Abstract delta modeling : software product lines and beyond

Helvensteijn, M.

A

DMW Operational Semantics

Formalization of the Workflow and Proofs of its Properties

A.1 The Subfeature Relation

From here on, assume a deltoid $(\mathcal{P}, \mathcal{D}, \cdot, \varepsilon, [\![\text{-}]\!])$ that exhibits consistent conflict resolution (Definition 3.18) and a feature set $\mathcal{F}$.

Based on the subfeature relation defined in Section 7.2 (page 150), we extend the specification of a product line as follows:

▶ **A.1. Definition (Structured Product Line Specification):** A *structured product line specification* is a triple $sPLS = (\Phi, \twoheadrightarrow, \mathrm{V})$ where (Φ, V) is a product line specification (Definition 4.19) and $\twoheadrightarrow \subseteq \mathcal{F} \times \mathcal{F}$ is a direct subfeature relation. For all such specifications we require the following two properties to hold for all features $f, g \in \mathcal{F}$, all feature configurations $F \in \Phi$:

$$\textbf{a.}\quad f \twoheadrightarrow g \quad\Longrightarrow\quad (g \in F \quad\Longrightarrow\quad f \in F)$$

$$\textbf{b.}\quad f \twoheadrightarrow g \quad\Longrightarrow\quad (\mathrm{V}(\{g\}) \quad\subseteq\quad \mathrm{V}(\{f\}))$$

Namely, (a) that the selection of any feature implies the selection of its superfeatures and (b) a product's support for any feature implies support for its superfeatures. The set of all structured product line specifications is denoted $sP\mathcal{LS}$. If the deltoid or feature set is not clear from context, we attach a subscript as in $sP\mathcal{LS}_{Dt,\mathcal{F}}$. ⌟

A.2 Non-interference

The *non-interference* property described in Section 7.3 (page 151) is formally defined as follows:

▶ **A.2. Definition (Non-Interference):** A given deltoid $Dt = (\,\mathcal{P}, \mathcal{D}, \cdot, \varepsilon, [\![\text{-}]\!]\,)$ and valuation function $\mathrm{V}\colon \mathrm{Pow}(\mathcal{F}) \to \mathrm{Pow}(\mathcal{P})$ jointly exhibit the property of *non-interference* iff for all deltas $x, y, z \in \mathcal{D}$, products p and feature selections F:

$$\underbrace{z \cdot y \cdot x = z \cdot x \cdot y}_{\text{a}} \quad \Longrightarrow \quad \underbrace{[\![z \cdot x]\!](p) \subseteq \mathrm{V}(F) \;\Rightarrow\; [\![z \cdot y \cdot x]\!](p) \subseteq \mathrm{V}(F)}_{\text{b}} \qquad \lrcorner$$

So we can make use of locality if (a) for all delta-pairs x, y that commute in some context z (which may or may not be resolving a conflict between them), (b) in that same context, any property introduced to the final product by x cannot be broken by the presence or absence of y.

From this point on, we assume a structured product line specification $sPLS = (\,\Phi, \twoheadrightarrow, \mathrm{V}\,)$ which exhibits non-interfere with the earlier assumed deltoid Dt (Definition A.2).

A.3 An Operational Semantics

The job-based model introduced in Section 7.4.4 (page 153) is now described as an *operational semantics* (Section 1.7.11), in which the steps are transitions (Notation 1.49).

A.3.1 Job Status

A job can be in one of three stages. We define a type of mapping to keep track of the status of all jobs:

▶ **A.3. Definition (Job Status Map):** Given some set of already developed deltas $D \subseteq \mathcal{D}$, a *job status map* $\mathrm{J}\colon (\,\mathrm{Pow}(\mathcal{F}) \cup \mathrm{Pow}(D)\,) \rightharpoonup (\,\{\mathrm{av}, \mathrm{ip}\} \cup D\,)$ is a finite partial function (Definition 1.17) mapping each job $j \in \mathrm{Pow}(\mathcal{F}) \cup \mathrm{Pow}(D)$ to its current status. Either:

- it is not recognized as a viable job (yet): $\mathrm{J}(j) = \bot$,
- it is available: $\mathrm{J}(j) = \mathrm{av}$,
- it is in progress: $\mathrm{J}(j) = \mathrm{ip}$, or
- it is finished, and has resulted in delta $d \in D$: $\mathrm{J}(j) = d$ $\lrcorner$

A.3.2 Configurations

Each state of the workflow is represented by a configuration as follows:

▶ **A.4. Definition (Workflow States):** A *workflow state* is a configuration:

$$ws \quad = \quad \langle\, adm,\ \mathrm{J} \,\rangle$$

where $adm = (D, \prec, \gamma)$ is the annotated delta model in progress and the function $\mathrm{J}\colon (\,\mathrm{Pow}(\mathcal{F}) \cup \mathrm{Pow}(D)\,) \rightharpoonup (\,\{\mathrm{av}, \mathrm{ip}\} \cup D\,)$ is a job status map (Definition A.3) used for bookkeeping. The whole configuration space is denoted WS. ⌟

The initial state of the workflow is simple. The annotated delta model is still empty and no jobs have been formulated yet:

▶ **A.5. Definition (Initial State):** The *initial state* of the workflow is defined as follows:

$$ws_0 \quad \stackrel{\text{def}}{=} \quad \langle\, adm_0,\, \mathrm{J}_0 \,\rangle \quad = \quad \langle\, \varnothing,\, \varnothing,\, \varnothing,\, \varnothing \,\rangle$$

with an empty annotated delta model $(\,\varnothing, \varnothing, \varnothing\,)$ —parentheses are omitted—, and an empty set of initial jobs $\varnothing$. ⌟

Steps 1 to 6 of the workflow description in Section 7.4.3 are each represented as inference rules (Notation 1.15), which define valid state transitions $\longrightarrow \subseteq WS \times WS$. The whole development process can then be represented as a chain of n transitions:

$$\langle\, adm_0,\, \mathrm{J}_0 \,\rangle \;\longrightarrow\; \ldots \;\longrightarrow\; \langle\, adm_n,\, \mathrm{J}_n \,\rangle$$

The workflow is finished after $\frac{1}{3}n$ jobs $j \in \mathrm{Pow}(\mathcal{F}) \cup \mathrm{Pow}(D)$, each of which goes through 3 transitions:

1. Identifying the job (steps 1, 3 and 5), which inserts it into the job status map J with status 'available' (av),
2. starting a job, which gives it the status 'in progress' (ip), and
3. finishing a job, which results in a new delta (steps 2, 4 and 6).

▶ **A.6. Notation:** We identify each specific transition t by its job and transition number, e.g., $t = \{f\}_3$ is the transition that takes place upon finishing the implementation of feature f. We sometimes annotate the transition arrow with this information: $\xrightarrow{\{f\}_3}$. ⌟

By splitting each job up this way, we explicitly allow interleaving its transitions with the transitions of other jobs, while keeping important updates as atomic operations. This makes it clear which jobs can be performed concurrently.

A.3.3 Inference Rules

The job order described in Section 7.4.4 (page 153) is formally encoded in the inference rules of the operational semantics. We now define these inference rules. Together they define the transition relation $\longrightarrow$. We need to make sure that the workflow eventually terminates at the configuration representing a correct product line $(\Phi, 0, adm_n)$.

Identifying New Jobs

First, define the inference rule for introducing a new feature implementation job. But to do so, we first have to determine exactly which feature combinations will (eventually) get a specific delta to implement them:

▶ **A.7. Definition (Viable Feature Combination):** A *viable feature combination* is a feature set $F \subseteq \mathcal{F}$ such that:

$$\mathsf{vf}(F) \quad \overset{\text{def}}{\Longleftrightarrow} \quad \underbrace{\exists F' \in \Phi\colon\ F \subseteq F'}_{\text{a}} \quad \wedge \quad \underbrace{\mathrm{V}(F) \neq \bigcap_{E \nrightarrow F} \mathrm{V}(E)}_{\text{b}} \qquad \lrcorner$$

A feature combination is viable as a feature implementation job iff (a) all of its features can be selected together and, (b) when selected, present requirements that are not already presented by some combination of weaker subsets. If not for these conditions, we would have to 'implement' many deltas that do nothing.

Next, we'll define a much used shorthand notation:

A.8. Notation: The *viable feature combination order* $\nrightarrow_{\mathrm{v}} \subseteq \mathcal{F} \times \mathcal{F}$ is the feature combination order (Definition 7.3) restricted to viable feature combinations: $\nrightarrow_{\mathrm{v}} \overset{\text{def}}{=} \nrightarrow \cap\, \mathsf{vf}^2$ $\lrcorner$

Now for the inference rule itself:

▶ **A.9. Definition (Workflow Inference Rule):** NEW-FEATURE-JOB

$$\frac{\overbrace{\mathsf{vf}(F)}^{\text{a}} \qquad \overbrace{\forall E \nrightarrow_{\mathrm{v}} F\colon\ \mathrm{J}(E) \in D}^{\text{b}}}{\langle D,\ \prec,\ \gamma,\ \mathrm{J}\,[F \mapsto \bot]\rangle \xrightarrow{F_1} \langle D,\ \prec,\ \gamma,\ \mathrm{J}\,[F \mapsto \mathrm{av}]\rangle} \qquad \lrcorner$$

To recognize F as a valid job from the current state, it needs (a) to be viable, and (b) any weaker viable feature combinations must be already implemented. That way, functionality is implemented in the proper order.

We need this particular ordering because deltas implementing stronger feature combinations should have knowledge and control over deltas implementing weaker ones. After writing deltas to implement the features *SH*, *EC* and *KM* from Section 4.5.1, for example, we would like the resultant annotated delta model to look like the one in Figure 4.3. We need the $\{\, SH, EC, KM \,\}$ job to be available only after the 'smaller' jobs are finished. This is assuming the general case that each combination needs special consideration. In simpler cases, parametric deltas may be used (which were the topic of Section 4.5), but that is out of scope for the workflow description of this chapter.

Now to define an inference rule for identifying conflict resolution jobs. Formally, a conflict occurs between two deltas, as discussed in Section 3.3. However, when a there is a set of deltas with many (related) conflicts, we will also want to introduce conflict-resolving deltas for some larger sets, if they have a non-empty joint application condition, in order to cover all combinations. We again define a predicate to help us:

▶ **A.10. Definition (Viable Conflict Set):** Given an annotated delta model $adm = (D, \prec, \gamma)$, a set $C \subseteq D$ is a *viable conflict set* iff:

$$\mathsf{vc}(C) \quad \overset{\text{def}}{\iff} \quad \underbrace{\gamma_\cap(C) \neq \varnothing}_{\text{a}} \quad \wedge \quad \underbrace{\begin{array}{l}\forall x \in C\colon\ \exists y \in C\colon\ x \not\lightning y \ \wedge \\ \nexists z \in D\colon\ \gamma_\cap(\{x,y\}) \subseteq \gamma(z) \ \wedge \ (x,y) \lhd z\end{array}}_{\text{b}}$$

If the delta model is not clear from context, we attach a subscript as in vc_{adm} or $\mathsf{vc}_{\prec}$. ⌟

A delta set is a viable as a conflict resolution job iff (a) all of its deltas can be selected together (Definition 4.12), and (b) all are in unresolved conflict with at least one other delta in the set. Now, the inference rule itself:

▶ **A.11. Definition (Workflow Inference Rule):** NEW-CONFLICT-JOB

$$\frac{\overbrace{\mathsf{vc}(C)}^{\text{a}} \qquad \mathsf{vc}(C') \implies \overbrace{\gamma_\cap(C') \not\supset \gamma_\cap(C)}^{\text{b}} \ \wedge \ \overbrace{\gamma_\cap(C') = \gamma_\cap(C) \implies C' \subseteq C}^{\text{c}}}{\langle D, \prec, \gamma, \mathrm{J}[C \mapsto \bot]\rangle \xrightarrow{C_1} \langle D, \prec, \gamma, \mathrm{J}[C \mapsto \mathrm{av}]\rangle}$$

⌟

To recognize C as a valid new job, it (a) needs to be viable, (b) may not have a stronger joint application condition than any other viable set and (c) must be the largest of all viable sets that share the same joint application condition. Condition (b) ensures that more generally applicable conflict resolvers are developed first. Condition (c) ensures that no duplicate work is performed, and that the workflow eventually terminates (Appendix A.4.1).

Starting a Job

Starting a job is the simplest inference rule, but having it is important to make the possibility of concurrent development explicit, i.e., that more than one job can be in progress at the same time.

▶ **A.12. Definition (Workflow Inference Rule):** STARTING-JOB

$$\frac{}{\langle adm, \mathrm{J}[j \mapsto \mathrm{av}]\rangle \xrightarrow{j_2} \langle adm, \mathrm{J}[j \mapsto \mathrm{ip}]\rangle}$$

⌟

When a job is started, its status is simply set from 'available' to 'in progress', to prevent a job from being started more than once.

Finishing a Job

We now present the final two inference rules, responsible for validating the correctness of a developed delta and integrating it into the annotated delta model. First, the rule for finishing a feature implementation job:

▶ **A.13. Definition (Workflow Inference Rule):** FINISHING-FEATURE-JOB

$$\frac{\begin{array}{l}\overbrace{\prec_\Delta \;=\; \{\; (\mathrm{J}(E), d\,) \;\mid\; E \twoheadrightarrow_{\mathrm{v}} F \;\}}^{\text{a}} \qquad \overbrace{(\,\forall E \twoheadrightarrow_{\mathrm{v}} F\colon\; [\![\downarrow \mathrm{J}(E)]\!](c) \;\subseteq\; \mathrm{V}(E)\,) \;\Rightarrow\; [\![\downarrow d]\!](c) \;\subseteq\; \mathrm{V}(F)}^{\text{c}}\end{array}}{\begin{array}{llll}\langle\; D \setminus \{d\}, & \prec, & \gamma, & \mathrm{J}\,[\,F \mapsto \mathrm{ip}\,]\;\rangle \xrightarrow{F_3} \\ \langle\; D \cup \{d\}, & \prec \cup \prec_\Delta, & \gamma[\,d \mapsto \{F' \in \Phi \mid F \subseteq F'\}\,], & \mathrm{J}\,[\,F \mapsto d\,]\;\rangle \\ \underbrace{\phantom{D \setminus \{d\},}}_{\text{b}} & \underbrace{}_{\text{a}} & \underbrace{\phantom{\gamma[\,d \mapsto \{F' \in \Phi \mid F \subseteq F'\}\,],}}_{\text{d}} & \underbrace{\phantom{\mathrm{J}\,[\,F \mapsto d\,]\;\rangle}}_{\text{e}}\end{array}}$$

To implement a feature or feature interaction, a new delta d is developed (a) to be applied later than the deltas that implement weaker viable feature combinations. (b) It is added to the delta set. (c) It needs to have a local delta model that satisfies the requirements of F, with the assumption that all weaker viable feature combinations similarly satisfy their own requirements. (d) It is applied whenever all features in F are selected. Finally, (e) we map the job F to the delta d that implements it. (As you can see, we may use this mapping later to (a) order subsequent feature implementation deltas.)

And last but not least, the rule for finishing a conflict resolution job:

▶ **A.14. Definition (Workflow Inference Rule):** FINISHING-CONFLICT-JOB

$$\frac{\overbrace{\forall x, y \in C\colon\;\; z \cdot y \cdot x \;=\; z \cdot x \cdot y}^{\text{a}} \qquad \overbrace{\forall x \in C\colon\;\; [\![\downarrow z]\!](c) \;\subseteq\; [\![\downarrow x]\!](c)}^{\text{b}}}{\begin{array}{llll}\langle D \setminus \{z\}, & \prec, & \gamma, & \mathrm{J}\,[\,C \mapsto \mathrm{ip}\,]\;\rangle \xrightarrow{C_3} \\ \langle D \cup \{z\}, & \prec \cup \{\,(d, z\,) \mid x \preccurlyeq d \vee y \preccurlyeq d\,\}, & \gamma\,[\,z \mapsto \gamma(C)\,], & \mathrm{J}\,[\,C \mapsto z\,]\;\rangle \\ \underbrace{\phantom{D \setminus \{z\},}}_{\text{c}} & \underbrace{\phantom{\prec \cup \{\,(d, z\,) \mid x \preccurlyeq d \vee y \preccurlyeq d\,\},}}_{\text{d}} & \underbrace{}_{\text{e}} & \underbrace{\phantom{\mathrm{J}\,[\,C \mapsto z\,]\;\rangle}}_{\text{f}}\end{array}}$$

To properly resolve the conflict between all deltas in C, a new delta z is developed that (a) allows all deltas in C to commute and (b) has a resulting local delta model that preserves the requirements that were preserved by each individual delta in C. It is (c) added to the delta set, (d) to be applied later than the conflicting deltas, (e) whenever those are applied too. Finally, (f) we map the job C to the delta z that implements it.

The Full Transition Relation

And that concludes the formulation of the abstract delta modeling workflow:

▶ **A.15. Definition (DMW Transition Relation):** The transition relation of the delta modeling workflow operational semantics is the smallest relation $\longrightarrow \subseteq WS \times WS$ characterized by Definitions A.9, A.11, A.12, A.13 and A.14.

A.4 Analysis

This section presents proofs of three main theorems about the workflow. First, Appendix A.4.1 proves termination. Then, Appendices A.4.2 and A.4.3 prove that any product line created through the workflow is unambiguous and totally correct with respect to the product line specification that was used as input.

A.4.1 Termination

First, we show that the workflow eventually terminates. This being an operational semantics, the workflow is finished when we reach a configuration that is stuck, i.e., from which there are no valid transitions left to take.

▶ **A.16. Theorem:** The workflow is guaranteed to terminate, i.e., starting from initial state ws_0 (Definition A.5), there is no infinite transition path: $ws_0 \not\longmapsto^{\infty}$ (Definition 1.50, page 28).

Proof: Two kinds of job exist. Feature implementation jobs are identified by sets of features, and generated directly from the product line specification (Figure 7.4). Since there are only a finite number of features in a feature model (Notation 4.2 and Definition 4.3), these jobs can never be a source of divergence, even if every possible combination would require a separate delta.

Conflict resolution jobs, however, are generated not from the specification, but from the implementation itself. They add a new conflict resolution delta to the set D. Conflict resolution deltas may cause new conflicts themselves, so there is a potentially infinite source of new conflicts. Figure 7.5 shows the feedback loop in question.

To prove termination we show that $\longrightarrow$ is well-founded, but for simplicity we'll only consider conflict resolution jobs:

$$\begin{aligned} \langle\, D,\ \prec,\ \gamma,\ \mathrm{J}\,\rangle \ &\xrightarrow{C_1}\ \langle D,\ \prec,\ \gamma,\ \mathrm{J}\,[\,C \mapsto \mathrm{av}\,]\,\rangle \\ &\xrightarrow{C_2}\ \langle D,\ \prec,\ \gamma,\ \mathrm{J}\,[\,C \mapsto \mathrm{ip}\,]\,\rangle \\ &\xrightarrow{C_3}\ \langle D',\ \prec',\ \gamma',\ \mathrm{J}\,[\,C \mapsto z\,]\ \rangle \end{aligned}$$

In particular, we assign a value from a well-founded set to each delta d, and show that the value assigned to z is strictly smaller than that assigned to the deltas $x \in C$ of the conflict-set it resolves. This value is the pair $(\,\gamma(d), \not\lessgtr(d)\,)$, where

$$d \not\lessgtr d' \quad \iff \quad \gamma(d) = \gamma(d') \ \wedge\ d \not\prec d' \ \wedge\ d' \not\prec d$$

is a symmetric relation between deltas that share the same application condition and can potentially be in conflict with each other. The pair reduces lexicographically from x to z if either:

- $\gamma(z) \subset \gamma(x)$, i.e., z has a stronger application condition than x, or
- $\gamma(z) = \gamma(x) \ \wedge\ \not\lessgtr(z) \subset \not\lessgtr(x)$, i.e., z has an application condition equal to x, but there are strictly fewer deltas it can potentially conflict with.

By Definition A.14, we have $\gamma(z) = \gamma_{\cap}(C)$, so the application condition of z is always equal to or stronger than that of $x \in C$ (Definition 4.12). If $\gamma(z) \subset \gamma(x)$, we are done. If $\gamma(z) = \gamma(x)$, then we can show that $\not\lessgtr(z) \subset \not\lessgtr(x)$:

- Take a delta $d \not\lessgtr z$.
  (a) We have $\gamma(d) = \gamma(z) = \gamma(x)$.

(b) Because $d \not\prec z$ and $x \prec z$ we also have $d \not\prec x$.
(c) Assume $x \prec d$. Without loss of generality we can assume d was the result of a conflict resolution job C' with $x \in C'$ (it may in fact be a feature implementation delta, but this can only happen finitely often, so we dismiss it as a source of divergence). This leads to contradiction, as by Definition A.11, C' would be largest conflict set with the same joint application condition, so we'd have $C \subseteq C'$, and by Definition A.14, d would have resolved all conflicts in C already, making it inviable as the current job. So we have $x \not\prec d$.
(d) From (a), (b) and (c) we conclude that $\not\lessgtr(z) \subseteq \not\lessgtr(x)$.

- By Definition A.11, we have $|C| > 1$, so there is clearly at least one delta $y \not\lessgtr x$ with $\neg(y \not\lessgtr z)$. So $\not\lessgtr(z) \neq \not\lessgtr(x)$, and therefore $\not\lessgtr(z) \subset \not\lessgtr(x)$.

So by this measure, $(\,\gamma(z), \not\lessgtr(z)\,)$ is strictly smaller than $(\,\gamma(x), \not\lessgtr(x)\,)$. As there is clearly a smallest value $(\varnothing, \varnothing)$, the described relation is well-founded, and there cannot be an infinite decreasing chain of conflict resolutions. □

In short, if any conflict resolving delta is in conflict itself, it requires a new conflict resolving delta with a lower 'value', and this can only happen a finite number of times. As an extreme example, there could be one conflict resolving delta z, with $d \prec z$ for all other deltas $d \in D$, giving $\not\lessgtr(z) = \varnothing$.

A.4.2 Unambiguity

The product line implementation *PLI* resulting from the workflow is supposed to be totally correct with regard to the specification *sPLS*. This is proved in Appendix A.4.3. But first, we need an intermediate result: unambiguity (Definition 4.11, page 104). We prove that every feature configuration gives rise to an unambiguous selected delta model.

▶ **A.17. Theorem:** Given a stuck configuration of the workflow $\langle\, D, \prec, \gamma, \mathrm{J}\,\rangle$, the corresponding product line implementation $PLI = (\,\Phi, c, D, \prec, \gamma\,)$ is unambiguous.

Proof: We'll prove by contradiction that *PLI* is *globally unambiguous*, which implies that it is generally unambiguous (Theorem 4.16, page 106).

Assume that *PLI* is not globally unambiguous. This means that there exists a pair of deltas $x, y \in D$ with all of the following properties:

- They are in conflict: $x \lightning y$,
- They have a non-empty joint application condition: $\gamma_\cap(\{x, y\}) \neq \varnothing$
- There is no delta $z \in D$ such that $\gamma_\cap(\{x, y\}) \subseteq \gamma(z)$ and $(x, y) \lhd z$

Then by Definition A.10, $\{x, y\}$ is a viable conflict set. Therefore, the inference rule NEW-CONFLICT-JOB (Definition A.11) is applicable to configuration $\langle\, D, \prec, \gamma, \mathrm{J}\,\rangle$, meaning it is not stuck and the workflow is not finished.

This contradiction proves the original statement: a stuck configuration yields a product line implementation that is globally unambiguous, and, therefore, unambiguous as per Definition 4.11. □

A.4.3 Total Correctness

Finally, we prove that the resulting product line implementation is totally correct w.r.t. the given structured product line specification as defined in Definition 4.20 (page 108):

▶ **A.18. Theorem:** Given a final (stuck) configuration of the workflow $\langle D, \prec, \gamma, \mathrm{J} \rangle$, the corresponding product line implementation $PLI = (\Phi, c, D, \prec, \gamma)$ is totally correct with regard to the given product line specification $sPLS = (\Phi, \not\Rightarrow, \mathrm{V})$.

Proof: Call the annotated delta model $adm = (D, \prec, \gamma)$.

Take an arbitrary feature configuration $F \in \Phi$. We name the selected delta model $dm_F = (D_F, \prec_F) = adm \restriction F$. We then name the set $VF = \{ G \subseteq F \mid \mathsf{vf}(G) \}$ of viable feature combinations that are a subset of F (Definition A.7).

By Definition A.9, each $G \in VF$ becomes a new feature job. Moreover, by Definition A.13a, the job map J is a homomorphism from $(VF, \not\Rightarrow_{\mathrm{v}} \cap VF^2)$ to $(D_F, \prec_F)$, i.e., for all feature combinations $G_1, G_2 \subseteq F$:

$$G_1 \not\Rightarrow_{\mathrm{v}} G_2 \quad \Longleftrightarrow \quad \mathrm{J}(G_1) \prec_F \mathrm{J}(G_2)$$

This is also illustrated in Figures 7.4 and 7.5.

We can prove that $\mathrm{V}(F) = \bigcup_{G \in VF} \mathrm{V}(G)$. While this is not true in the general case (as stated in Section 4.4.1), it is now true by construction. If it were not, there would need to still be a feature combination $E \subseteq F$ with $\mathsf{vf}(E)$ and $E \notin VF$. But if there was, NEW-FEATURE-JOB would still apply, and our 'final' configuration would not be stuck. But it is.

We can prove that for all $G \in VF$, we have $[\![\downarrow \mathrm{J}(G)]\!](c) \subseteq \mathrm{V}(G)$, by induction on the strength of G. Both the base and inductive case are proved rather straightforwardly by using Definition A.13c.

That being true, we have $[\![dm_F]\!](c) \subseteq \mathrm{V}(F)$ if none of the deltas outside of the local delta model $\downarrow \mathrm{J}(G)$ break the introduced functionality. There are two kinds of such deltas: deltas d with $\mathrm{J}(G) \prec_F d$ and deltas unordered with $\mathrm{J}(G)$. The former type cannot interfere: because of Definitions A.14b and A.13c, developers have to obey local constraints not to break features of delta's before them. The latter type also cannot interfere: because PLI is unambiguous (Theorem A.17), all pairs of deltas $x, y \in D_F$ are either ordered by $\prec_F$, or they commute in the context of the full derivation $d_1 \cdot y \cdot x \cdot d_2 = d_1 \cdot x \cdot y \cdot d_2 \in \mathrm{derv}(dm_F)$, and we assumed a deltoid with non-interference (Definition A.2).

This leads to our desired result:

$$\forall F \in \Phi\colon \quad [\![adm \restriction F]\!](c) \subseteq \mathrm{V}(F) \qquad \square$$