quad w = \{g \mapsto f \mid g \in G\} \cup \{c \mapsto c \mid c \in C\}$.

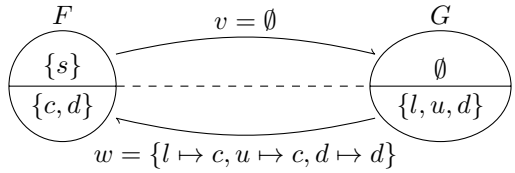
Correctness is given by the following lemma:

Lemma 33. $RiC(P, f, k) = RiC'(P, f, k)$.

Example 34. *Recall the password scenario from Example 31.*

$$Pwd = \{c, cd, s\}$$

where $Pwd \in \widehat{F}$*, and* $F = \{c, d, s\}$*. We now apply a refinement that replaces the characters feature* (c) *by lowercase* (l) *or uppercase* (u)*, depending on the presence of the digits feature* (d)*. Define* $G = \{l, u, d\}$ *and* $C = \{d\}$ *to be the target features and the context features, respectively. We apply the refinement in context based on* k *defined as* $k(\emptyset) = \{lu\}$*, and* $k(d) = \{l, u, lu\}$*. Here* k *represents how the context influences the replacement of* f*: in any context* f *can be replaced by the product* lu*, and when* d *is present* f *can also be replaced by* l *or* u*. That is, when there are no digits in a password there must be both lowercase and uppercase letters. The view partition used to calculate the refinement is depicted below.*



The refinement is encoded as follows.

$$\begin{aligned} & Ric(Pwd, c, k) \\ = \ & \{c, cd, s\} \oplus \{ld, ud, lu, lud, d, \emptyset\} \\ = \ & \{c \oplus lu, cd \oplus ld, cd \oplus ud, cd \oplus lud, s \oplus \emptyset\} \\ = \ & \{lu, ld, ud, lud, s\} \end{aligned}$$

In both this application and the previous one, we had to treat specially the case where a feature could be replaced by the empty set, in effect, that it could be removed from a product line. This hiccough would be placed under the hood of any tool implementing our ideas.

VIII. Related Work

Griss [7] briefly mentions the advantage of having different views on a feature model, where views are feature diagrams that display different levels of detail. A more radical insight is that "different stakeholders perceive differently what is variable" [16]. By considering developers and customers as the two main stakeholders, Pohl et al. [16] introduce the concepts of internal and external variability: external variability is visible to the customer while internal variability is hidden. Internal variability often represents finer-grained variation points at lower levels of abstraction.

Höfner et al. [17] formalise software product lines using the *feature algebra* model, and also describe reconciliation. Product lines are semirings with some extra properties, where features and products differ from product lines only on the properties they obey. The authors use the concrete example of sets and multisets of features to describe products, and sets of products to describe product lines, although other examples also fit their general formalisation. An abstraction of a product line can remove references to features and add new products. For example, the reduction $\{\mathtt{t}, \mathtt{tibn}\} \rightsquigarrow \{\mathtt{t}, \mathtt{tiP}\}$ from Example 1 is an abstraction only in our setting, while $\{\mathtt{t}, \mathtt{tibn}\} \rightsquigarrow \{\mathtt{t}, \mathtt{tibn}, \mathtt{tP}\}$ and $\{\mathtt{t}, \mathtt{tibn}\} \rightsquigarrow \{\emptyset, \mathtt{P}\}$ are abstractions only in feature algebra.

In feature algebra a view is just a product line, and reconciliation of views is achieved by combining all possible products and filtering the result using a given set of requirements. Thus, reconciliation is guided by extra requirements, while in our approach reconciliation is guided by the view partition between features of the reconciled views. We explore compatibility of views, disregarded in Höfner et al.'s approach, and allow developers of different views to refer to simplified versions of each other's views.

Existing work on views is generally presented at a different level of abstraction than our approach, such as architectural views and views on other software models [18]. Solomon [19] proposes using pushouts to merge architectures when different software systems need to be merged. Other approaches use pushouts and pullbacks and other algebraic techniques for model synchronisation [20] and version control in model-driven engineering [21]. Curiously, our approach uses pullbacks and not pushouts for combining models.

Acher et al. [22] present a technique for composing feature models out of smaller ones. Their work focusses mostly on composing feature diagrams, though some operations at the semantic level (of sets of sets of features) are provided. Our work focusses exclusively on the underlying semantics and we provide a much richer theory. Segura et al. [23] use graph transformations for merging feature models, at the diagrammatic level. Their merging is akin to our view reconciliation, but by performing transformations on diagrams, they also lack a clear formal semantics.

Bowman et al. [24] present a framework for viewpoint consistency. Their framework covers many aspects of software models, though not feature models. Recent work in this line [25] considers model transformations across different views, where the views are represented by different kinds of models (state machines, class diagrams, etc). In our setting we work with only one kind of model.

IX. Conclusion and Future Work

This paper presents a semantic perspective on views for features models. Views enable feature models to be developed in a more modular fashion, where each stakeholder can have independent perspectives on a product line. Our theory provides a means for checking the compatibility of different views and for the reconciliation of compatible views.

For future work we will determine the constraints on view partitions to ensure that our techniques can apply to three or more views. We also plan to apply our techniques to more complex models, such as behavioural ones, underlying software product lines, and to implement our ideas in the developing HATS ABS toolkit [26].

References

[1] K. C. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, "Feature-Oriented domain analysis (FODA) feasibility study," Carnegie Mellon University Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-021, 1990.

[2] D. Batory, D. Benavides, and A. Ruiz-Cortes, "Automated analysis of feature models: challenges ahead," *Commun. ACM*, vol. 49, no. 12, pp. 45–47, 2006.

[3] D. Batory, "Feature models, grammars, and propositional formulas," in *Software Product Lines*. Springer-Verlag, 2005, pp. 7–20.

[4] K. Czarnecki and U. Eisenecker, *Generative programming*. Addison Wesley, 2000.

[5] J. V. Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *WICSA '01*. IEEE Computer Society, 2001, p. 45.

[6] M. Eriksson, J. Börstler, and K. Borg, "The PLUSS approach - domain modeling with features, use cases and use case realizations," *Software Product Lines*, pp. 33–44, 2005.

[7] M. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," in *Software Reuse, 1998. Proceedings. Fifth International Conference on*, 1998, pp. 76–85.

[8] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker, "Generative programming for embedded software: An industrial experience report," in *Generative Programming and Component Engineering*. Springer-Verlag, 2002, pp. 156–172.

[9] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," in *Software Product Lines*. Springer-Verlag, 2004, pp. 162–164.

[10] M. Riebisch, "Towards a more precise definition of feature models," in *Modelling Variability for Object-Oriented Product Lines*, 2003, pp. 64–76.

[11] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps, "Generic semantics of feature diagrams," *Computer Networks*, vol. 51, no. 2, pp. 456–479, Feb. 2007.

[12] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: a literature review," *Information Systems*, 2010.

[13] P. Schobbens, P. Heymans, and J. Trigaux, "Feature diagrams: A survey and a formal semantics," in *Requirements Engineering, 14th IEEE International Conference*, 2006, pp. 139–148.

[14] M. Simos, D. Creps, C. Klinger, L. Levine, and D. Allemang, "Organization domain modeling (ODM) guidebook, Version 2.0," Lockheed Martin Tactical Defense Systems, Tech. Rep. ST ARS-VC-A025/001/00, 1996.

[15] S. Mac Lane, *Categories for the Working Mathematician (Graduate Texts in Mathematics)*, 2nd ed. Springer, 1998.

[16] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering*. Springer-Verlag, 2005.

[17] P. Höfner, R. Khedri, and B. Möller, "An algebra of product families," *Software and Systems Modeling*, 2009.

[18] N. Boucké, D. Weyns, R. Hilliard, T. Holvoet, and A. Helleboogh, "Characterizing relations between architectural views," in *ECSA '08*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 66–81.

[19] A. Solomon, "Pushout: A mathematical model of architectural merger," in *Ershov Memorial Conference*, ser. Lecture Notes in Computer Science, I. Virbitskaite and A. Voronkov, Eds., vol. 4378. Springer-Verlag, 2006, pp. 389–399.

[20] Z. Diskin, "Algebraic models for bidirectional model synchronization," in *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 21–36.

[21] A. Rutle, A. Rossini, Y. Lamo, and U. Wolter, "A category-theoretical approach to the formalisation of version control in mde," in *FASE*, ser. Lecture Notes in Computer Science, M. Chechik and M. Wirsing, Eds., vol. 5503. Springer, 2009, pp. 64–78.

[22] M. Acher, P. Collet, P. Lahire, and R. France, "Composing feature models." in *SLE*, ser. Lecture Notes in Computer Science, M. van den Brand, D. Gasevic, and J. Gray, Eds., vol. 5969. Springer-Verlag, 2009, pp. 62–81.

[23] S. Segura, D. Benavides, A. Ruiz-Cortés, and P. Trinidad, "Automated merging of feature models using graph transformations," *Post-proceedings of the Second Summer School on Generative and Transformational Techniques in Software Engineering*, vol. 5235, pp. 489–505, 2008.

[24] H. Bowman, M. Steen, E. A. Boiten, and J. Derrick, "A formal framework for viewpoint consistency," *Formal Methods in System Design*, vol. 21, no. 2, pp. 111–166, 2002.

[25] J. Derrick and H. Wehrheim, "Model transformations across views," *Science of Computer Programming*, vol. 75, no. 3, pp. 192–210, 2010.

[26] "Highly Adaptable and Trustworthy Software using Formal Methods," Mar. 2009, http://www.hats-project.eu.

APPENDIX

Proposition 35. *Given a partial function $f : F \rightharpoonup G$. $\overline{f} : \overline{F} \to \overline{G}$ is total.*

Proof: Immediate from definition. ∎

Proof of functoriality of $\overline{\cdot}$ and $\widehat{\cdot}$ (§III-B). Let $p \in \overline{F}$, $u : F \rightharpoonup G$, and $v : G \rightharpoonup H$. The functor $\overline{\cdot}$ preserves identity morphisms and composition, as we show below. The proof that $\widehat{\cdot}$ is also a functor follows the same reasoning.

$$\begin{aligned}
&\overline{id_F}(p)\\
&= \{id_F(a) \mid a \in p \cap \mathrm{def}(id_F)\}\\
&= \{a \mid a \in p\}\\
&= id_{\overline{F}}(p)
\end{aligned}$$

$$\begin{aligned}
&\overline{u \circ v}(p)\\
&= \{u(v(a)) \mid a \in p \cap \mathrm{def}(u \circ v)\}\\
&= \{u(v(a)) \mid a \in p, a \in \mathrm{def}(v), v(a) \in \mathrm{def}(u)\}\\
&= \overline{u}(\{v(a) \mid a \in p, a \in \mathrm{def}(v)\})\\
&= \overline{u}(\overline{v}(p))\\
&= (\overline{u} \circ \overline{v})(p)
\end{aligned}$$

∎

Proof of Theorem 13. We split the proof into two parts. We prove in Lemma 36 that the diagram for products is a pullback, and we prove in Lemma 37 that the diagram for product lines is a weak pullback. These two lemmas are presented below. ∎

Lemma 36. *For a view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$, the following diagram is a pullback:*

$$\begin{array}{ccc}
\overline{F_r + G_r} & \xrightarrow{\overline{w_r}} & \overline{F}\\
{\scriptstyle \overline{v_r}}\downarrow & \text{p.b.} & \downarrow{\scriptstyle \overline{v_a}}\\
\overline{G} & \xrightarrow[\overline{w_a}]{} & \overline{F_a + G_a}
\end{array}$$

Proof: By definition, the canonical pullback in sets is:

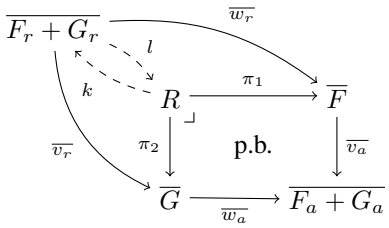
$$R = \{(p, q) \in \overline{F} \times \overline{G} \mid \overline{v_a}(p) = \overline{w_a}(q)\}.$$

To prove that the diagram is a pullback we show that $\overline{F_r + G_r}$ is isomorphic to R, given by the following functions, where $\oplus$ is defined in Definition 19:

$$\begin{array}{rcl@{\qquad}rcl}
k & : & R \to \overline{F_r + G_r} & l & : & \overline{F_r + G_r} \to R\\
(p, q) & \mapsto & p \oplus q & p & \mapsto & (\overline{w_r}(p), \overline{v_r}(p)).
\end{array}$$

Clearly $k(R) \subseteq \overline{F_r + G_r}$, and the reconciliation is defined for all pair in R. We show that the codomain of l is R based on the commutativity at the level of features. Observe that $v_a \circ w_r = w_a \circ v_r$, because when $a \in F$, $v_a(w_r(a)) = v_a(a) = v_r(a) = w_a(v_r(a))$, and similarly when $a \in G$. Therefore, because $\widehat{\cdot}$ is a functor, the outer arrows of the diagram below commute. We define $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$. Our definition of l yields trivially that $\pi_1 \circ l = \overline{w_r}$ and $\pi_2 \circ l = \overline{v_r}$.

We also conclude that, because R is a pullback, our definition of l is unique.



We claim that $k(R)$ is isomorphic to R, where the l is the inverse of k.

- $l \circ k = id_R$. Let $(p, q) \in R$, that is, $p \in \overline{F}$, $q \in \overline{q}$, and $\overline{v_a}(p) = \overline{w_a}(q)$. Then

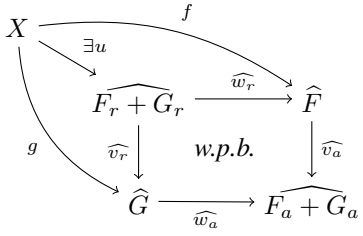
$$\begin{aligned}
l(k(p, q)) &= l(p \oplus q)\\
&= (\overline{w_r}(p \oplus q), \overline{v_r}(p \oplus q))\\
&= (p, q)
\end{aligned}$$

  where the last step follows from Lemma 17, proven below, and because $\overline{v_a}(p) = \overline{w_a}(q)$.
- $k \circ l = id_{\overline{F_r + G_r}}$. Let $p \in \overline{F_r + G_r}$ (observe that $\overline{v_a}(\overline{w_r}(p)) = \overline{w_a}(\overline{v_r}(p))$, that is, $\overline{w_r}(p) \frown \overline{v_r}(p)$). Then

$$\begin{aligned}
& k(l(p))\\
=\ & k(\overline{w_r}(p), \overline{v_r}(p))\\
=\ & \overline{w_r}(p) \oplus \overline{v_r}(p)\\
=\ & (\overline{w_r}(p) \cap Fs) \cup (\overline{v_r}(p) \cap Gs)\\
=\ & (\{w_r(a) \mid a \in p, a \in \mathrm{def}(w_r)\} \cap Fs) \ \cup\\
& (\{v_r(a) \mid a \in p, a \in \mathrm{def}(v_r)\} \cap Gs)\\
=\ & (\{w_r(a) \mid a \in p, a \in \mathrm{def}(w_r), w_r(a) \in Fs\}) \ \cup\\
& (\{v_r(a) \mid a \in p, a \in \mathrm{def}(v_r), v_r(a) \in Gs\})\\
=\ & (\{a \mid a \in p, a \in \mathrm{def}(w_r), a \in Fs\}) \ \cup\\
& (\{a \mid a \in p, a \in \mathrm{def}(v_r), a \in Gs\})\\
=\ & (\{a \mid a \in p, a \in Fs\}) \cup (\{a \mid a \in p, a \in Gs\})\\
=\ & (p \cap Fs) \cup (p \cap Gs)\\
=\ & p
\end{aligned}$$

∎

Lemma 37. *For a view partition $\sigma = (F_r, F_a, G_r, G_a, v, w)$, the following diagram is a weak pullback:*



Proof: Let u be defined as follows, where $\oplus$ is defined in Definition 19.

$$\begin{array}{rcl} u & : & X \to \widehat{F_r + G_r} \\ u(P) & = & f(P) \oplus g(P) \end{array}$$

We show that, for every $f : X \to \widehat{F}$ and $g : X \to \widehat{G}$ that makes the outer diagram commute ($\widehat{v_a} \circ f = \widehat{w_a} \circ g$), the function u exists, i.e., makes the rest of the diagram commute. The proof follows directly from Lemma 20.

$$\begin{array}{lll} & (\widehat{w_r} \circ (f \oplus g))(P) & (\widehat{v_r} \circ (f \oplus g))(P) \\ = & \widehat{w_r}(f(P) \oplus g(P)) & = \widehat{v_r}(f(P) \oplus g(P)) \\ = & f(P) & = g(P) \end{array}$$ ■

Proof of Lemma 15. We show that, when $a \in \mathrm{def}(v)$ or $b \in \mathrm{def}(w)$, then $v_a(a) = w_a(b)$ implies $a \in F_r \Leftrightarrow b \notin G_r$. It can be easily verified that the codomain of $v_a(a)$ and $w_a(b)$ only match when $a \in F_r$ and $b \notin G_r$, or when $a \notin F_r$ and $b \in G_r$. For example, if $a \in F_r$ and $b \in G_r$, then $v_a(a) = v(a) \in G_a$ and $w_a(b) \in F_a$, thus $v_a(a) \neq w_a(b)$. Due to partiality, even if the codomains do not match it could also happen that $v_a(a) = \bot = w_a(b)$, but this is invalidated by our initial assumption. ■

Proof of Lemma 17.

$$\begin{array}{rcl} \overline{w_r}(p \oplus q) & = & \overline{w_r}((p \upharpoonright F_r) \cup (q \upharpoonright G_r)) \\ & = & (p \upharpoonright F_r) \cup \overline{w}(q \upharpoonright G_r) \\ & = & (p \upharpoonright F_r) \cup (\overline{w_a}(q) \upharpoonright F_a) \\ & = & (p \upharpoonright F_r) \cup (\overline{v_a}(p) \upharpoonright F_a) \\ & = & (p \upharpoonright F_r) \cup \overline{v_a}(p \upharpoonright F_a) \\ & = & (p \upharpoonright F_r) \cup (p \upharpoonright F_a) \\ & = & p \end{array}$$

Similarly, for $\overline{v_r}(p \oplus q) = q$. ■

Proof of Lemma 20. We show that $\widehat{w_r}(P \oplus Q) = P$, and omit the analog proof for $\widehat{v_r}(P \oplus Q) = Q$.

$$\begin{array}{rcl} & \widehat{w_r}(P \oplus Q) & \\ = & \widehat{w_r}(\{(p \upharpoonright F_r) \cup (q \upharpoonright G_r) \mid p \in P, q \in Q, p \frown q\}) & \\ = & \{\overline{w_r}(p) \mid & \\ & \quad p \in \{(p \upharpoonright F_r) \cup (q \upharpoonright G_r) \mid p \in P, q \in Q, p \frown q\}\} & \\ = & \{\overline{w_r}((p \upharpoonright F_r) \cup (q \upharpoonright G_r)) \mid p \in P, q \in Q, p \frown q\} & \\ = & \{p \mid p \in P, q \in Q, p \frown q\} & \\ = & P & \end{array}$$

where the second last step follows from Lemma 17, and the last step follows from the fact that, when $P \frown Q$, $\forall p \in P \cdot \exists q \in Q \cdot p \frown q$, as we show below.

$$\begin{array}{rcl} P \frown Q & \Leftrightarrow & \widehat{v_a}(P) = \widehat{w_a}(Q) \\ & \Leftrightarrow & \{\overline{v_a}(p) \mid p \in P\} = \{\overline{w_a}(q) \mid q \in Q\} \\ & \Leftrightarrow & \forall p \in P \cdot \exists q \in Q \cdot \overline{v_a}(p) = \overline{w_a}(q) \wedge \\ & & \quad \forall q \in Q \cdot \exists p \in P \cdot \overline{v_a}(p) = \overline{w_a}(q) \end{array}$$ ■

Proof of Lemma 22.

1) $a \in F_r \Rightarrow v_a(w_r(a)) = v_a(a) = v_r(a) = w_a(v_r(a))$
   $a \in G_r \Rightarrow w_a(v_r(a)) = w_a(a) = w_r(a) = v_a(w_r(a))$
2,3) Follow from (1) and because $\overline{\cdot}$ and $\widehat{\cdot}$ are functors.
4) $a \in F_r \Rightarrow w_r(a) \oplus v_r(a) = a \oplus v(a) = a$
   $a \in G_r \Rightarrow w_r(a) \oplus v_r(a) = w(a) \oplus a = a$
   The last part follows by the definition of $\oplus$.
5) $$\begin{array}{rcl} & \overline{w_r}(p) \oplus \overline{v_r}(p) & \\ = & \{w_r(a) \mid a \in p, a \in \mathrm{def}(w_r), w_r(a) \in F_r\} & \\ & \cup \{v_r(a) \mid a \in p, a \in \mathrm{def}(v_r), v_r(a) \in G_r\} & \\ = & \{a \mid a \in p, a \in F_r\} \cup \{a \mid a \in p, a \in G_r\} & \\ = & (p \cap F_r) \cup (p \cap G_r) & \\ = & p & \end{array}$$
6) $$\begin{array}{rcl} & \widehat{w_r}(P) \oplus \widehat{v_r}(P) & \\ = & \{p \oplus q \mid p \in \widehat{w_r}(P), q \in \widehat{v_r}(P), p \frown q\} & \\ = & \{\overline{w_r}(p) \oplus \overline{v_r}(q) \mid p \in P, q \in P, \overline{w_r}(p) \frown \overline{v_r}(q)\} & \\ \supseteq & \{\overline{w_r}(p) \oplus \overline{v_r}(p) \mid p \in P, \overline{w_r}(p) \frown \overline{v_r}(p)\} & \\ = & \{p \mid p \in P\} & \\ = & P & \end{array}$$ ■

Proof of Lemma 23. $\widehat{c}$ is a closure operator.

1) $P \subseteq \widehat{c}(P)$ by the Lemma 22.
2) Let $P \subseteq Q$. Then:
$$\begin{array}{rcl} & \widehat{c}(Q) & \\ = & \widehat{w_r}(Q) \oplus \widehat{v_r}(Q) & \\ = & \{p \oplus q \mid p \in \widehat{w_r}(Q), q \in \widehat{v_r}(Q), p \frown q\} & \\ = & \{\overline{w_r}(p) \oplus \overline{v_r}(q) \mid p \in Q, q \in Q, \overline{w_r}(p) \frown \overline{v_r}(q)\} & \\ \supseteq & \{\overline{w_r}(p) \oplus \overline{v_r}(q) \mid p \in P, q \in P, \overline{w_r}(p) \frown \overline{v_r}(q)\} & \\ = & \widehat{c}(P) & \end{array}$$
3) Trivially, $\widehat{c}(\widehat{c}(P)) \supseteq \widehat{c}(P)$. For the other direction we start by using the same reasoning as before:
$$\begin{array}{rcl} & \widehat{c}(\widehat{c}(P)) & \\ = & \{\overline{w_r}(p) \oplus \overline{v_r}(q) \mid p, q \in \widehat{c}(P), \overline{w_r}(p) \frown \overline{v_r}(q)\} & \end{array}$$
We now show that $\overline{w_r}(p) \oplus \overline{v_r}(q) \in \widehat{c}(P)$, knowing that (i) $p, q \in \widehat{c}(P)$ and (ii) $\overline{w_r}(p) \frown \overline{v_r}(q)$.
The condition (i) implies that $\exists p_1, p_2, q_1, q_2 \in P$ such that $p = \overline{w_r}(p_1) \oplus \overline{v_r}(p_2)$, $q = \overline{w_r}(q_1) \oplus \overline{v_r}(q_2)$,

$\overline{w_r}(p_1) \frown \overline{v_r}(p_2)$ and $\overline{w_r}(q_1) \frown \overline{v_r}(q_2)$. Hence:

$$\begin{aligned}
& \overline{w_r}(p) \oplus \overline{v_r}(q) \\
= \; & \overline{w_r}(\overline{w_r}(p_1) \oplus \overline{v_r}(p_2)) \oplus \overline{v_r}(\overline{w_r}(q_1) \oplus \overline{v_r}(q_2)) \\
= \; & \overline{w_r}((\overline{w_r}(p_1) \cap F_r) \cup (\overline{v_r}(p_2) \cap G_r)) \oplus \\
& \overline{v_r}((\overline{w_r}(q_1) \cap F_r) \cup (\overline{v_r}(q_2) \cap G_r)) \\
= \; & ((\overline{w_r}(p_1) \cap F_r) \cup \overline{w}(\overline{v_r}(p_2) \cap G_r)) \cap F_r \cup \\
& (\overline{v}(\overline{w_r}(q_1) \cap F_r) \cup (\overline{v_r}(q_2) \cap G_r)) \cap G_r \\
= \; & (\overline{w_r}(p_1) \cap F_r) \cup (\overline{v_r}(q_2) \cap G_r) \\
= \; & \overline{w_r}(p_1) \oplus \overline{v_r}(q_2)
\end{aligned}$$

We just need to show that $\overline{w_r}(p_1) \frown \overline{v_r}(q_2)$ to prove that $\overline{w_r}(p) \oplus \overline{v_r}(q) \in \widehat{c}(P)$. This last step is shown from condition (ii) and by the Lemma 20:

$$\begin{aligned}
& \overline{w_r}(p) \frown \overline{v_r}(q) \\
\Leftrightarrow \; & \overline{v_a}(\overline{w_r}(p)) = \overline{w_a}(\overline{v_r}(q)) \\
\Leftrightarrow \; & \overline{v_a}(\overline{w_r}(\overline{w_r}(p_1) \oplus \overline{v_r}(p_2))) = \\
& \overline{w_a}(\overline{v_r}(\overline{w_r}(q_1) \oplus \overline{v_r}(q_2))) \\
\Leftrightarrow \; & \overline{v_a}(\overline{w_r}(p_1)) = \overline{w_a}(\overline{v_r}(q_2)) \\
\Leftrightarrow \; & \overline{w_r}(p_1) \frown \overline{v_r}(q_2)
\end{aligned}$$

■

Proof of Lemma 25. We show that $\widehat{c}(P)$ is the maximum (wrt set inclusion) of $[P]$, the equivalence class containing P. That is, if $P \sim Q$ then (1) $\widehat{c}(P) \sim P$, (2) $\widehat{c}(P) \sim Q$, and (3) $\widehat{c}(P) = \widehat{c}(Q)$. Condition (1) is justified by Lemma 20:

$$\widehat{w_r}(\widehat{c}(P)) = \widehat{w_r}(\widehat{w_r}(P) \oplus \widehat{v_r}(P)) = \widehat{w_r}(P)$$
$$\widehat{v_r}(\widehat{c}(P)) = \widehat{v_r}(\widehat{w_r}(P) \oplus \widehat{v_r}(P)) = \widehat{v_r}(P).$$

Condition (2) is a consequence of conditions (1) and (3), and the proof of condition (3) follows the same reasoning as the proof of condition (1), and uses the fact that $P \sim Q$:

$$\widehat{w_r}(\widehat{c}(P)) = \widehat{w_r}(P) = \widehat{w_r}(Q) = \widehat{w_r}(\widehat{c}(Q))$$
$$\widehat{v_r}(\widehat{c}(P)) = \widehat{v_r}(P) = \widehat{v_r}(Q) = \widehat{v_r}(\widehat{c}(Q)).$$

■

Lemma 38. *Let $f : E \rightharpoonup F$ and $g : E \rightharpoonup G$ be view functions such that $f \frown g$, and let $R = g \circ f^{-1}$. If $a_1 \, R \, b_1$ and $a_2 \, R \, b_2$, where $a_1 \neq a_2$ and $b_1 \neq b_2$, then $(a_1, b_2) \notin R$.*

Proof: This lemma can be proved by observing that $f^{-1}(a_1) \# g^{-1}(b_2)$ can never hold. If a_1, a_2, b_1, and b_2 are as defined in the lemma, then $\exists e_i \in f^{-1}(a_i) \cap g^{-1}(b_i)$ for $i \in \{1, 2\}$. When $a_1 \, R \, b_2$ the condition $f^{-1}(a_1) \# g^{-1}(b_2)$ no longer holds. If $a_1 \, R \, b_2$ then $\exists e_3 \cdot f(e_3) = a_1 \wedge g(e_3) = b_2$. Therefore $\{e_3, e_1\} \subseteq f^{-1}(a_1)$ and $\{e_3, e_2\} \subseteq g^{-1}(b_2)$, but $e_1 \notin g^{-1}(b_2)$ (because $g(e_1) = b_1$) and $e_2 \notin f^{-1}(a_1)$ (because $f(e_2) = a_2$). We conclude that $f^{-1}(a_1) \# g^{-1}(b_2)$ cannot hold. ■

Proof of Theorem 27. Let $P \in \widehat{E}$, and $R = g \circ f^{-1} \subseteq F \times G$. R is partitioned into two parts: pairs with more than one image and its complements. For that we define the partitions $F = F_r \uplus F_a$ and $G = G_r \uplus G_a$ as follows.

$$F_a = \{x \in F \mid |R(x)| > 1\} \tag{22}$$
$$F_r = F \setminus F_a \tag{23}$$
$$G_a = R(F_r) \tag{24}$$
$$G_r = G \setminus G_a \tag{25}$$

We now define $v : F_r \rightharpoonup G$ and $w : G_r \rightharpoonup F$ such that $v = R$ and $w = R^{-1}$ restricted to the corresponding domains. To show that $\sigma = (F_r, F_a, G_r, G_a, v, w)$ is a view partition, we still need to verify that v and w are (partial) functions, and that the codomains of v and w are G_a and F_a, respectively.

1) w *is a function* – If $w(y) = x$ and $w(y) = x'$, then $x \, R \, y$ and $x' \, R \, y$. By definition of F_a, $\exists y' \in G \cdot x \, R \, y'$, which contradicts Lemma 38. Hence $x = x'$.
2) v *is a function* – If $v(x) = y$, then by the definition of F_r we know that $|R(x)| \leq 1$, hence y is unique.
3) $\mathrm{cod}(v) = G_a$ – By definition of G_a.
4) $\mathrm{cod}(w) = F_a$ – We show that if $x \in F_a$ and $x \, R \, y$, then $y \in G_r$. If $y \in G_a$ ($\equiv y \notin G_r$), then (1) $\exists x' \in F_r \cdot x' \, R \, y$ because v is onto , and (2) $\exists y' \in G \cdot x \, R \, y'$ by the definition of F_a. But (1) and (2) contradict Lemma 38, hence $y \in G_r$.

By the definition and properties of v and w we also conclude that $v + w^{-1} = R$. As a consequence, we show that

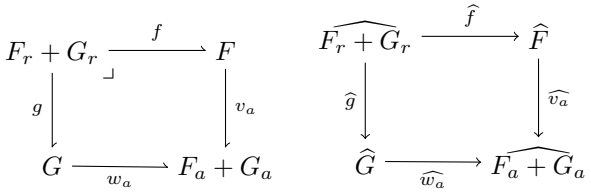
$$\forall a \in E \cdot f(a) \frown_\sigma g(a). \tag{26}$$

Recall that $f(a) \frown g(a) \stackrel{\text{def}}{=} v_a(f(a)) = w_a(g(a))$. Observe now that, when $f(a) \in F_r$ we have $v_a(f(a)) = v(f(a)) = (g \circ f^{-1})(f(a)) = g(a) = w_a(g(a))$, and when $f(a) \in F_a$ we have $v_a(g(a)) = f(a) = (f \circ g^{-1})(g(a)) = w(g(a)) = w_a(g(a))$.

Finally, we use Equation (26) to show the final result:

$$\forall P \in \widehat{E} \cdot \widehat{f}(P) \frown_\sigma \widehat{g}(P). \tag{27}$$

It is enough to verify that the left diagram below always commutes hence, because $\widehat{\cdot}$ is a functor, the right diagram below also commutes.



■

Proof of Lemma 30.

$$
\begin{aligned}
& P \oplus_\sigma Q_{f\emptyset} \\
= \; & \{p \oplus q \mid p \in P, q \in Q_{f\emptyset}, p \frown q\} \\
= \; & \{p \oplus q \mid p \in P, q \in Q_f, q \neq \emptyset, p \frown q\} \cup \\
& \{p \oplus \emptyset \mid p \in P, p \frown \emptyset\} \\
= \; & \{p \oplus q \mid p \in P, q \in Q_f, q \neq \emptyset, \overline{w_r}(p) = \overline{v_r}(q)\} \cup \\
& \{p \oplus \emptyset \mid p \in P, \overline{w_r}(p) = \emptyset\} \\
\overset{*}{=} \; & \{(p \cap F_r) \cup (q \cap G_r) \mid p \in P, q \in Q_f, q \neq \emptyset, f \in p\} \cup \\
& \{p \oplus \emptyset \mid p \in P, f \notin p\} \\
= \; & \{(p \cap (F \backslash \{f\})) \cup (q \cap \{g \mid g \in q', q' \in Q_f\}) \\
& \mid p \in P, q \in Q_f, q \neq \emptyset, \; f \in p\} \cup \\
& \{p \mid p \in P, f \notin p\} \\
= \; & \{(p \backslash \{f\}) \cup q \mid p \in P, q \in Q_f, q \neq \emptyset, \; f \in p\} \cup \\
& \{p \mid p \in P, f \notin p\}
\end{aligned}
$$

In the forth step, marked with '*', we interpret compatibility in our scenario. The condition $\overline{w_r}(p) = \overline{v_r}(q)$ holds exactly when $f \in p$ and $q \neq \emptyset$, or when $f \notin p$ and $q \neq \emptyset$, because f is the only abstraction. ■

Proof of Lemma 33.

$$
\begin{aligned}
& P \oplus_\sigma Q_k \\
= & P \oplus_\sigma (\{c \cup g \mid (c, g) \in k\} \cup \widehat{C}) \\
= & \{p \oplus q \mid p \in P, q \in \{c \cup g \mid (c, g) \in k\}, p \frown q\} \\
\cup & \{p \oplus c \mid p \in P, c \in \overline{C}, p \frown c\}
\end{aligned}
$$

We start by reducing the first element of the union above.

$$
\begin{aligned}
& \{p \oplus q \mid p \in P, q \in \{c \cup g \mid (c, g) \in k\}, p \frown q\} \\
= & \{p \oplus (c \cup g) \mid \\
& \quad p \in P, c \in \overline{C}, g \in k(c), \overline{v_a}(p) = \overline{w_a}(c \cup g)\} \\
= & \{(p \cap (F \backslash (\{f\} \cup C))) \cup ((c \cup g) \cap G) \mid \\
& \quad p \in P, c \in \overline{C}, g \in k(c), \overline{v_a}(p) = \overline{w_a}(c \cup g)\} \\
= & \{(p \backslash (\{f\} \cup C)) \cup c \cup g \mid \\
& \quad p \in P, c \in \overline{C}, g \in k(c), p \cap (\{f\} \cup C) = \{f\} \cup c, g \neq \emptyset\} \\
\cup & \{(p \backslash (\{f\} \cup C)) \cup c \cup g \mid \\
& \quad p \in P, c \in \overline{C}, g \in k(c), p \cap (\{f\} \cup C) = c, g = \emptyset\} \\
= & \{(p \backslash (\{f\} \cup C)) \cup c \cup g \mid \\
& \quad p \in P, c \in \overline{C}, g \in k(c), p \cap C = c, f \in p \Leftrightarrow g \neq \emptyset\} \\
= & \{p \backslash \{f\} \cup g \mid p \in P, g \in k(p \cap C), f \in p \Leftrightarrow g \neq \emptyset\}
\end{aligned}
$$

We now reduce the second element of the union.

$$
\begin{aligned}
& \{p \oplus c \mid p \in P, c \in \overline{C}, p \frown c\} \\
= & \{p \oplus c \mid p \in P, c \in \overline{C}, \overline{v_a}(p) = \overline{w_a}(c)\} \\
= & \{p \oplus c \mid p \in P, c \in \overline{C}, p \cap (\{f\} \cup C) = c\} \\
= & \{p \cap (\{f\} \cup C) \cup (c \cap G) \mid p \in P, f \notin p, p \cap C = c\} \\
= & \{p \cap (\{f\} \cup C) \cup (p \cap C) \mid p \in P, f \notin p\} \\
= & \{p \mid p \in P, f \notin p\}
\end{aligned}
$$

■