Simple Feature Engineering via Neat Default Retrenchments

R. Banach, C. Jeske
School of Computer Science, Manchester University, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk, jeskec@cs.man.ac.uk

Abstract. Feature engineering deliberately stages the incorporation of elements of functionality into a system according to perceived user and market needs. Conventional refinement based techniques for feature engineering suffer from the need to have successive features build smoothly on their predecessors, since contradicting what has already been established is anathema for any refinement technique. Real feature engineering however must at times insist on such contradictions. Retrenchment offers a more flexible approach for capturing such less well behaved development steps within a formal framework that interworks smoothly with refinement, and a generic account of 'simple' feature engineering (encompassing situations in which operations may be dealt with, one at a time) is given, using a simple language to express feature oriented descriptions (FODs) of operations, and a simple rewriting formalism to express changes in the FOD. The generic account shows that under a set of reasonable assumptions, the retrenchments belong to the class of neat, default retrenchments.
Keywords. Feature Engineering, Retrenchment, Neat and Default Retrenchment.

1 Introduction

In [Zave (2001)] a *feature* of a software system is described as 'an optional or incremental unit of functionality', and the technique of developing or evolving software by taking into account successive features is called feature engineering, and the idea is much more widely applicable in system development than just in the telecommunications engineering field. The aim of this paper is to give an account of a simple kind of feature engineering in a formal framework broadly in sympathy with model based refinement [de Roever (1998)]. This is not a new goal. For instance [Back (2002), Back and Sere (1996), Katz (1993), Francez and Forman (1990)] all propose formulations of refinement, specifically superposition refinement and closely related concepts, in which successive features are built smoothly upon the facilities offered by their predecessors. More recently, the Event-B proposal [Abrial (2010)] takes the same idea of accumulation of design detail in successive refinement stages, and reworks it in the specific B idiom. In fact any model based refinement formalism that admits a suitably rich notion of data refinement can incorporate the same or similar ideas, relatively easily.

The caveat in all of the preceding methodologies is the innocent looking phrase 'successive features are built smoothly upon the facilities offered by their predecessors'. However, the reality is that it is manifestly not the case that features necessarily conform to such a convenient discipline. Especially in telecommunications engineering [Zave (2001)], providers invent new features that telephone systems might offer, without constraining their imaginations regarding how the new features might interact with existing functionality: that is left as a challenging and important problem for system integrators. In other words, feature engineering in the real world is frequently

a brown field activity, outside the scope of the clean disciplines that would seek to organise the features to be implemented in a top down way. In particular this is because releases of the system at different times require different collections of features, and there is no *a priori* reason why these different collections should necessarily conform to the required refinement framework. The same is going to be true for almost any long lived digital product, as time and market pull demand changes in functionality that can contradict earlier behaviour.

In the face of such functional anarchy, which inevitably has to face situations where the new system being developed must contradict some properties possessed by its predecessor, refinement is rather hamstrung in what it can offer as an encompassing development methodology, since contradicting what has already been established is anathema for any refinement technique. In this regard, the more indulgent ways of retrenchment [Banach et al. (2007), Banach et al. (2008), Banach and Jeske (2009a), Banach and Jeske (2009b)] offer more scope for giving an account of the process that bears some relation to what is actually done by the engineers.

What makes retrenchment especially useful for this purpose is its compatibility with model based refinement [Banach and Jeske (2009a), Jeske (2005)], and the fact that suitably formulated, it defaults to refinement when the deviation between the two systems being related is close enough [Banach et al. (2007)]. These are the qualities we would want in a more flexible account of feature engineering, since, when a new release of a large system is designed, it is clearly not going to be the case that *all* of its functionality contradicts what was present in the previous release, even if *some* of it might. The judicious use of retrenchment in tandem with refinement allows those portions of the relationship between the two systems that are close, to enjoy stronger properties, while those portions of the relationship in which the two systems differ more drastically are not excluded from the formal account.

In this paper, we give an account of (what we call) *simple* feature engineering, possessing a degree of rigour comparable to what is seen in purely refinement based developments, yet accomodating the vagaries of a realistic engineering process via the use of retrenchment.

The remainder of the paper is structured as follows. Section 2 gives the basic definitions of systems and operations, these being built on straightforward transition system concepts. Section 3 introduces features, making precise the sense in which this paper is concerned with *simple* feature engineering. Section 4 gives a relatively informal overview of our feature engineering process, pointing out how subsequent sections deal with the various issues identified in a more formal way. It also introduces a number of examples. Two examples are used periodically in the later material as running examples to illustrate specific technical points as they come up. A third example is a small self-contained telephony example to illutrate the process as a whole. A final example indicates what non-simple feature engineering might entail.

Then come the technical sections of the paper. In Section 5 we define refinement and retrenchment and the properties of the two that we need later. Section 6 covers regular relations and their properties, especially sequential composition. Section 7 introduces a small language for feature manipulation, giving rise to the feature oriented description (FOD) of operations. The more important properties of the FOD are described. In Section 8 we investigate the evolution of FODs of operations via a rewriting formalism for FODs that allows changes in an operation to be dealt with other

than by structural induction. In Section 9 we show how the changes formulated using the techniques of Section 8 can be captured using the retrenchment machinery introduced earlier, and we describe the properties of those retrenchments, specifically that they are neat default retrenchments. We also consider feature engineering processes that consist of sequences of overrides, pointing out their particular properties. Thus we fulfil the promise of this paper. Section 10 concludes.

2 Systems and Operations

Throughout the paper, our deliberations will be about systems and relationships between pairs of systems. In such a pair of systems, one system will typically be referred to as the abstract system *Abs*, the other as the concrete system *Conc*.

The abstract system has a set of operation names $\mathsf{Ops}_{\mathrm{A}}$, with typical element Op_{A}. An operation Op_{A} will work on the abstract state space U having typical element u (the before-state), and an input space $\mathsf{I}_{Op_{\mathrm{A}}}$ with typical element i. Op_{A} will produce an after-state typically written u', once more in U, and an output o drawn from an output space $\mathsf{O}_{Op_{\mathrm{A}}}$. Initial states are those that satisfy the property $Init_{\mathrm{A}}(u')$. In this paper we work exclusively in a transition system framework, so an operation Op_{A} is given by its transition or step relation. Individual steps are written u -(i, Op_{A}, o)-› u', while the set of all steps belonging to the operation Op_{A} is collectively referred to as $stp_{Op_{\mathrm{A}}}(u, i, u', o)$.

At the concrete level we have a similar setup. Typical operation names are $Op_{\mathrm{C}} \in \mathsf{Ops}_{\mathrm{C}}$. The states are $v \in \mathsf{V}$, inputs are $j \in \mathsf{J}_{Op_{\mathrm{C}}}$, and outputs are $p \in \mathsf{P}_{Op_{\mathrm{C}}}$. Initial states satisfy $Init_{\mathrm{C}}(v')$. Typical individual transitions are written v -(j, Op_{C}, p)-› v', being elements of the concrete step relation $stp_{Op_{\mathrm{C}}}(v, j, v', p)$.

3 Operations and Features

In this section we introduce features, and clarify the distinction between operations and features.

An operation is intended to be a structural unit of the system, eg. a command that can be called at the system's external interface, or a procedure that can be called at an internal API interface. As is clear from Section 2, an operation consists of a set of transitions, the idea being that (at the given level of abstraction) the call of an operation runs and returns having executed a single transition of the system. The main property we require of an operation is *model completeness*. This states that the operation is able to execute a transition whenever it makes sense for its environment (or an internal scheduling policy, if appropriate) to demand one. Thus a system defined using a model complete set of operations prescribes well defined responses to any demand that the environment (or internal scheduler) may reasonably make of it — in particular, it contains no underspecification.

We capture the set of before-states and inputs at which it makes sense for a transition to be demanded from an operation, as the operation's contextual precondition, written ctxpre(Op) for an operation Op. We assume that ctxpre(Op) is supplied externally to the definition of the operation, as one of the outputs of a suitable requirements engineering process.

Note that model completeness of an operation does not imply totality.

Definition 3.1 An operation *Op* is model complete iff:

$$\mathrm{ctxpre}(Op) \;\Rightarrow\; \mathrm{dom}(Op) \tag{3.1}$$

Henceforth we will assume all operations are model complete. We insist on this, since, unlike in those refinement developments which take place as uninterrupted activities (during which an underspecified system model is acceptable as a description of a partially developed system since it is never 'released in the real world'), we understand feature (re-)engineering to take place between systems which *are* 'released in the real world', and which must therefore supply all appropriate responses (and only appropriate responses) to all permitted demands.

Whereas an operation is a structural concept, a feature (in the most general sense) is intended to be a subset of the system's functionality that is coherent as regards meeting an external system goal. That is, from a requirements perspective, a feature will meet the goal in a way that makes sense to the system's users. In principle, meeting such a goal may impinge upon the activities of several operations, but we will work under the simplifying assumption that even if meeting such a requirements goal impacts several operations, it is sufficient to treat each operation in isolation, i.e. to treat each operation as if it alone were affected by a feature unique to itself. This is the essence of *simple* feature engineering.

Definition 3.2 A feature is simple iff it affects only a single operation. Simple feature engineering consists of developing systems using simple features alone.

The quesiton for the wider requirements arena then becomes: under what circumstances it is sufficient to neglect the functional interdependencies between aspects of different operations (in the context of trying to cater for a given external system goal), and under what circumstances it is insufficient to do so? There has to be enough confidence in the understanding of the problem domain to permit the view that no issues significant for the development of features are lurking in the complex interactions between individual operations, i.e. there are no important development issues that cannot be mastered by considering features and operations one at a time. The 'simple feature engineering' of this paper's title refers to the feature engineering of systems that can be successfully accomplished under this simplifying assumption. Henceforth we are concerned exclusively with simple feature engineering.

Given that we define systems using transitions, a feature must also consist of a set of transitions. In contrast to operations, and given that the structural unit of the system is the operation, we do *not* require features to be model complete — only some of the demands of an operation might be relevant to a given feature. Provided we have in mind the same level of abstraction for both operations and features, we arrive at the following.

Definition 3.3 At a given level of abstraction, a simple feature is a subset of the transitions belonging to an operation.

It thus follows that an operation consists of features, suitably combined. Equally, there is nothing in Definition 3.3 to preclude some operation from consisting of just a single feature. Notationally, we can use the conventions established in Section 2 to discuss both operations and features. Of course, a production level software engi-

neering environment will have concrete syntax for capturing both operations and features; whereupon we assume there will be syntactic means for casting a feature into an operation when necessary.

4 Simple Feature Engineering

In this section we overview the kind of system development process our technical framework will permit. We also introduce and overview some examples, the details of which will be used to illustrate the technical content in various ways.

Any realistic development process will contain both informal and formal parts. At the time of initial conception, everything is still informal, whereas any code that is finally produced is necessarily a formal artifact. In between, there is a whole range of activities, and of ways of arranging these activities in relation to each other, and for many of them, there is a choice as regards how formally or not they are carried out. In this paper we do not intend to be prescriptive about such matters of large scale software engineering methodology, but to explore the consequences of approaching those parts which are concerned with (simple) feature manipulation from a formal perspective (under our definition of simple feature, and under what we regard as reasonable assumptions about it).

Since the focus here is on the manipulation of simple features, the manipulations we are concerned with, are able to act as individual ingredients in a range of different software engineering methodologies. For instance, in the context of a waterfall approach, which posits a fairly linear arrangement of stages from initial conception to implementation, the feature manipulations we envisage represent something of a sideways assault. This is a reasonable picture of a re-engineering process which has to be able to reconsider arbitrary aspects of a product design — in response to external market forces say. Such reconsiderations will in general not be smoothly aligned with the preceding design, and a more disruptive design technique (which we underpin with retrenchment) is needed. By contrast, in a spiral approach, such reworkings of what exists already are intrinsic to the methodology, and the disruptive design technique that our feature engineering approach will turn out to embody, is the thing that connects successive loops of the spiral in the design quadrant, as successive use cases are integrated into the design, and their conflicts with use cases already present are resolved.

Refinement and retrenchment of operations (the technical details of which are discussed in Section 5) underpin the formal parts of the development process as just indicated. Thus the main point of the paper is to investigate the further consequences of the additional feature oriented structure.

As mentioned in the Introduction, the sole use of refinement as formal development technique has the strong tendency to impose a linear, waterfall-like, structure on the development activity. This is because successive stages of a refinement process need to be conservative extensions of their predecessors, logically. In such a situation, any rework or modification of any of the system models involved, almost invariably entails starting the whole process afresh, at least from the logical point of view (even if more expedient measures are used in practice).

As also mentioned in the Introduction, retrenchment is much more flexible, and permits much more wide-ranging interventions to be inflicted on the system modelling

process. We therefore regard it as a vital ingredient in any formal account of system development that hopes to make good contact with what is actually done in reality. The focus of the paper thus becomes to capture a single development step using refinement and retrenchment from a feature oriented standpoint, and to explore the properties of such a description.

The first step is to make precise how an operation is composed out of features. This is the job of Section 7. A small language for feature composition is introduced, and its properties are investigated. Bearing in mind the remarks above, the main thing that the approach must achieve, is to allow manipulation of expressions in the language by means other than stuctural induction, because the engineering process may demand alterations to parts of the feature oriented description of an operation (as the expressions in the language are called) that lie buried arbitrarily deeply in the structure. To this end, a normal form theorem is proved (and some of its consequences are explored), which shows that no matter how complex a feature oriented description of an operation may be, it is equivalent to a description using an expression that has a very shallow syntactic structure. This means that *any* intervention to the feature oriented description may be viewed as being inflicted at a shallow level.

Once we have our descriptive mechanisms in place, we can consider how an operation built out of features can change during a development step, which is dealt with in Section 8. Various things can happen. New features can be added, existing features can be modified, and other old features can be eliminated completely. The changes in the community of features contributing to an operation, and in their individual natures, in general will impact the signature of the operation, which complicates the description of the passage from one development stage to its successor. Thus the technical description of a development step is broken up into small pieces.

First, the signature is enlarged to accomodate any new types needed for the operation-to-be. Second, the feature oriented description is changed using instances of two generic modification mechanisms — a change of conditional (in conditional subexpressions) and a term rewrite applied in the interior of the feature expression. Third, any now redundant types in the signature are discarded. Fourth any needed change in data representation can finally be imposed. It turns out that the addition and removal of types from the signature, when separated from the other aspects of operation evolution, yields relations between the before- and after-signatures that are regular (in the technical sense of the word). The material on regular relations needed for this paper is prepared beforehand in Section 6.

Although the preceding captures development steps of feature oriented descriptions, it does not do it by means of retrenchment yet. Section 9 is concerned with showing that the development process can be made to fit the retrenchment mould. In principle, it would be possible to do this 'as is' —indeed the retrieve, within and output relations of such a retrenchment can readily be constructed as regular relations— but our aim is to show that with a couple of mild assumptions, the last relation contributing to the retrenchment, namely the concedes relation, can be constructed in such a way that the retrenchment as a whole satisfies the criteria for being a 'neat' retrenchment.

The neatness property of a retrenchment attests to a particular kind of separation between what we might term 'conforming' and 'nonconforming' behaviours — where by conforming we mean the recognisable persistence of previous behaviour (attributable, in our case, to the persistence of the same feature) in the new model, and by

nonconforming we mean the clean replacement of previous behaviour by new behaviour (attributable, in our case, to the introduction of a new feature that has come into play).

Regarding the assumptions we need, we firstly assume that model completeness maps smoothly through a development step. Secondly we assume that different features do not get to overlap during the development step, i.e. if some feature's functionality survives the development step, then it is *the same feature* that performs that functionality, and not some other feature. Thirdly we assume that different features do not overlap at a single level of abstraction, i.e. when an operation executes from some before-state and input, there is no ambiguity about which feature is responsible for the outcome. With these assumptions, it is plausible that the kind of separation between conforming and nonconforming behaviours indicated above can be established.

Finally, it is noted that a very natural way of building up an operation out of features is to add features sequentially, making each added feature override the functionality of any predecessors with which it overlaps. This pattern is discussed in the light of the preceding results.

Various examples, which we introduce now, illustrate the technical material. Two running examples crop up sporadically in the body of the ensuing text. Neither of the two illustrate every technical point mentioned above, but where either of them does, an increment of its development is interleaved into the technical presentation at the appropriate point. A third example is treated here in its entirety, albeit without the benefit of all the technical details of the theory to follow. This helps to give an early overview of the capabilities of the theory. A fourth example illustrates typical issues that can pertain to 'non-simple' feature interactions.

Example I (Light Switch) Consider a light switch on the wall of a room. Normally it can just switch the light on and off. We will use this to exemplify some elements of the theory developed below. For now, consider that the switch can only be in the '*on*' or '*off*' positions. If it is already in the '*on*' position, it is not sensible to make a further attempt to switch it on. So, if *SwitchOn* is the name of the switching on operation, the '*on*' state would be excluded from ctxpre(*SwitchOn*). Later in the paper we extend this example to introduce a dimmer feature that requires a non-trivial change of data representation.

Example II (Car Locking) Consider a mechanical car central locking system. If the car is already '*locked*', one cannot apply a further turn of the key in the locking direction to attempt to lock the car a further time. So, if *Lock* is the name of the locking operation, the '*locked*' state would not be in ctxpre(*Lock*). We take the central locking system example on a more extensive development excursion. The simple *Lock* and *UnLock* operations are refined to a level of detail involving a number of individual doors. Subsequently, mechanical locking is further enhanced to electronic locking, which permits the locking and unlocking of the car using a wireless key fob. The way that all this is handled by the feature formalism is brought out in a number of excerpts, as we develop the theory.

Example III (Call Forwarding in Simple Telephony) We consider a small feature engineering problem from the telephony domain, as an illustration of the whole feature engineering process. We start with a rather primitive connection model. It is

based on the state variable *calls*, which is a partial injection (notation $\rightarrowtail\!\!\!\mapsto$) on the set of available phones NUM, in which the domain and range of the active calls relation do not intersect:

$$calls : \mathsf{NUM} \rightarrowtail\!\!\!\mapsto \mathsf{NUM} \;\text{ where }\; \mathrm{dom}(calls) \cap \mathrm{rng}(calls) = \varnothing \tag{4.1}$$

Defining $\mathrm{free}(n) \equiv n \notin (\mathrm{dom}(calls) \cup \mathrm{rng}(calls))$, connection itself is given as follows:

$$\begin{array}{l} calls \text{ -}((n, i), Connect, o)\text{-}\!\!\rightarrow calls' \;\text{ where} \\ \quad \mathrm{free}(n) \wedge (\text{if } \mathrm{free}(i) \wedge (n \neq i) \\ \qquad\qquad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto i\} \\ \qquad\qquad \text{else } o = NO \wedge calls' = calls) \end{array} \tag{4.2}$$

This says that a connection can be instigated from a free phone number *n*, and if the desired destination *i* is both free and distinct from the originator, then the connection is made, otherwise it is refused; the output *o* models the system response heard in the earpiece in the two cases. We can regard *Connect* as a feature, since it models a straightforwardly self-contained piece of functionality, and also as an operation, since it corresponds naturally enough to something exposed in its entirety at the user interface. Its domain is the set of pairs (*calls*, (*n*, *i*)) where *calls* is a set of currently active calls, and *n* is a currently free number (and *i* is unrestricted).

The next goal is to introduce a rather primitive call forwarding feature. It depends on a forwarding table *fortab*, which is a partial function (notation $\mapsto$) on the set of available phones NUM, in which the domain and range of *fortab* do not intersect:

$$fortab : \mathsf{NUM} \mapsto \mathsf{NUM} \;\text{ where }\; \mathrm{dom}(fortab) \cap \mathrm{rng}(fortab) = \varnothing \tag{4.3}$$

Now forwarding is given as follows:

$$\begin{array}{l} calls \text{ -}((n, i), Forward, o)\text{-}\!\!\rightarrow calls' \;\text{ where} \\ \quad \mathrm{free}(n) \wedge \neg\mathrm{free}(i) \wedge i \in \mathrm{dom}(fortab) \wedge \\ \quad fortab(i) = z \wedge \mathrm{free}(z) \wedge (z \neq n) \wedge \\ \quad o = OK \wedge calls' = calls \cup \{n \mapsto z\} \end{array} \tag{4.4}$$

We make the requirements level observation that forwarding is not something that one invokes directly. Rather, it is a consequence of doing something else, namely trying to connect a call. So, this time, *Forward* is a feature which is not an operation in its own right, but is merely a contributor to another activity, an enhanced version of the connect operation, $Connect_{CF}$, defined as:

$$\begin{array}{l} calls \text{ -}((n, i), Connect_{CF}, o)\text{-}\!\!\rightarrow calls' \;\text{ where} \\ \quad \mathrm{free}(n) \wedge (\text{if } \mathrm{free}(i) \wedge (n \neq i) \\ \qquad\qquad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto i\} \\ \qquad\qquad \text{else if } \neg\mathrm{free}(i) \wedge i \in \mathrm{dom}(fortab) \wedge \\ \qquad\qquad\qquad fortab(i) = z \wedge \mathrm{free}(z) \wedge (z \neq n) \\ \qquad\qquad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto z\} \\ \qquad\qquad \text{else } o = NO \wedge calls' = calls) \end{array} \tag{4.5}$$

It is relatively clear that $Connect_{CF}$ is obtained by overriding that part of the functionality of *Connect* that is in the domain of *Forward* by the functionality of *Forward*. In terms of the feature manipulation language of Section 7.1 we express this thus:

$$Connect_{CF} = Connect \ntriangleleft Forward \tag{4.6}$$

Example IV (Chinese Text Display System) We consider, in outline, as a candidate for *non*-simple feature engineering, a system for displaying Chinese text, and enhancing it with some novel features. Chinese text consists of characters, which are of uniform width and are uniformly spaced when printed. This tends to suggest that all the characters have independent significance in a string of text. Each character is pronounced using a single syllable (Pinyin being the standard coding of these syllables in the Roman alphabet), and each syllable is endowed with one of four tones or the neutral tone. The Chinese language itself, however, is polysyllabic, and although most characters are associated with a single syllable and a single tone, many characters can take on more than one syllable and/or more than one tone. The polysyllabic nature of the language makes the determination of the correct syllable and tone context dependent.

We envisage some enhancements to a system for displaying normal black and white Chinese text, the idea being to make the display more informative for the novice learner. One enhancement which can be made, is to use colour to indicate the required tone. This is modelled by an operation *ToneCol* that displays each character in the colour corresponding to the most common tone for that character. Another enhancement aids reading by showing the romanised syllables corresponding to the characters in the text. This is modelled by an operation *Pinyin* that displays the Pinyin corresponding to the most common syllable for that character in small text above the character. A third enhancement which can be made, aids reading by highlighting the polysyllabic words. This is modelled by an operation *Parse* that underlines the corresponding groups of characters in the text.

We can envisage increasingly sophisticated versions of the *Parse* operation, that can cope with increasingly complex ambiguities in Chinese. However, the feature engineering of such enhancements to *Parse* cannot be performed independently of other operations, since the deciphering of the real meaning of the text can throw up situations in which characters in the text acquire different tones and pronunciations. Thus the design and any subsequent re-engineering of *Parse* affects the working of both *ToneCol* and *Pinyin*, and in this sense, such feature engineering of *Parse* is not simple in our sense. Of course, the impact on *ToneCol* or on *Pinyin* owing to modifications of *Parse* can be *described* well enough by our simple feature engineering framework. However, the point is that the simple feature engineering framework provides no *additional concepts* to ensure that any such modifications are consistent in some appropriate sense with those of *Parse*.

5 Refinement and Retrenchment

In this section we introduce our notions of refinement and retrenchment, and those of their properties that we need in the sequel.

5.1 Retrenchment

In our transition system framework, a retrenchment between two systems *Abs* and *Conc*, is defined by three facts. Firstly, we demand that $\mathsf{Ops}_{\mathrm{AC}} \equiv \mathsf{Ops}_{\mathrm{A}} \cap \mathsf{Ops}_{\mathrm{C}} \neq \varnothing$, i.e. there is a non-empty set of pairs of abstract and concrete operations, assumed identified by having the same name.[1] Secondly, we have relations as follows: there

is a single retrieve relation $G(u, v)$ between the abstract and concrete state spaces;[2] and for each operation $Op \in \mathsf{Ops}_{AC}$, we have within, output and concedes relations: $P_{Op}(i, j, u, v)$, $O_{Op}(o, p; u', v', i, j, u, v)$ and $C_{Op}(u', v', o, p; i, j, u, v)$ respectively. The within,[3] output and concedes relations are over the variables shown, i.e. the within relations involve the inputs and before-states, while the output and concedes relations involve predominantly the outputs and after-states, though inputs and before-states can also feature if required.[4] We suppress the 'A' and 'C' subscripts on *Op* in these relations since they concern both levels of abstraction equally. Thirdly a collection of properties (the proof obligations or POs) must hold. The initial states must satisfy:

$$Init_C(v') \;\Rightarrow\; (\exists\, u' \bullet Init_A(u') \wedge G(u', v')) \tag{5.1}$$

and for every corresponding operation pair Op_A and Op_C, the abstract and concrete step relations must satisfy the operation PO:

$$\begin{aligned} &G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_C}(v, j, v', p) \;\Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_A}(u, i, u', o) \wedge ((G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \vee \\ &\qquad C_{Op}(u', v', o, p; i, j, u, v))) \end{aligned} \tag{5.2}$$

The retrenchment POs give a good idea of what the components of the retrenchment data are for. Thus the retrieve relation plays a conventional role, relating the two state spaces. The within relation acts as a constraint, limiting the scope of what the retrenchment is able to claim. The output relation strengthens the retrieve relation in the conclusion when the latter is re-established by the PO, allowing more incisive statements to be made when needed. Finally, the concedes relation permits a description of the state of affairs when re-establishing the retrieve relation in the conclusion fails. It is this last aspect which is most characteristic of retrenchment, and which most differentiates it from various flavours of model based refinement.

Associated with the operation PO is the retrenchment simulation relation given by removing the quantification, and changing the implication to a conjunction:

$$\begin{aligned} &G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_C}(v, j, v', p) \wedge stp_{Op_A}(u, i, u', o) \wedge \\ &((G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \vee C_{Op}(u', v', o, p; i, j, u, v)) \end{aligned} \tag{5.3}$$

The simulation relation is what we get by removing the 'don't care' interpretation of the implication in (5.2).

5.2 Refinement

Given the preceding, refinement is (for us) given by a simplification. Firstly, $\mathsf{Ops}_A = \mathsf{Ops}_C = \mathsf{Ops}$, i.e. there is a bijection between abstract and concrete operations, indicated by name identity. Secondly, we have (simpler than before) relations: a retrieve relation $G(u, v)$ between abstract and concrete state spaces; and for each operation $Op \in \mathsf{Ops}$, input and output relations: $In_{Op}(i, j)$, $Out_{Op}(o, p)$ respectively. This

1. This confirms that the 'A' and 'C' subscripts on operation names are meta level tags, suppressed when not needed.
2. $G(u, v)$ can also be referred to as the *gluing* relation.
3. $P_{Op}(i, j, u, v)$ can also be referred to as the *provided* relation.
4. We note that the semicolons in O_{Op} and C_{Op} are purely cosmetic, separating the variables 'of most interest' from others which are permitted, if seldom needed in practice.

time the input and output relations are over the input and output variables alone. For our version of refinement, the retrenchment POs simplify to the following:

$$Init_{\mathrm{C}}(v') \ \Rightarrow\ (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge G(u', v')) \tag{5.4}$$

$$\begin{aligned} & G(u, v) \wedge In_{Op}(i, j) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \ \Rightarrow \\ & \quad (\exists\, u', o \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge G(u', v') \wedge Out_{Op}(o, p)) \end{aligned} \tag{5.5}$$

and the simulation relation simplifies in the analogous way:

$$\begin{aligned} & G(u, v) \wedge In_{Op}(i, j) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \wedge \\ & stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge G(u', v') \wedge Out_{Op}(o, p) \end{aligned} \tag{5.6}$$

We observe that the condition $\mathsf{Ops}_{\mathrm{A}} = \mathsf{Ops}_{\mathrm{C}}$ rather flies in the face of the discussion in Section 1, in that superposition refinement invariably admits the introduction of new operations at the refined level, to handle aspects of the newly introduced functionality that do not easily lend themselves to being absorbed into existing operations. We argue though, that this is not problematic. On the most obvious level, one can reformulate the introduction of such new operations as additional retrenchments if desired. Slightly more subtly, the interworking of refinement and retrenchment, outlined in Section 5.3 and treated in depth in [Banach and Jeske (2009a)], is designed in such a way that adding such new operations, together with the attendant POs that go along with them (eg. POs that guarantee relative deadlock freedom in the refined system) can be done seamlessly, without spoiling the theory. So for simplicity's sake, and because refinement in its own right plays a relatively small role in this paper, we continue with the simpler picture.

5.3 Refinement and Retrenchment Interworking, the Tower

We envisage system development (whether explicitly feature based or not) to consist of a number of stages, some refinements, some retrenchments. For simplicity, let us assume that refinements change the level of abstraction towards an implementation, but that retrenchments maintain the level of abstraction, being concerned with system evolution at a single level (we will relax this assumption shortly). Then we can arrange these stages into a diagram, with refinements as arrows going downwards, and retrenchments as arrows going horizontally. This suggests a grid-like pattern, the *Tower Pattern*, into which individual stages can be placed. Fig. 1.(b) shows an outline development consisting of four refinements and four retrenchments, interleaved in one particular way.

To limit the potential anarchy arising from doing different development stages in different orders, we would want paths through the grid that are different but that have the same end points, to be coherent in some sensible way. The fundamental results about this are treated in detail in [Banach and Jeske (2009a)]. That paper states and proves a full suite of square completion and factorisation theorems regarding the basic commuting square in Fig. 1.(a), which concerns four systems, A, B, C, D, and the two retrenchments and two refinements connecting them. Thus if in Fig. 1.(a) we do not have say, B and $Ret_{A,B}$ and $Ref_{B,D}$, then they can be constructed canonically from the remainder via the Lifting Theorem. In [Banach and Jeske (2009a)] the Lifting Theorem is proved by factorising a 'non-horizontal' generic retrenchment from A to D. This enables the treatment of retrenchments that change the level of abstraction to be dealt with by the same means.

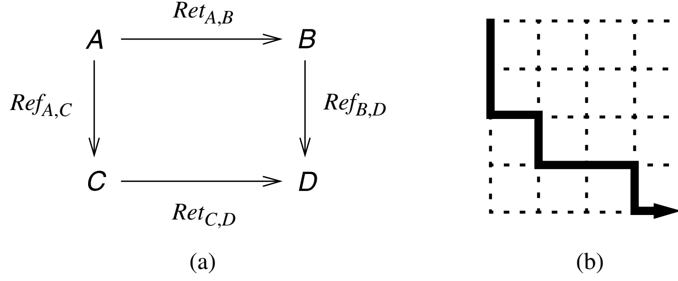

Fig. 1. The basic Tower commuting square, and an outline development.

Particularly when we deal with refinement and retrenchment compatibility, it is useful to demand the 'healthiness condition' in (5.7) for retrenchments (where dom(R) is the domain of relation R — for transition relations such as $stp_{Op_{\mathrm{A}}}$, it is the set of before-state/input pairs from which a transition of Op_{A} issues). However, since the hypotheses of (5.7) effectively weaken those of the retrenchment operation PO, it can often be ignored if the discussion is purely focused on the operation PO.

$$G(u, v) \wedge P_{Op}(i, j, u, v) \;\Rightarrow\; \mathrm{dom}(stp_{Op_{\mathrm{A}}}) \wedge \mathrm{dom}(stp_{Op_{\mathrm{C}}}) \tag{5.7}$$

5.4 Default Retrenchments

Default retrenchments make precise the intuition that 'an arbitrary pair of systems' can be related by retrenchment. Since default retrenchments arise in a generic manner, they can be used to give generic treatments of many situations via retrenchment. We will use them in our treatment of feature engineering below. The following is adapted from [Banach et al. (2007)].

Proposition 5.1 Suppose given two systems *Abs* and *Conc*. Let $\mathsf{Ops}_{\mathrm{AC}} = \mathsf{Ops}_{\mathrm{A}} \cap \mathsf{Ops}_{\mathrm{C}}$. Let $G(u, v)$ and $\{P_{Op}(i, j, u, v), O_{Op}(o, p; u', v', i, j, u, v) \mid Op \in \mathsf{Ops}_{\mathrm{AC}}\}$ be arbitrary relations in the variables stated. Let default within and concedes relations $\{P^{\mathrm{Def}}{}_{Op} \mid Op \in \mathsf{Ops}_{\mathrm{AC}}\}$ and $\{C^{\mathrm{Def}}{}_{Op} \mid Op \in \mathsf{Ops}_{\mathrm{AC}}\}$ be given by:

$$\begin{aligned} &P^{\mathrm{Def}}{}_{Op}(i, j, u, v) \;\equiv \\ &\quad (G(u, v) \wedge P_{Op}(i, j, u, v) \wedge \\ &\qquad (\exists\, u', o, v', p \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p))) \end{aligned} \tag{5.8}$$

$$\begin{aligned} &C^{\mathrm{Def}}{}_{Op}(u', v', o, p; i, j, u, v) \;\equiv \\ &\quad (G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge stp_{Op_{\mathrm{C}}}(v, j, v', p) \wedge \\ &\qquad \neg\, (G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \end{aligned} \tag{5.9}$$

Then G and $\{P^{\mathrm{Def}}{}_{Op}, O_{Op}, C^{\mathrm{Def}}{}_{Op} \mid Op \in \mathsf{Ops}_{\mathrm{AC}}\}$ define a retrenchment from *Abs* to *Conc* called the default retrenchment from *Abs* to *Conc*.

Note that if the healthiness condition (5.7) holds, then the default within relation is identical to the original one.

5.5 Neat Retrenchments

Besides default retrenchments, below we also need another kind of retrenchment, the neat retrenchment.

Definition 5.2 Suppose given a retrenchment between two systems *Abs* and *Conc*, with the usual notations. Then the retrenchment is neat iff for all $Op \in \mathsf{Ops}_{AC}$:

$$\mathrm{pre}^{\mathrm{Ret}}{}_{Op}(u, i, v, j) \wedge \mathrm{pre}^{\mathrm{Con}}{}_{Op}(u, i, v, j) \ \equiv\ \mathsf{false} \tag{5.10}$$

where:

$$\mathrm{pre}^{\mathrm{Ret}}{}_{Op}(u, i, v, j) \ \equiv\ (\exists\, u', o, v', p \bullet \overline{G}_{Op}(u', v', o, p; i, j, u, v)) \tag{5.11}$$

$$\mathrm{pre}^{\mathrm{Con}}{}_{Op}(u, i, v, j) \ \equiv\ (\exists\, u', o, v', p \bullet \overline{C}_{Op}(u', v', o, p; i, j, u, v)) \tag{5.12}$$

where in turn:

$$\begin{aligned} &\overline{G}_{Op}(u', v', o, p; i, j, u, v) \ \equiv \\ &\quad (G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_A}(u, i, u', o) \wedge stp_{Op_C}(v, j, v', p) \wedge \\ &\quad G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \end{aligned} \tag{5.13}$$

$$\begin{aligned} &\overline{C}_{Op}(u', v', o, p; i, j, u, v) \ \equiv \\ &\quad (G(u, v) \wedge P_{Op}(i, j, u, v) \wedge stp_{Op_A}(u, i, u', o) \wedge stp_{Op_C}(v, j, v', p) \wedge \\ &\quad C_{Op}(u', v', o, p; i, j, u, v)) \end{aligned} \tag{5.14}$$

We see in $\overline{G}_{Op}$ and $\overline{C}_{Op}$ the refining and non-refining parts of the retrenchment simulation relation (5.3). The quantifications that lead from $\overline{G}_{Op}$ and $\overline{C}_{Op}$ to $\mathrm{pre}^{\mathrm{Ret}}{}_{Op}$ and $\mathrm{pre}^{\mathrm{Con}}{}_{Op}$ can be seen as deriving certain guards for the joint system that one could construct from *Abs* and *Conc*. The condition (5.10) thus demands that jointly refining and jointly non-refining behaviours of *Abs* and *Conc* are to be kept apart in a particular kind of way. In [Banach and Jeske (2009b)], the neat retrenchments arise as the middle layer in a three level hierarchy of retrenchments that satisfy additional conditions (the tidy, neat and fastidious retrenchments). In that paper their various properties, such as composition and associativity, are explored in depth.

6 Regular Relations and their Properties

If X and Y are sets, then a relation $R : X \leftrightarrow Y$ is just a subset of $X \times Y$ i.e. $R \subseteq X \times Y$.

Definition 6.1 A relation $R : X \leftrightarrow Y$ is regular iff it satisfies one of the following (equivalent) criteria:

(1) There is a set θ and two partial functions $f : X \nrightarrow \theta$ and $g : Y \nrightarrow \theta$ such that $R = f \mathbin{⨾} g^{-1}$ (R is also called difunctional, witnessed by such an f and g).

(2) There are partitions $\{[x_1], [x_2], \dots\}$ of $\mathrm{dom}(R)$, and $\{[y_1], [y_2], \dots\}$ of $\mathrm{ran}(R)$, indexed by a common set I, such that $x\, R\, y$ iff $(\exists\, i \in I \mid x \in [x_i] \wedge y \in [y_i])$ (so there is a bijection between sets in the partitions of $\mathrm{dom}(R)$ and $\mathrm{ran}(R)$, and $x\, R\, y$ iff x and y belong to bijectively related sets, with R restricting to the universal relation on such related pairs of sets).

(3) $R \mathbin{⨾} R^{\mathrm{T}} \mathbin{⨾} R \subseteq R$ (where R^{T} is the transpose of R).

The (relatively selfevident) equivalence of these criteria, as well as their equivalence to various other criteria is discussed in many places, including [Tarski (1941), Suppes (1972), Schmidt and Strohlein (1993), Banach (1994)]. The utility of regular relations in the modelling of digital systems has long been noted, eg. in [Banach (1995)], so their appearance in the present context is entirely unsurprising, and they will figure prominently in the remainder of this paper. We next focus on some compositional properties of regular relations.

Proposition 6.2 Let $R : X \leftrightarrow Y$ and $S : X \leftrightarrow Y$ both be regular.

(1) $R \cap S : X \leftrightarrow Y$ is regular (written $R \wedge S$).

(2) Suppose $\mathrm{dom}(R) \cap \mathrm{dom}(S) = \varnothing = \mathrm{ran}(R) \cap \mathrm{ran}(S)$. Then $R \cup S : X \leftrightarrow Y$ is regular (written $R \uplus S$, union asserted disjoint[5]).

(3) $R \blacklozenge S \,⨟\, ((R \blacklozenge S)^{\mathrm{T}} ⨟ R \blacklozenge S)^* = (R \blacklozenge S ⨟ (R \blacklozenge S)^{\mathrm{T}})^* ⨟ R \blacklozenge S : X \leftrightarrow Y$ is regular (written $\langle R \blacklozenge S \rangle^*$, where $\blacklozenge$ is any binary operator on sets that yields sets of the same type,[6] and $*$ is reflexive transitive closure).

Proof. The arguments for (1) and (2) are obvious. For (3), we note that for any relation $T : X \leftrightarrow Y$, the relation $T ⨟ (T^{\mathrm{T}} ⨟ T)^*$ is easily shown to satisfy Definition 6.1.(3) and thus to be regular. ☺

In contrast to $T ⨟ (T^{\mathrm{T}} ⨟ T)^*$ which is the unique smallest regular relation that contains T, there is no unique largest regular relation that T contains. For a counterexample consider $X = \{x_1, x_3\}$ and $Y = \{y_2, y_4\}$, and $x_i \; T \; y_j$ iff $|i-j| = 1$. T is not regular (it lacks (x_1, y_4)), but *any* proper subrelation of T is regular.

Proposition 6.2 refers to various kinds of ‘parallel composition’ of regular relations, notably to ‘pure’ parallel composition itself, option (1), which trivially encompasses such cases as domain or range restriction. For sequential composition, there is no ‘pure’ (i.e. unconstrained) option, a significant feature of regular relations.

Definition 6.3 Let $R : X \leftrightarrow Y$ be a regular relation. For $y_1, y_2 \in Y$, we write $y_1 \sim_R y_2$ iff $(\exists\, x \in X \mid x \; R \; y_1 \wedge x \; R \; y_2)$, and for $x_1, x_2 \in X$, we write $x_1 \sim_R x_2$ iff $(\exists\, y \in Y \mid x_1 \; R \; y \wedge x_2 \; R \; y)$. Since R restricts to universal relations on corresponding partition sets (Definition 6.1.(2)), $\sim_R$ (in both X and Y) makes these partition sets into cliques. Write $[x]_{\sim_R}$ and $[y]_{\sim_R}$ for the cliques containing $x \in X$ and $y \in Y$.

Now consider two regular relations $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow Z$, whose sequential composition $R ⨟ S$ is also regular. A moment’s thought shows that the cliques in X and Z generated by $\sim_{R;S}$ must, in both the X and Z cases, be generated by a partition of some subset of the partition generated by $\sim_R$ and $\sim_S$ respectively. Thus in X, each clique $[x]_{\sim_{R;S}}$ is the union of one or more cliques $[x]_{\sim_R} \ldots$ (but a given $[x]_{\sim_R}$ clique need not be included in any $[x]_{\sim_{R;S}}$ clique), and similarly for Z.

---

5. In this paper we distinguish ‘union asserted disjoint’ from conventional ‘disjoint union’. Union asserted disjoint is a normal union between sets that happen to be (pairwise) disjoint, and is undefined if they are not pairwise disjoint. See Section 7, particularly Definition 7.1 and Definition 7.2, for further discussion.

6. For the purposes of this paper, it is enough to characterise a type as a given set, or a set formed from existing types by basic combinators, eg. product, relation, or powerset (cf. Z, B).

Each pairing of an X clique $[x]_{\sim_{R;S}}$ with its corresponding Z clique $[z]_{\sim_{R;S}}$ is witnessed by a family of Y cliques $\{[y_1]_{\sim_R}, [y_2]_{\sim_R}, \dots\}$ (which are in bijective correspondence with the $[x]_{\sim_R}$ cliques inside $[x]_{\sim_{R;S}}$) and another family of Y cliques $\{[y_1]_{\sim_S}, [y_2]_{\sim_S}, \dots\}$ (which are in bijective correspondence with the $[z]_{\sim_S}$ cliques inside $[z]_{\sim_{R;S}}$). A moment's thought shows that each such $[y_i]_{\sim_R}$ clique must have non-empty intersection with each $[y_j]_{\sim_S}$ clique (so that $[x]_{\sim_{R;S}}$ can be universally related to $[z]_{\sim_{R;S}}$).

This leads to the following criterion for sequential composability of regular relations.

Definition 6.4 Let $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow Z$ be regular relations. We say that R and S are regular-sequentially (RS) compatible iff the following holds:

$$\begin{aligned}&\{y_{11}, y_{21}, y_{22}\} \subseteq \mathrm{ran}(R) \cap \mathrm{dom}(S) \wedge y_{11} \sim_S y_{21} \wedge y_{21} \sim_R y_{22} \Rightarrow \\ &\quad (\exists\, y_{12} \in \mathrm{ran}(R) \cap \mathrm{dom}(S) \mid y_{11} \sim_R y_{12} \wedge y_{12} \sim_S y_{22}) \qquad (6.1)\end{aligned}$$

An equivalent condition is obtained by appropriately interchanging R and S in (6.1).

Theorem 6.5 Let $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow Z$ be regular relations. Then $R \fatsemi S$ is regular iff R and S are RS compatible.

Proof. Suppose R and S are regular RS compatible relations. We prove that $R \fatsemi S$ is regular. To this end, suppose that $x_1\ (R \fatsemi S)\ z_1\ (R \fatsemi S)^{\mathrm{T}}\ x_2\ (R \fatsemi S)\ z_2$. We must show that $x_1\ (R \fatsemi S)\ z_2$, giving (3) of Definition 6.1. Suppose $x_1\ (R \fatsemi S)\ z_1$ is witnessed by y_{11}, so that $x_1\ R\ y_{11}$ and $y_{11}\ S\ z_1$. Similarly, let $z_1\ (R \fatsemi S)^{\mathrm{T}}\ x_2$ be witnessed by y_{21}, and let $x_2\ (R \fatsemi S)\ z_2$ be witnessed by y_{22}. So $y_{11} \sim_S y_{21}$ (because of z_1), and $y_{21} \sim_R y_{22}$ (because of x_2). Obviously $\{y_{11}, y_{21}, y_{22}\} \subseteq \mathrm{ran}(R) \cap \mathrm{dom}(S)$, so the hypotheses of (6.1) hold. Since R and S are RS compatible, (6.1) gives a $y_{12} \in \mathrm{ran}(R) \cap \mathrm{dom}(S)$ such that $y_{11} \sim_R y_{12}$ and $y_{12} \sim_S y_{22}$. Now, since $y_{11} \sim_R y_{12}$ there is an x such that $x\ R\ y_{11}$ and $x\ R\ y_{12}$. With $x_1\ R\ y_{11}$ we deduce that $x_1\ R\ y_{12}$ because R is regular. Similarly we deduce that $y_{12}\ S\ z_2$. Composing, we get $x_1\ (R \fatsemi S)\ z_2$ witnessed by y_{12}, as required.

For the converse, suppose that R, S and $R \fatsemi S$ are all regular. We must show that R and S are RS compatible. Choose y_{11}, y_{21}, y_{22} to satisfy the hypotheses of (6.1). Since $y_{11} \in \mathrm{ran}(R)$, let $x_1\ R\ y_{11}$. Since $y_{11} \sim_S y_{21}$, let z_1 be such that $y_{11}\ S\ z_1$ and $y_{21}\ S\ z_1$. Similarly let x_2 be such that $x_2\ R\ y_{21}$ and $x_2\ R\ y_{22}$. Since $y_{22} \in \mathrm{dom}(S)$, let $y_{22}\ S\ z_2$. We thus have $x_1\ (R \fatsemi S)\ z_1\ (R \fatsemi S)^{\mathrm{T}}\ x_2\ (R \fatsemi S)\ z_2$. Since $R \fatsemi S$ is regular, $x_1\ (R \fatsemi S)\ z_2$, a fact which must be witnessed by some $y_{12} \in \mathrm{ran}(R) \cap \mathrm{dom}(S)$ such that $x_1\ R\ y_{12}$ and $y_{12}\ S\ z_2$. Now we easily see that $y_{11} \sim_R y_{12}$ and $y_{12} \sim_S y_{22}$ as required. ☺

Regarding the relations G, P_{Op}, O_{Op}, C_{Op}, of a retrenchment (or any relations formed from them, typically using the options sanctioned by Proposition 6.2), by regularity, we mean regularity when they are viewed as relations from the relevant cartesian product of abstract data spaces to the corresponding cartesian product of concrete data spaces.

Definition 6.6 A retrenchment has regular data iff for all operations Op, the relation given by $G(u, v) \wedge P_{Op}(i, j, u, v)$, the relation given by $G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)$, and the relation given by $C_{Op}(u', v', o, p; i, j, u, v)$, are all regular in the sense just mentioned (where in the case of $G \wedge P_{Op}$ and of $G' \wedge O_{Op}$, we implicitly assume that G and G' are extended by appropriate universal relations on the other variables involved, in order that the overall relation has the correct signature). We write the equivalence classes of the domain and range types of these relations using the notation $[u, i]_{G \wedge P}$, $[v, j]_{G \wedge P}$, $[u', o, i, u]_{G' \wedge O}$, $[v', p, j, v]_{G' \wedge O}$, $[u', o, i, u]_C$, $[v', p, j, v]_C$.

Definition 6.7 A retrenchment respects its regular data, iff it has regular data, and the following all hold. For every abstract transition u -(i, Op_{A}, o)-> u', $[u, i]_{G \wedge P}$, $[u', o, i, u]_{G' \wedge O}$, $[u', o, i, u]_C$ all exist, and:

(1) If $(\underline{u}, \underline{i}) \in [u, i]_{G \wedge P}$ and $\underline{u}$ -($\underline{i}$, Op_{A}, $\underline{o}$)-> $\underline{u}'$ is an abstract transition, then for some (u', o), $(\underline{u}', \underline{o}, \underline{i}, \underline{u}) \in [u', o, i, u]_{G' \wedge O}$, and $(\underline{u}', \underline{o}, \underline{i}, \underline{u}) \in [u', o, i, u]_C$.

(2) If $(\underline{u}', \underline{o}, \underline{i}, \underline{u}) \in [u', o, i, u]_{G' \wedge O}$ and $\underline{u}$ -($\underline{i}$, Op_{A}, $\underline{o}$)-> $\underline{u}'$ is an abstract transition, then $(\underline{u}, \underline{i}) \in [u, i]_{G \wedge P}$.

(3) If $(\underline{u}', \underline{o}, \underline{i}, \underline{u}) \in [u', o, i, u]_C$ and $\underline{u}$ -($\underline{i}$, Op_{A}, $\underline{o}$)-> $\underline{u}'$ is an abstract transition, then $(\underline{u}, \underline{i}) \in [u, i]_{G \wedge P}$.

For every concrete transition v -(j, Op_{C}, p)-> v', $[v, j]_{G \wedge P}$, $[v', p, j, v]_{G' \wedge O}$, $[v', p, j, v]_C$ all exist, and:

(4) If $(\underline{v}, \underline{j}) \in [v, j]_{G \wedge P}$ and $\underline{v}$ -($\underline{j}$, Op_{C}, $\underline{p}$)-> $\underline{v}'$ is a concrete transition, then for some (v', p), $(\underline{v}', \underline{p}, \underline{j}, \underline{v}) \in [v', p, j, v]_{G' \wedge O}$, and $(\underline{v}', \underline{p}, \underline{j}, \underline{v}) \in [v', p, j, v]_C$.

(5) If $(\underline{v}', \underline{p}, \underline{j}, \underline{v}) \in [v', p, j, v]_{G' \wedge O}$ and $\underline{v}$ -($\underline{j}$, Op_{C}, $\underline{p}$)-> $\underline{v}'$ is a concrete transition, then $(\underline{v}, \underline{j}) \in [v, j]_{G \wedge P}$.

(6) If $(\underline{v}', \underline{p}, \underline{j}, \underline{v}) \in [v', p, j, v]_C$ and $\underline{v}$ -($\underline{j}$, Op_{C}, $\underline{p}$)-> $\underline{v}'$ is a concrete transition, then $(\underline{v}, \underline{j}) \in [v, j]_{G \wedge P}$.

The following results are easy to show (see [Banach and Jeske (2009b)]).

Proposition 6.8 In a retrenchment which respects its regular data, the abstract and concrete transitions are related by a regular relation.

Corollary 6.9 A default retrenchment which respects its regular data, is neat.

7 A Simple Feature Language

Now we start to draw out the technical consequences of the development process outlined in Sections 2, 3 and 4. In this section we focus on how features may be combined to make operations.

The assumption of *simple* feature engineering permits us to focus on a generic operation Op whose incarnations through the feature oriented development are labelled Op_k, where the signature of Op_k will be $\mathsf{U}_k \times \mathsf{I}_{Op,k} \leftrightarrow \mathsf{U}_k \times \mathsf{O}_{Op,k}$ in the usual way, and where $k = 0, 1, \ldots$ denotes the level of the development. For each k, Op_k is thus assumed model complete. From now on we will suppress or reinstate the level subscript k, depending on whether we are discussing only a single level of the overall development or multiple levels.

Individual features contributing to Op are denoted f_{d} where d indicates the name of the feature (we write $f_{\mathrm{d},k}$ when we need to distinguish the level). Each feature contributing to Op will be a (partial) relation with the same signature as Op, namely $\mathsf{U} \times \mathsf{I}_{Op} \leftrightarrow \mathsf{U} \times \mathsf{O}_{Op}$. The simplicity assumption allows us to pretend henceforth that feature d contributes *only* to Op and not to other operations.

From Definition 3.3 and the remarks surrounding it, it is clear, that since all features that contribute to a given Op have the same signature, features may be combined using any operator on relations that yields a result with the appropriate relational sig-

nature. This leads us to a language for manipulating features built round such operators, whose properties we develop below.

7.1 Feature Expressions and their Normal Forms

We fix on the following rather simple menu of combinators. It is rich enough to show the potential utility of retrenchment based techniques in this area, without detracting through excessive complexity.

Definition 7.1 Feature expression combinators (illustrated working on individual features, but applicable to feature expressions in general).

(1) Union: written (_ ∪ _) ; eg. $(f_{\mathrm{d}} \cup f_{\mathrm{e}})$

(2) Union asserted disjoint: written (_ $\uplus$ _) ; eg. $(f_{\mathrm{d}} \uplus f_{\mathrm{e}})$

(3) Override: written (_ $\lessdot$ _) ; eg. $(f_{\mathrm{e}} \lessdot f_{\mathrm{d}})$

(4) Conditional: written (If _ then _ else _ fi) ; eg. (If $p(u, i)$ then f_{d} else f_{e} fi) also written (_ $\lessdot_p$ _) ; eg. $(f_{\mathrm{e}} \lessdot_p f_{\mathrm{d}})$

(5) Case: written (Case _ of _: _ ; … ; esac) ;
eg. (Case $p(u, i)$ of v_0: f_{d_0} ; v_1: f_{d_1} ; … v_n: f_{d_n} ; else f_{e} ; esac)
also written (_ • _: _ ; … ;; _) ; eg. (p • v_0: f_{d_0} ; v_1: f_{d_1} ; … v_n: f_{d_n} ;; f_{e})

The feature expression combinators are assigned meanings as follows.

Definition 7.2 The semantics of the combinators in Definition 7.1 is given in the following five paragraphs.

(1) Union has the semantics of conventional set theoretic union.

(2) Union asserted disjoint is slightly unusual in that its semantics is that of conventional set theoretic union, but with the side condition that its operand relations have disjoint domains.[7] The elements of a union asserted disjoint can each be mapped back to the set from which they came, without confusion, because of the side condition. This is in contrast to disjoint union proper, which guarantees such a 'birth certificate' property unconditionally, by employing some behind-the-scenes machinery, typically involving the tagging of each element of the disjoint union with some label indicating where it came from. We reject the general construction for our purposes because it always introduces some additional set theoretic machinery which is not specified canonically. In our environment, set theoretic details are supposed to correspond to elements of the application that we are trying to model, and arbitrary unspecified set theoretic mechanisms can have no place. Notations employing union asserted disjoint in which the side condition is not true are undefined.

(4) Conditional composition behaves as expected. When $p(u, i)$ evaluates to true, then the conditional (If $p(u, i)$ then f_{d} else f_{e} fi) is true iff $stp_{f_{\mathrm{d}}}(u, i, u', o)$ is true. When the condition $p(u, i)$ evaluates to false, then the conditional (If $p(u, i)$ then f_{d} else f_{e} fi) is true iff $stp_{f_{\mathrm{e}}}(u, i, u', o)$ is true. And $p(u, i)$ must always evaluate to

7. N.B. Unlike in Proposition 6.2.(2), for union asserted disjoint of *features*, we do not insist that the ranges are disjoint.

one of these, or else the whole expression is not defined. N.B. Note the differing order of the operands in the two notations.

(3) Override is a special case of Conditional in which $p(u, i)$ is $((u, i) \in \mathrm{dom}(stp_{f_\mathrm{d}}))$.

(5) Case is defined in the usual way. We demand that $p(u, i)$ is a (partial) function, yielding at most one value for any u, i, or else the whole expression is not defined. When $p(u, i)$ evaluates to one of the values in $\{v_0 \dots v_n\}$, v_j say, then the case construct is true iff for the relevant j, $stp_{f_{\mathrm{d}_j}}(u, i, u', o)$ is true. Otherwise, i.e. if $p(u, i)$ yields a value not in $\{v_0 \dots v_n\}$, or if $p(u, i)$ is undefined, then the case construct is true iff the (mandatory) 'else clause' $stp_{f_\mathrm{e}}(u, i, u', o)$ is true.

These operators are sufficient to model others we might also want to consider. For example domain restriction and domain subtraction can be modelled by:

$$(S \lhd f_\mathrm{d}) \equiv (\mathrm{If}\ (u, i) \in (\mathrm{dom}(stp_{f_\mathrm{d}}) \cap S)\ \mathrm{then}\ f_\mathrm{d}\ \mathrm{else}\ \varnothing\ \mathrm{fi}) \tag{7.1}$$

$$(S \ntriangleleft f_\mathrm{d}) \equiv (\mathrm{If}\ (u, i) \in (\mathrm{dom}(stp_{f_\mathrm{d}}) - S)\ \mathrm{then}\ f_\mathrm{d}\ \mathrm{else}\ \varnothing\ \mathrm{fi}) \tag{7.2}$$

It is clear from the above that feature expressions can contain subexpressions of two kinds: a subexpression[8] can be a feature subexpression (a FSE, a subexpression of feature type), or a condition subexpression (a CSE, a subexpression of boolean type). We call a subexpression a pure FSE, iff it is not a subexpression of a CSE.

With this collection of feature combination operators at our disposal we can regard the *stp* relation of an operator *Op* as the value of an expression (a feature expression, FE), built out of features and using these combinators. When an operator *Op* is defined by a FE, we call this a feature oriented definition (FOD) of *Op*.

Example I (Light Switch) If the possible states of the light switch are *on* and *off*, and we have two operations given by *off* -(*SwitchOn*)-› *on* and *on* -(*SwitchOff*)-› *off*, then interpreting *SwitchOn* and *SwitchOff* as features (which is always permissible mathematically), we can design the new operation $Toggle \equiv (SwitchOn \cup SwitchOff)$.

Example II (Car Locking) At level 0 we just have the *unlocked* and *locked* states, and and two operations *unlocked* -($Lock_0$)-› *locked* and *locked* -($UnLock_0$)-› *unlocked*. At a finer level of detail, we refine this to level 1, where there are two doors, the driver door and the passenger door. There are operations to lock and unlock the doors individually: for the driver we have (*dr-unl*, $*$) -($DrLck_1$)-› (*dr-lck*, $*$) and (*dr-lck*, $*$) -($DrUnl_1$)-› (*dr-unl*, $*$); and for the passenger ($*$, *ps-unl*) -($PsLck_1$)-› ($*$, *ps-lck*) and ($*$, *ps-lck*) -($PsUnl_1$)-› ($*$, *ps-unl*). In these, $*$ represents indifference to the other component of the state, which remains unchanged during the operation. With these individual operations, the earlier level 0 operations $Lock_0$ and $UnLock_0$ can be refined to $Lock_1$ and $UnLock_1$. $Lock_1$ deterministically locks all unlocked doors, $Lock_1 = ((\{(dr\text{-}unl, ps\text{-}unl)\} \ntriangleleft (DrLck_1 \cup PsLck_1)) \cup LockBoth_1)$, where $LockBoth_1$ is given by (*dr-unl*, *ps-unl*) -($LockBoth_1$)-› (*dr-lck*, *ps-lck*). Contrastingly, $UnLock_1$ nondeterministically unlocks at least one of the two locked doors,[9] so $UnLock_1 = ((\{(dr\text{-}lck, ps\text{-}lck)\} \lhd (DrUnl_1 \cup PsUnl_1)) \cup UnLockBoth_1)$, where we have (*dr-lck*, *ps-lck*) -($UnLockBoth_1$)-› (*dr-unl*, *ps-unl*), and we retained all the parentheses.

8. When we say subexpression, we mean subexpression occurrence (unless otherwise stated).
9. Such nondeterminism is not really acceptable in a description of a model complete user level operation. But let us tolerate it for the sake of a simple example.

We want to emphasise semantic issues in this paper. In this regard, we regard FEs as a fairly abstract syntax for the relations that they denote. So we view FEs as identical if they differ at worst by the renaming of constituent individual features, or by the permutation of the parameters of commutative combinators; otherwise they are regarded as distinct. Thus $f_{\mathrm{d}} \cup f_{\mathrm{e}}$ and $f_{\mathrm{e}} \cup f_{\mathrm{d}}$ are regarded as identical FEs. On the other hand $f_{\mathrm{d}} \cup f_{\mathrm{d}}$ and f_{d} are different FEs which evaluate to the same relation (i.e. they are FEs that are *equivalent*, not identical). All this highlights the fact that the symbols employed for the feature combination operators are used ambiguously, being used as lexical elements of the abstract syntax on the one hand, and as functions that map their arguments to a relation that yields the semantics of the abstract syntax on the other. The context distinguishes the two uses; when speaking of FEs, we invariably have in mind the lexical use.

In future we will write $\mathrm{dom}(f_{\mathrm{d}})$ and $\mathrm{dom}(Op)$ instead of $\mathrm{dom}(stp_{f_{\mathrm{d}}})$ and $\mathrm{dom}(stp_{Op})$. If ϕ is a FE, we write $\mathrm{dom}(\phi)$ for the domain of the relation that ϕ defines.

Theorem 7.3 Every FE ϕ has a normal form, $\mathrm{NF}(\phi)$:

$$\mathrm{NF}(\phi) \equiv \phi_1 \underline{\cup} \phi_2 \underline{\cup} \ldots \underline{\cup} \phi_n = \phi \tag{7.3}$$

such that:

(1) Each ϕ_j is nonempty, unless ϕ itself denotes the empty relation (in which case $\mathrm{NF}(\phi) \equiv \varnothing$).

(2) $\mathsf{U} \times \mathsf{I}_{Op}$ is partitioned into:

$$\mathrm{dom}(\phi_1) \underline{\cup} \mathrm{dom}(\phi_2) \underline{\cup} \ldots \underline{\cup} \mathrm{dom}(\phi_n) \underline{\cup} ((\mathsf{U} \times \mathsf{I}_{Op}) - \mathrm{dom}(\phi)) \tag{7.4}$$

where the last term is omitted if ϕ is total, and all terms except $\mathsf{U} \times \mathsf{I}_{Op}$ are omitted if ϕ is the empty relation.

(3) For each j, $\phi_j = \mathrm{dom}(\phi_j) \lhd (f_{\mathrm{a}_j} \cup f_{\mathrm{b}_j} \cup \ldots \cup f_{\mathrm{z}_j})$ where:

(i) the $f_{\mathrm{a}_j}, f_{\mathrm{b}_j} \ldots f_{\mathrm{z}_j}$ are distinct individual features occuring in the FE ϕ;

(ii) for each f_{b_j}, $\mathrm{dom}(\phi_j) \subseteq \mathrm{dom}(f_{\mathrm{b}_j})$;

(iii) if $j_1 \neq j_2$, then the union expressions for ϕ_{j1} and ϕ_{j2}, $(f_{\mathrm{a}_{j_1}} \cup f_{\mathrm{b}_{j_1}} \cup \ldots \cup f_{\mathrm{z}_{j_1}})$ and $(f_{\mathrm{a}_{j_2}} \cup f_{\mathrm{b}_{j_2}} \cup \ldots \cup f_{\mathrm{z}_{j_2}})$ respectively, differ by at least one individual feature.

Proof. We go by induction on the structure of ϕ. If ϕ is $\varnothing$, or is an individual feature f_{b}, (7.3) is an identity and the remaining conclusions are trivial.

Suppose $\phi \equiv (\phi_{\mathrm{a}} \cup \phi_{\mathrm{b}})$, and suppose $\mathrm{NF}(\phi_{\mathrm{a}})$ and $\mathrm{NF}(\phi_{\mathrm{b}})$ are known. Then since:

$$\mathrm{dom}(\phi_{\mathrm{a}} \cup \phi_{\mathrm{b}}) = \mathrm{dom}(\phi_{\mathrm{a}} \cap \phi_{\mathrm{b}}) \underline{\cup} \mathrm{dom}(\phi_{\mathrm{a}} - \phi_{\mathrm{b}}) \underline{\cup} \mathrm{dom}(\phi_{\mathrm{b}} - \phi_{\mathrm{a}}) \tag{7.5}$$

we can use this decomposition of $\mathrm{dom}(\phi_{\mathrm{a}} \cup \phi_{\mathrm{b}})$ to generate a common refinement of the partitions of $\mathsf{U} \times \mathsf{I}_{Op}$ from the NFs of ϕ_{a} and ϕ_{b}, and to define the ϕ_js belonging to this partition. In a preprocessing phase, every element $\phi_j = (f_{\mathrm{a}_j} \cup f_{\mathrm{b}_j} \cup \ldots \cup f_{\mathrm{z}_j})$ of the NF for ϕ_{a} whose domain intersects both $\mathrm{dom}(\phi_{\mathrm{a}} - \phi_{\mathrm{b}})$ and $\mathrm{dom}(\phi_{\mathrm{a}} \cap \phi_{\mathrm{b}})$ is first split into two across the boundary of $\mathrm{dom}(\phi_{\mathrm{b}})$. This yields $(\mathrm{dom}(\phi_j) - \mathrm{dom}(\phi_{\mathrm{b}})) \lhd (f_{\mathrm{a}_j} \cup f_{\mathrm{b}_j} \cup \ldots \cup f_{\mathrm{z}_j})$ and $(\mathrm{dom}(\phi_j) \cap \mathrm{dom}(\phi_{\mathrm{b}})) \lhd (f_{\mathrm{a}_j} \cup f_{\mathrm{b}_j} \cup \ldots \cup f_{\mathrm{z}_j})$. We do likewise for every element ϕ_j of the NF for ϕ_{b} whose domain intersects both $\mathrm{dom}(\phi_{\mathrm{b}} - \phi_{\mathrm{a}})$ and $\mathrm{dom}(\phi_{\mathrm{a}} \cap \phi_{\mathrm{b}})$. It is clear that this first phase preserves all the desired properties of

the NFs for ϕ_a and ϕ_b, except for (3).(iii) — obviously the two parts of an element that has been split, each contain the same component features. Now, the domain of each element of these modified NFs for ϕ_a and ϕ_b is a subset of one of the following: (i) $\mathrm{dom}(\phi_a \cap \phi_b)$, (ii) $\mathrm{dom}(\phi_a - \phi_b)$, (iii) $\mathrm{dom}(\phi_b - \phi_a)$. The NF for $\phi_a \cup \phi_b$ is now constructed as follows.

Any element $((\mathsf{U} \times \mathsf{I}_{Op}) - \mathrm{dom}(\phi_a))$ or $((\mathsf{U} \times \mathsf{I}_{Op}) - \mathrm{dom}(\phi_b))$ present in the original partitions of $\mathsf{U} \times \mathsf{I}_{Op}$, is refined to an element $((\mathsf{U} \times \mathsf{I}_{Op}) - \mathrm{dom}(\phi_a \cup \phi_b))$ of the new partition, provided it is nonempty.

If $\mathrm{dom}(\phi_{a_{j1}}) \cap \mathrm{dom}(\phi_{b_{j2}}) \neq \varnothing$, where $\phi_{a_{j1}} = \mathrm{dom}(\phi_{a_{j1}}) \lhd (f_{a_{j1}} \cup f_{aa_{j1}} \cup \ldots \cup f_{aaa_{j1}})$ and $\phi_{b_{j2}} = \mathrm{dom}(\phi_{b_{j2}}) \lhd (f_{b_{j2}} \cup f_{bb_{j2}} \cup \ldots \cup f_{bbb_{j2}})$ are elements of the modified NFs of ϕ_a and ϕ_b, then an element of the new NF for $\phi_a \cup \phi_b$ is defined by $(\mathrm{dom}(\phi_{a_{j1}}) \cap \mathrm{dom}(\phi_{b_{j2}})) \lhd (f_{a_{j1}} \cup f_{aa_{j1}} \cup \ldots \cup f_{aaa_{j1}} \cup f_{b_{j2}} \cup f_{bb_{j2}} \cup \ldots \cup f_{bbb_{j2}})$, where duplicate occurrences of features are removed from the union expression. The set $(\mathrm{dom}(\phi_{a_{j1}}) \cap \mathrm{dom}(\phi_{b_{j2}}))$ forms an element of the new partition of $\mathsf{U} \times \mathsf{I}_{Op}$.

Elements of the original NF for ϕ_a whose domains lie wholly inside $\mathrm{dom}(\phi_a - \phi_b)$ form elements of the new NF. Their domains form elements of the new partition of $\mathsf{U} \times \mathsf{I}_{Op}$. Likewise for elements of the original NF for ϕ_b whose domains lie wholly inside $\mathrm{dom}(\phi_b - \phi_a)$.

Since it may happen (in the construction thus far) that more than one element of the new partition of $\mathsf{U} \times \mathsf{I}_{Op}$ is the domain for the same subcollection of individual features occuring in the original FE ϕ, in a postprocessing phase, we amalgamate any such elements belonging to the same subcollection — all the elements belonging to the subcollection are replaced by a single one whose domain is the union of the subcollection's elements' domains.

This completes the construction for the $\phi \equiv (\phi_a \cup \phi_b)$ case. It is easy to see that the construction possesses the claimed properties.

Suppose $\phi \equiv (\phi_a \uplus \phi_b)$. Then we have a simpler version of the preceding.

Suppose $\phi \equiv (\phi_b \lessdot_p \phi_a)$. Let *true*($p$) denote the subset of $\mathsf{U} \times \mathsf{I}_{Op}$ where p is true; let *false*(p) denote its complement. In a preprocessing phase, elements of the NFs of both ϕ_a and ϕ_b are first split into two across the boundary between *true*(p) and *false*(p) in the obvious way. New partition element $((\mathsf{U} \times \mathsf{I}_{Op}) - \mathrm{dom}(\phi_b \lessdot_p \phi_a))$ is generated, provided it is nonempty. The new NF then consists of elements of the NF of ϕ_a (and the corresponding partition elements) whose domains lie entirely in *true*(p) together with elements of the NF of ϕ_b (and the corresponding partition elements) whose domains lie entirely in *false*(p), discarding of course any empty ones, and amalgamating any partition elements that share the same individual feature collections.

Suppose $\phi \equiv (\phi_b \lessdot \phi_a)$. This is a special case of the preceding.

Suppose $\phi \equiv (p \bullet v_0 \colon \phi_{a_0} \,;\, v_1 \colon \phi_{a_1} \,;\, \ldots\, v_n \colon \phi_{a_n} \,;;\, \phi_b)$. This is similar to the preceding.

We are done. ☺

The normal form theorem gives us a vivid picture of what systems built out of features using the combinators of Definition 7.1 look like. Fundamentally, the space of valid before-states and inputs partitions into a collection of subsets, on each of which a well defined subcollection of the features present in the original FE defines the be-

haviour by nondeterministic choice amongst them; essentially this reduces any FE to a case analysis.

Example I (Light Switch) The previously given operation *Toggle* ≡ *SwitchOn* ∪ *SwitchOff* is a union of two features with disjoint domains, so it can also be written as *Toggle* ≡ *SwitchOn* $\uplus$ *SwitchOff*, which is a normal form.

Example II (Car Locking) At level 1, the normal forms of $Lock_1$ and $UnLock_1$ are as follows. For $Lock_1$ we have $Lock_1 = \mathrm{NF}(Lock_1) \equiv (\{(dr\text{-}unl, ps\text{-}lck)\} \lhd DrLck_1) \uplus (\{(dr\text{-}lck, ps\text{-}unl)\} \lhd PsLck_1) \uplus LockBoth_1$. For $UnLock_1$ we have $UnLock_1 = \mathrm{NF}(UnLock_1) \equiv (\{(dr\text{-}lck, ps\text{-}lck)\} \lhd (DrUnl_1 \cup PsUnl_1 \cup UnLockBoth_1))$.

Definition 7.4 A feature expression ϕ is featurewise linear iff any individual feature f_d occurs at most once as a pure FSE of ϕ.

Theorem 7.5 Every FE ϕ is equivalent to a featurewise linear FE ϕ'.

Proof. Suppose $\mathrm{NF}(\phi) = \phi_1 \uplus \phi_2 \uplus \dots \uplus \phi_n$, in which the collection of individual features that occurs in $\phi_1 \dots \phi_n$ is $\{f_a, f_b, \dots, f_z\}$. For all $i \in \{a \dots z\}$, let $\mathrm{dom_{fl}}(f_i) \equiv \bigcup \{\mathrm{dom}(\phi_j) \mid j \in \{1 \dots n\}, f_i$ is an element of the union expression $\phi_j\}$. Then it is easy to see that ϕ is equivalent to:

$$\phi' \equiv (\mathrm{dom_{fl}}(f_a) \lhd f_a) \cup (\mathrm{dom_{fl}}(f_b) \lhd f_b) \cup \dots \cup (\mathrm{dom_{fl}}(f_z) \lhd f_z) \tag{7.6}$$

which is featurewise linear. ☺

Examples I and II (Light Switch and Car Locking) The normal forms derived above all happen to be featurewise linear.

7.2 Active Domains

We can get another handle on where individual feature occurrences (and more general subexpressions of a FOD) determine the behaviour defined, by solving a constraint problem in the manner of attributed grammars. We note that while the domains of individual features strive to describe how those features contribute to the overall operation of which they are a part, the conditions in the Case, Conditional and Override combinators, can curtail this desire by restricting the part of the relevant subexpression's domain that actually remains visible in the operation. We can calculate the tradeoff between these competing forces for each subexpression ϕ, using an inherited attribute $in(\phi)$ and a synthesised attribute $sy(\phi)$, using the semantics of the various combinators to impose appropriate relationships between them. In the most common case, the calculation starts with the most liberal possibility $\mathsf{U} \times \mathsf{I}_{Op}$ at the top, and passes the current least restrictive estimate down via $in(_)$ sets. At the leaves of the parse tree of the FE, these get curtailed by the domains of individual features, resulting in the bottom level $sy(_)$ sets. These get passed up the tree as $sy(_)$ sets of increasingly complex subexpressions. Thus the solution is obtained by a traversal of the parse tree of the FOD Φ, pushing $in(_)$ sets down and picking up $sy(_)$ sets on the return journey. Because we have no feature variables (which might lead to mutually recursive equations) it is clear that for any FOD Φ, the system in Definition 7.6 below can be solved for $sy(\phi)$, provided we know all the needed $\mathrm{dom}(f_c)$ sets, the various conditions p that occur within Φ, and finally $in(\Phi)$, the postulated $in(_)$ set for the top level FE Φ itself.

Definition 7.6 Let Φ be a FE and ϕ a FSE of Φ. The active domain $\mathrm{dom}_{\mathrm{act}}(\phi)/X$ (where $X \subseteq \mathsf{U} \times \mathsf{I}_{Op}$, and / is just (mathematical) punctuation), is given by finding the solution to the set of constraints generated as follows over the structure of Φ:

(1) The root expression Φ has $in(\Phi) = X$.

(2) An individual feature f_{c} has $sy(f_{\mathrm{c}}) = \mathrm{dom}(f_{\mathrm{c}}) \cap in(f_{\mathrm{c}})$.

(3) If $\phi \equiv (\phi_{\mathrm{d}} \cup \phi_{\mathrm{e}})$ then $in(\phi_{\mathrm{d}}) = in(\phi)$, $in(\phi_{\mathrm{e}}) = in(\phi)$, $sy(\phi) = sy(\phi_{\mathrm{d}}) \cup sy(\phi_{\mathrm{e}})$.

(4) If $\phi \equiv (\phi_{\mathrm{d}} \underline{\cup} \phi_{\mathrm{e}})$ then $in(\phi_{\mathrm{d}}) = in(\phi)$, $in(\phi_{\mathrm{e}}) = in(\phi)$, $sy(\phi) = sy(\phi_{\mathrm{d}}) \cup sy(\phi_{\mathrm{e}})$.

(5) If $\phi \equiv (\phi_{\mathrm{e}} \triangleleft\!\!+ \phi_{\mathrm{d}})$ then $in(\phi_{\mathrm{d}}) = in(\phi)$, $in(\phi_{\mathrm{e}}) = in(\phi) - sy(\phi_{\mathrm{d}})$,
$sy(\phi) = sy(\phi_{\mathrm{d}}) \cup sy(\phi_{\mathrm{e}})$.

(6) If $\phi \equiv (\phi_{\mathrm{e}} \triangleleft\!\!+_p \phi_{\mathrm{d}})$ then $in(\phi_{\mathrm{d}}) = in(\phi) \cap true(p)$, $in(\phi_{\mathrm{e}}) = in(\phi) \cap false(p)$,
$sy(\phi) = sy(\phi_{\mathrm{d}}) \cup sy(\phi_{\mathrm{e}})$.

(7) If $\phi \equiv (p \bullet v_0\colon \phi_{\mathrm{d}_0} ;\, v_1\colon \phi_{\mathrm{d}_1} ;\, \ldots\, v_n\colon \phi_{\mathrm{d}_n} ;;\, \phi_{\mathrm{e}})$ then $in(\phi_{\mathrm{d}_j}) = in(\phi) \cap p^{-1}(v_j)$,
$in(\phi_{\mathrm{e}}) = in(\phi) \cap (\mathsf{U} \times \mathsf{I}_{Op} - \bigcup_j (p^{-1}(v_j)))$, $sy(\phi) = \bigcup_j sy(\phi_{\mathrm{d}_j}) \cup sy(\phi_{\mathrm{e}})$.

(8) $\mathrm{dom}_{\mathrm{act}}(\phi)/X = sy(\phi)$.

Theorem 7.7 Let ϕ be a FE and $X \subseteq \mathsf{U} \times \mathsf{I}_{Op}$. Then $\mathrm{dom}_{\mathrm{act}}(\phi)/X = \mathrm{dom}(stp_\phi) \cap X$.

Proof. We proceed by induction on the structure of ϕ.

If $\phi \equiv f_{\mathrm{c}}$, then $\mathrm{dom}_{\mathrm{act}}(\phi)/X = sy(f_{\mathrm{c}}) = \mathrm{dom}(f_{\mathrm{c}}) \cap in(f_{\mathrm{c}}) = \mathrm{dom}(f_{\mathrm{c}}) \cap X = \mathrm{dom}(stp_\phi) \cap X$, giving the base case.

If $\phi \equiv (\phi_{\mathrm{d}} \cup \phi_{\mathrm{e}})$ or $\phi \equiv (\phi_{\mathrm{d}} \underline{\cup} \phi_{\mathrm{e}})$, then by induction:

$$\begin{aligned}
&\mathrm{dom}_{\mathrm{act}}(\phi)/X = sy(\phi) = sy(\phi_{\mathrm{d}}) \cup sy(\phi_{\mathrm{e}}) = \\
&\mathrm{dom}_{\mathrm{act}}(\phi_{\mathrm{d}})/X \cup \mathrm{dom}_{\mathrm{act}}(\phi_{\mathrm{e}})/X = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X) \cup (\mathrm{dom}(stp_{\phi_{\mathrm{e}}}) \cap X) = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cup \mathrm{dom}(stp_{\phi_{\mathrm{e}}})) \cap X = \\
&\mathrm{dom}(stp_{\phi_{\mathrm{d}}} \cup stp_{\phi_{\mathrm{e}}}) \cap X = \\
&\mathrm{dom}(stp_{\phi_{\mathrm{d}} \cup \phi_{\mathrm{e}}}) \cap X = \\
&\mathrm{dom}(stp_\phi) \cap X.
\end{aligned} \tag{7.7}$$

If $\phi \equiv (\phi_{\mathrm{e}} \triangleleft\!\!+ \phi_{\mathrm{d}})$ then by induction:

$$\begin{aligned}
&\mathrm{dom}_{\mathrm{act}}(\phi)/X = sy(\phi) = sy(\phi_{\mathrm{d}}) \cup sy(\phi_{\mathrm{e}}) = \\
&\mathrm{dom}_{\mathrm{act}}(\phi_{\mathrm{d}})/X \cup \mathrm{dom}_{\mathrm{act}}(\phi_{\mathrm{e}})/(X - sy(\phi_{\mathrm{d}})) = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X) \cup (\mathrm{dom}(stp_{\phi_{\mathrm{e}}}) \cap (X - sy(\phi_{\mathrm{d}}))) = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X) \cup (\mathrm{dom}(stp_{\phi_{\mathrm{e}}}) \cap (X - (\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X))) = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X) \cup ((\mathrm{dom}(stp_{\phi_{\mathrm{e}}}) \cap X) - (\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X)) = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X) \cup (\mathrm{dom}(stp_{\phi_{\mathrm{e}}}) \cap X) = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cup \mathrm{dom}(stp_{\phi_{\mathrm{e}}})) \cap X = \\
&\mathrm{dom}(stp_{\phi_{\mathrm{d}}} \cup stp_{\phi_{\mathrm{e}}}) \cap X = \\
&\mathrm{dom}(stp_{\phi_{\mathrm{e}} \triangleleft\!\!+ \phi_{\mathrm{d}}}) \cap X = \\
&\mathrm{dom}(stp_\phi) \cap X
\end{aligned} \tag{7.8}$$

If $\phi \equiv (\phi_{\mathrm{e}} \triangleleft\!\!+_p \phi_{\mathrm{d}})$ then by induction:

$$
\begin{aligned}
&\mathrm{dom}_{\mathrm{act}}(\phi)/X = sy(\phi) = sy(\phi_{\mathrm{d}}) \cup sy(\phi_{\mathrm{e}}) = \\
&\mathrm{dom}_{\mathrm{act}}(\phi_{\mathrm{d}})/(X \cap true(p)) \cup \mathrm{dom}_{\mathrm{act}}(\phi_{\mathrm{e}})/(X \cap false(p)) = \\
&(\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap X \cap true(p)) \cup (\mathrm{dom}(stp_{\phi_{\mathrm{e}}}) \cap X \cap false(p)) = \\
&((\mathrm{dom}(stp_{\phi_{\mathrm{d}}}) \cap true(p)) \cup (\mathrm{dom}(stp_{\phi_{\mathrm{e}}}) \cap false(p))) \cap X = \\
&\mathrm{dom}(stp_{\phi_{\mathrm{e}} \triangleleft_p \phi_{\mathrm{d}}}) \cap X = \\
&\mathrm{dom}(stp_{\phi}) \cap X
\end{aligned}
\tag{7.9}
$$

If $\phi \equiv (p \bullet v_0\colon \phi_{\mathrm{d}_0} \,;\, v_1\colon \phi_{\mathrm{d}_1} \,;\, \dots\, v_n\colon \phi_{\mathrm{d}_n} \,;;\, \phi_{\mathrm{e}})$ then the argument is similar to the previous case. We are done. ☺

Corollary 7.8 Let ϕ be (an occurrence of) a subexpression of a FE Φ. Then the subset of $\mathsf{U} \times \mathsf{I}_{Op}$ on which occurrence ϕ contributes transitions to stp_{Φ} is given by $\mathrm{dom}_{\mathrm{act}}(\phi)$, which is $\mathrm{dom}_{\mathrm{act}}(\phi)/X = sy(\phi)$ with X set to the $in(\phi)$ set obtained by starting from $in(\Phi) = \mathsf{U} \times \mathsf{I}_{Op}$.

Proof. All the non-base cases in Definition 7.6 calculate the *sy*(_) set of the subexpression in question as a union of the *sy*(_) sets of its children. Thus, once $sy(\phi)$ has been obtained on the basis of $in(\Phi) = \mathsf{U} \times \mathsf{I}_{Op}$, it is not constrained further in its ultimate contribution to $sy(\Phi)$. ☺

One evident byproduct of the calculational strategy just described, is that if we leave one or more ingredients as uninstantiated (set-valued) variables in the calculation of a desired *sy*(_) set, and reduce the rest of the calculation as far as possible, we end up with a set transformer expression that shows explicitly how the uninstantiated ingredients contribute to the desired *sy*(_) set.

Example II (Car Locking) Consider operation $UnLock_1 = ((\{(dr\text{-}lck, ps\text{-}lck)\} \lhd (DrUnl_1 \cup PsUnl_1)) \cup UnLockBoth_1)$, and let us calculate $\mathrm{dom}_{\mathrm{act}}(UnLock_1)/\mathsf{U}_1$, leaving $sy(PsUnl_1) \equiv Z$, unistantiated. We thus apply the individual steps from Definition 7.6, getting $\mathrm{dom}_{\mathrm{act}}(UnLock_1)/\mathsf{U}_1 = (\{(dr\text{-}lck, ps\text{-}lck)\} \cap (\{(dr\text{-}lck, ps\text{-}lck), (dr\text{-}unl, ps\text{-}lck)\} \cup Z)) \cup \{(dr\text{-}lck, ps\text{-}lck)\}$, and then we simplify the resulting expression, which gives $\mathrm{dom}_{\mathrm{act}}(UnLock_1)/\mathsf{U}_1 = (\{(dr\text{-}lck, ps\text{-}lck)\} \cup (\{(dr\text{-}lck, ps\text{-}lck)\} \cap Z)) \cup \{(dr\text{-}lck, ps\text{-}lck)\}$. Of course the latter expression is the constant set transformer $(\mathrm{dom}_{\mathrm{act}}(UnLock_1)/\mathsf{U}_1)(Z) = \{(dr\text{-}lck, ps\text{-}lck)\}$, but the penultimate form illustrates how a more general case might look.

8 Evolution of FODs of Operations

Feature engineering consists of manipulating the features that enter into an operation in order to achieve the effects desired. Given the context above, we have the option of describing this activity either via the FOD or via the NF. Depending on the details, it might be more convenient to align an implementation with one or the other of these descriptions. We turn our attention to how such descriptions evolve — amongst other things, the level index k now reappears.

We will consider two kinds of development step for FODs of operations. Neither kind of step will be required to conform to the natural inductive structure inherited from the syntactic form of the feature language, simply because there is no reason to assume that the development activity —driven as it is by many external considerations— will meekly conform to some independently proposed syntactic criteria of this kind. Indeed we have remarked already that model completeness at all stages of

development is a crucial consideration, and there need not be any correlation between that and syntactic considerations.

The first kind of development step simply alters the condition p in a Conditional or Case construct somewhere in the FOD. The second involves the infiltration of one or more new features into the current FOD of the operation. We say ‘infiltration’ to again stress that we are not necessarily working by recursion on the structure of the final FOD, but are contemplating more undisciplined interventions on the FOD.

Despite the unruly nature of such steps as regards the syntactic structure of the FOD, the normal form theorem assures us that all such development steps can be reduced to consideration of partitions of the before-state and input space and the specification of appropriate behaviour on any new pieces generated.

We examine this in more detail below. However before we do so we must recognise that it is seldom the case that all the features we need to deal with intrinsically have the same signature. Often, new features introduced during the development of a system need modified data structures to support the novel functionality, so as the development level index k increases through the various stages of a development, the various U_k, $\mathsf{I}_{Op,k}$, $\mathsf{O}_{Op,k}$ spaces we need cannot be assumed to stay the same. We postpone consideration of this for now.

8.1 Modification of the FOD

We consider the modification of a FOD, on the assumption that the signatures of all FEs entering into the discourse are the same. As previously noted, the modifications are of two kinds: the alteration of a boolean condition, and the infiltration of new features into the FOD.

Suppose we have an FOD $\Phi[\phi]$, which we want to change in the vicinity of ϕ. Here $\Phi[_]$ is a FE context, i.e. a FE with a hole $[_]$, a hole which is filled by an occurrence of a FE term; ϕ in this case.

Maybe we want to introduce a new feature f_c to act alongside ϕ. Unfortunately the domains of ϕ and f_c overlap, so we cannot just move to $\Phi[(\phi \mathbin{\underline{\cup}} f_c)]$ because the sub-expression $(\phi \mathbin{\underline{\cup}} f_c)$ would be ill defined. We could adopt the possibilities $\Phi[(\phi \triangleleft\!+ f_c)]$ or $\Phi[(f_c \triangleleft\!+ \phi)]$, but we might then have to acknowledge that the behaviour of ϕ or f_c alone in the region of overlap, is no longer appropriate in the presence of the other. Instead, we define a new feature f_x to take care of the interaction, making sure that the domain of f_x is precisely $\mathrm{dom}(f_c) \cap \mathrm{dom}(\phi)$. Now we can move from $\Phi[\phi]$ to $\Phi[((\phi \cup f_c) \triangleleft\!+ f_x)]$, avoiding the unnaturalness of the $(\phi \triangleleft\!+ f_c)$ or $(f_c \triangleleft\!+ \phi)$ partial solutions (and also of $\Phi[(\phi \cup f_c)]$, where the nondeterminism between ϕ and f_c in the overlapping region might be regarded as equally inappropriate).

The schema for modifications of FODs that we thus consider is the rewrite of $\Phi[\phi]$ to $\Phi[\gamma(\phi)]$. So we rewrite the occurrence of ϕ using the rule:

$$\kappa \Rightarrow \gamma(\kappa) \qquad (8.1)$$

where κ is a variable (of FE type).[10] The application of the rule matches κ to ϕ, and replaces it by $\gamma(\phi)$ in the context $\Phi[_]$, in the usual way.

Assumption 8.1 All permitted modifications of a FOD are expressible using rewrite rules of the form (8.1).

This spells out in detail what is meant by 'infiltration of new features'. Since the alteration of a boolean condition from p to q in a conditional FOD results in the alteration of $(\phi_e \lhd_p \phi_d)$ to $(\phi_e \lhd_q \phi_d) \equiv ((\phi_e \lhd_{\neg p} \phi_d) \lhd_{p \equiv q} (\phi_e \lhd_p \phi_d))$, we can also encompass these kinds of alteration under a slight extension of the conventions used in (8.1), namely using the rule:

$$\kappa \Rightarrow (\overline{\kappa} \lhd_{p \equiv q} \kappa) \tag{8.2}$$

where $\overline{\kappa}$ negates the condition inside the conditional expression that κ matches (provided κ indeed matches a conditional, undefined otherwise).

The interposition of γ affects the $in(_)$ and $sy(_)$ sets in the vicinity of ϕ. We express this via three operators which transform the original $in(\phi)$ and $sy(\phi)$ sets to sets appropriate to the new situation. Thus we have the operator $In_\gamma(_)$ given by (8.3), acting on the $in(\phi)$ set passed down into ϕ, which calculates how the $in(\phi)$ set changes as a result of ϕ finding itself in a new (more deeply nested) context. We have $Sy_\gamma(_)$ given by (8.4), acting on the $sy(\phi)$ set generated by ϕ, which calculates how the $sy(\phi)$ set changes as a result of ϕ finding itself in the more deeply nested context. Also we have $Sy_{\overline{\gamma}}(_)$ given by (8.5), acting on the $sy(\phi)$ set generated by ϕ, which calculates $sy(\gamma(\phi))$, the $sy(_)$ set of the FE term that now fills the original hole in $\Phi[_]$ after the infiltration. (Note that since $in(_)$ sets are purely inherited, an analogous $In_{\overline{\gamma}}(_)$ operator, that calculated $in(\gamma(\phi))$ in the original context $\Phi[_]$, would just be the identity, so is omitted.) In (8.3)-(8.5) the earlier slash notation is extended to not only associate a $sy(_)$ set with the relevant $in(_)$ set, but to associate both $in(_)$ sets and (slash adorned) $sy(_)$ sets with the contexts in which they are intended to be understood.

$$In_\gamma(in(\phi)/\Phi[_]) \;\equiv\; in(\phi)/\Phi[\gamma[_]] \tag{8.3}$$

$$Sy_\gamma(sy(\phi)/in(\phi)/\Phi[_]) \;\equiv\; sy(\phi)/in(\phi)/\Phi[\gamma[_]] \tag{8.4}$$

$$Sy_{\overline{\gamma}}(sy(\phi)/in(\phi)/\Phi[_]) \;\equiv\; sy(\gamma(\phi))/in(\gamma(\phi))/\Phi[_] \tag{8.5}$$

The equations in Definition 7.6 permit the calculation of In_γ, Sy_γ, $Sy_{\overline{\gamma}}$, from γ in terms of the other quantities of the system. We illustrate this on the specific infiltration rewrite rule $[_] \Rightarrow \gamma[_] \equiv (([_] \cup f_c) \lhd f_x)$ discussed earlier.

Example V (In_γ, Sy_γ, $Sy_{\overline{\gamma}}$, for $((\phi \cup f_c) \lhd f_x)$) We calculate In_γ, Sy_γ, $Sy_{\overline{\gamma}}$, for the FE rule $[_] \Rightarrow \gamma[_] \equiv (([_] \cup f_c) \lhd f_x)$ above. If we take all the stated facts about $\gamma[_]$ into account we can derive the following, where for clarity, we have removed the context (and other) information in the last lines of (8.6) and (8.7), and the last four lines of (8.8):

$In_\gamma(in(\phi)/\Phi[_]) = in(\phi)/\Phi[\gamma[_]] =$ (definition of $\gamma[_]$)
$in(\phi)/\Phi[(([_] \cup f_c) \lhd f_x)] =$ (Definition 7.6.(2)/(5))
$(in(\phi) - \mathrm{dom}(f_x))/\Phi[([_] \cup f_c)] =$ (Definition 7.6.(3))

10. Since κ is a variable, (8.1) is unlike a rule in a conventional term rewrite system (TRS), where left sides which are pure variables are forbidden. Since in a TRS, it is the job of the rules to deliver a desired set of terms (when applied according to some strategy and started from some given starting conditions), allowing a variable left side would allow replacement of *anything* by the right side, which is too permissive. In our context, the purpose of the rule is to formalise changes selected by the (human) designer, so such freedom in the application of the rule is appropriate, since the use of the rule is controlled from outside the formal system.

$$
\begin{aligned}
&(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] = \text{(ZF)}\\
&in(\phi)/\Phi[_] - \mathrm{dom}(f_{\mathrm{x}}) =\\
&in(\phi) - \mathrm{dom}(f_{\mathrm{x}}) && (8.6)
\end{aligned}
$$

$$
\begin{aligned}
&Sy_{\gamma}(sy(\phi)/in(\phi)/\Phi[_]) = sy(\phi)/in(\phi)/\Phi[\gamma[_]] = \text{(definition of } \gamma[_])\\
&sy(\phi)/in(\phi)/\Phi[(([_] \cup f_{\mathrm{c}}) \mathbin{⩤} f_{\mathrm{x}})] = (8.6)\\
&sy(\phi)/(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] = \text{(Theorem 7.7, ZF)}\\
&(sy(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] =\\
&sy(\phi) - \mathrm{dom}(f_{\mathrm{x}}) && (8.7)
\end{aligned}
$$

$$
\begin{aligned}
&Sy_{\overline{\gamma}}(sy(\phi)/in(\phi)/\Phi[_]) = sy(\gamma(\phi))/in(\gamma(\phi))/\Phi[_] = (in(_) \text{ depends only on } \Phi[_])\\
&sy(\gamma(\phi))/in(\phi)/\Phi[_] = \text{(definition of } \gamma[_])\\
&sy(((\phi \cup f_{\mathrm{c}}) \mathbin{⩤} f_{\mathrm{x}}))/in(\phi)/\Phi[_] = \text{(Definition 7.6.(5))}\\
&sy(f_{\mathrm{x}})/in(\phi)/\Phi[_] \cup sy(\phi \cup f_{\mathrm{c}})/(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] = \text{(Definition 7.6.(2)/(3))}\\
&(\mathrm{dom}(f_{\mathrm{x}}) \cap in(\phi))/in(\phi)/\Phi[_] \cup sy(\phi)/(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] \cup\\
&\qquad sy(f_{\mathrm{c}})/(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] = \text{(Theorem 7.7, Definition 7.6.(2), ZF)}\\
&(\mathrm{dom}(f_{\mathrm{x}}) \cap in(\phi))/in(\phi)/\Phi[_] \cup\\
&\qquad ((sy(\phi) - \mathrm{dom}(f_{\mathrm{x}})) \cap (in(\phi) - \mathrm{dom}(f_{\mathrm{x}})))/(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] \cup\\
&\qquad ((\mathrm{dom}(f_{\mathrm{c}}) - \mathrm{dom}(f_{\mathrm{x}})) \cap (in(\phi) - \mathrm{dom}(f_{\mathrm{x}})))/(in(\phi) - \mathrm{dom}(f_{\mathrm{x}}))/\Phi[_] = \text{(ZF)}\\
&(\mathrm{dom}(f_{\mathrm{x}}) \cap in(\phi)) \cup ((sy(\phi) - \mathrm{dom}(f_{\mathrm{x}})) \cap in(\phi)) \cup ((\mathrm{dom}(f_{\mathrm{c}}) - \mathrm{dom}(f_{\mathrm{x}})) \cap in(\phi)) =\\
&(sy(\phi) \cap in(\phi)) \cup (\mathrm{dom}(f_{\mathrm{c}}) \cap in(\phi)) =\\
&(sy(\phi) \cup \mathrm{dom}(f_{\mathrm{c}})) \cap in(\phi) =\\
&sy(\phi) \cup (\mathrm{dom}(f_{\mathrm{c}}) \cap in(\phi)) && (8.8)
\end{aligned}
$$

Thus we see from (8.6) that the $in(_)$ set passed down into ϕ when the context $\gamma[_]$ is infiltrated into $\Phi[_]$ gets reduced (with respect to what it originally was, i.e. $in(\phi)$) by the removal of any elements in common with $\mathrm{dom}(f_{\mathrm{x}})$ — given the form of $\gamma[_]$, this is as we would expect. The calculation of the corresponding $sy(\phi)$ set is similarly affected — since $sy(\phi)$ is calculated from the internal structure of ϕ, which remains unchanged (the calculation being modulo the $in(_)$ set passed down), Theorem 7.7 allows us to conclude that a similar reduction applies as is derived in (8.7). Finally, the $sy(_)$ set of the FE $\gamma(\phi)$ depends more decisively on the structure of $\gamma[_]$. Since the $in(_)$ set passed down into $\gamma(\phi)$ is identical to the original $in(\phi)$ set (as noted earlier), we just use Definition 7.6 to unravel the effect of $\gamma[_]$ till we have the result in terms of the original $sy(\phi)$, $in(\phi)$, and constants. The last few equalities in (8.8) follow from the conditions that we know hold in this situation, namely that $sy(\phi) \subseteq in(\phi)$ and that $\mathrm{dom}(f_{\mathrm{x}}) = \mathrm{dom}(f_{\mathrm{c}}) \cap \mathrm{dom}(\phi)$.

On this basis we can state:

$$In_{\gamma}(_) = (_) - \mathrm{dom}(f_{\mathrm{x}}) \tag{8.9}$$

$$Sy_{\gamma}(_) = (_) - \mathrm{dom}(f_{\mathrm{x}}) \tag{8.10}$$

$$Sy_{\overline{\gamma}}(_) = (_) \cup (\mathrm{dom}(f_{\mathrm{c}}) \cap in(\phi)) \tag{8.11}$$

Note that just because we know (from (8.11)) that $sy(\phi)$ is a subset of $Sy_{\overline{\gamma}}(sy(\phi))$, does not of course imply that the before-FE and after-FE of the infiltration behave the same way on the $sy(\phi)$ part of the domain. For example, the after-FE also hosts f_{x} on part of $sy(\phi)$. Similar remarks apply more generally.

8.2 Addition and Removal of Data Spaces

Now we consider changing data spaces. We tackle this in isolation, unencumbered by any thought of changing the FOD.

Suppose that feature f_{d}, which we want to introduce in the passage from level k to level $k+1$, just requires additional supporting data u_{d} in the state, drawn from a space of values for u_{d}, U_{d} say, this being over and above the state already available at level k. Then U_{d} will be present in U_{k+1} and not be present in U_k. As such, U_{d} will be present in U_{k+1} as a new cartesian factor alongside all the other state components, since U_{d} will be independent of any other data types used in the system.

Given this, we must now consider how a feature f_{c} which is present at level k and persists into level $k+1$, and so is well defined on states in U_k, is to be understood on a larger state space including U_{d}. The answer is easy. In common with programming practice, in which an update of a variable leaves all other variables unaffected, we understand the relation representing f_{c} in the larger state space including U_{d}, to be the relation on U_k extended by the identity on irrelevant factors such as U_{d}. Specifically, if u -($i, f_{\mathrm{c},k}, o$)-› u' is a typical transition for individual feature f_{c} at level k, then at level $k+1$, the transition u -($i, f_{\mathrm{c},k}, o$)-› u' will be represented by a collection of transitions (u, u_{d}) -($i, f_{\mathrm{c},k+1}, o$)-› (u', u_{d}), one for every element $u_{\mathrm{d}} \in \mathsf{U}_{\mathrm{d}}$, where for simplicity we have assumed that the addition of the space U_{d} is the only alteration in the state spaces needed in the passage from level k to level $k+1$. The same idea works for as many new state components as we might need to introduce in the passage from level k to level $k+1$.

Provided we restrict to this way of modifying the state spaces between levels, in general, U_k will be a cartesian product of individual types $\mathsf{U}_{\mathrm{a},k} \times \mathsf{U}_{\mathrm{b},k} \times \ldots \times \mathsf{U}_{\mathrm{d},k}$ some of which are present because a specific feature demands them, others being common to the activity of several or all of the features at level k. Similarly for level $k+1$, where U_{k+1} will be $\mathsf{U}_{\mathrm{a}',k+1} \times \mathsf{U}_{\mathrm{b}',k+1} \times \ldots \times \mathsf{U}_{\mathrm{e}',k+1}$. Regarding the relationship between the state spaces at levels k and $k+1$, some of the $\mathsf{U}_{\mathrm{b},k}$ can be identified with some of the $\mathsf{U}_{\mathrm{b}',k+1}$. This will be because they are data types ‘used in the same way’ by features that are present at both levels — typically they will be the types of the same variables in a syntactic description of the common features.[11] The remainder of the $\mathsf{U}_{\mathrm{c},k} \ldots \mathsf{U}_{\mathrm{d},k}$ will be present only at level k, and the remainder of the $\mathsf{U}_{\mathrm{c}',k+1} \ldots \mathsf{U}_{\mathrm{e}',k+1}$ will be present only at level $k+1$; both effects arise because those spaces concern features present exclusively at one level but not the other.[12]

11. We must emphasise that in a purely semantic framework like ours, strictly speaking, such a correspondence remains outside the formalism without recourse to some such syntactic description of features, explaining why ‘used in the same way’ appears in quotes.

12. Having a $\mathsf{U}_{c,k}$ not present at level $k+1$ happens when a feature is removed (by being completely overridden) at level $k+1$. Why introduce something earlier only to override it later? In a monolithic development it makes no sense. But in a long lived, multi-staged development process, when there might be millions of units of an earlier design out in the field, it will be impossible to pretend that a feature installed earlier can simply be erased from the development. Telephony is the obvious example. If a feature is completely overridden in this manner it need not be implemented, and so the data that it would otherwise need, i.e. $\mathsf{U}_{c,k}$, can be removed from the state space.

Proposition 8.2 When data spaces U_k and U_{k+1} differ only by the addition or removal of cartesian factors, the relationship between U_k and U_{k+1} is a total surjective regular relation $\rho_{\mathsf{U},k,k+1}$.

Proof. Let a typical value in U_k be $u_k = (u_{a,k}, \dots, u_{c,k}, u_{d_1,k}, \dots, u_{d_n,k})$ and a typical value in U_{k+1} be $u_{k+1} = (u_{d'_1,k+1}, \dots, u_{d'_n,k+1}, u_{e',k+1}, \dots, u_{g',k+1})$, where in the passage from level k to level $k+1$, subspace $\mathsf{U}_{a,k} \times \dots \times \mathsf{U}_{c,k}$ is removed from U_k, subspace $\mathsf{U}_{e',k+1} \times \dots \times \mathsf{U}_{g',k+1}$ is added to form U_{k+1}, and where the common subspaces, n of them, $\mathsf{U}_{d_1,k} = \mathsf{U}_{d'_1,k+1}, \dots, \mathsf{U}_{d_n,k} = \mathsf{U}_{d'_n,k+1}$ are identified as indicated. Then:

$$\rho_{\mathsf{U},k,k+1}(u_k, u_{k+1}) \equiv (u_{d_1,k} = u_{d'_1,k+1} \wedge \dots \wedge u_{d_n,k} = u_{d'_n,k+1}) \tag{8.12}$$

We see that $\rho_{\mathsf{U},k,k+1}$ is the composition of the projection that discards $(u_{a,k}, \dots, u_{c,k})$ from u_k followed by the inverse projection that adds $(u_{e',k+1}, \dots, u_{g',k+1})$ to get u_{k+1}. Since projections are functions, this displays $\rho_{\mathsf{U},k,k+1}$ in difunctional form. ☺

Example I (Light Switch) Assuming that at level 0 the only states are $\{off, on\} = \mathsf{U}_0$, suppose we wished to introduce a dimmer feature into the switch, that allowed the light it controlled to exhibit varying degrees of brightness. At level 1 we adjoin the states $\{1 \dots n\}$ to the level 0 state space, getting $\mathsf{U}_1 = \{off, on\} \times \{1 \dots n\}$. The (on, b) states now represent varying degrees of brightness, while the (off, b) states all represent an extinguished light.[13] Clearly the natural relationship $\rho_{\mathsf{U},0,1}$ between the level 0 and level 1 state spaces is an inverse projection from level 0 to level 1.

Regular relations between the various levels that arise in this way, possess properties regarding sequential composition not shared by arbitrary chains of sequentially composable relations (that happen to be regular).

Proposition 8.3 When data spaces U_k, U_{k+1} and U_{k+2} differ only by the addition or removal of cartesian factors, the sequential composition $\rho_{\mathsf{U},k,k+1} \mathbin{⨟} \rho_{\mathsf{U},k+1,k+2}$ of the regular relations $\rho_{\mathsf{U},k,k+1}$ and $\rho_{\mathsf{U},k+1,k+2}$ from U_k to U_{k+1}, and from U_{k+1} to U_{k+2}, is a total surjective regular relation $\rho_{\mathsf{U},k,k+2}$. Consequently $\rho_{\mathsf{U},k,k+1}$ and $\rho_{\mathsf{U},k+1,k+2}$ are RS compatible.

Proof. For the relationship between levels k and $k+1$, suppose U_k and U_{k+1} are related via $\rho_{\mathsf{U},k,k+1}$ as in (8.12). For the relationship between levels $k+1$ and $k+2$, let us relabel the space U_{k+1} by letting a typical value in U_{k+1} be $u_{k+1} = (u_{a',k+1}, \dots, u_{c',k+1}, u_{e'_1,k+1}, \dots, u_{e'_m,k+1})$, and let a typcal value in U_{k+2} be $u_{k+2} = (u_{e''_1,k+2}, \dots, u_{e''_m,k+2}, u_{f'',k+2}, \dots, u_{h'',k+2})$, where subspace $\mathsf{U}_{a',k+1} \times \dots \times \mathsf{U}_{c',k+1}$ is removed from U_{k+1} in the passage from level $k+1$ to level $k+2$, and subspace $\mathsf{U}_{f'',k+2} \times \dots \times \mathsf{U}_{h'',k+2}$ is added. Suppose $\mathsf{U}_{e'_1,k+1} = \mathsf{U}_{e''_1,k+2}, \dots, \mathsf{U}_{e'_m,k+1} = \mathsf{U}_{e''_m,k+2}$ are the common types in the natural relationship between levels $k+1$ and $k+2$, m of them, identified as indicated. Then the counterpart of (8.12) for levels $k+1$ and $k+2$ is:

$$\rho_{\mathsf{U},k+1,k+2}(u_{k+1}, u_{k+2}) \equiv (u_{e'_1,k+1} = u_{e''_1,k+2} \wedge \dots \wedge u_{e'_m,k+1} = u_{e''_m,k+2}) \tag{8.13}$$

Let θ_{k+1} be the relabelling function so that θ_{k+1} captures the bijection between the labels of the $(u_{d'_1,k+1}, \dots, u_{d'_n,k+1}, u_{e',k+1}, \dots, u_{g',k+1})$ decomposition of u_{k+1} and the labels of the $(u_{a',k+1}, \dots, u_{c',k+1}, u_{e'_1,k+1}, \dots, u_{e'_m,k+1})$ decomposition. Then the sequential composition of $\rho_{\mathsf{U},k,k+1}$ and $\rho_{\mathsf{U},k+1,k+2}$ is defined by:

13. We will overcome the unnatural aspects of this representation in Section 8.3.

$$
\begin{aligned}
&(\rho_{\mathsf{U},k,k+1} \mathbin{⨾} \rho_{\mathsf{U},k+1,k+2})(u_k, u_{k+2}) \equiv \\
&\quad ((u_{\mathrm{d}_{j1},k} = u_{\mathrm{d}'_{j1},k+1} \wedge \theta_{k+1}(\mathrm{d}'_{j1,k}) = \mathrm{e}'_{j1,k} \wedge u_{\mathrm{e}'_{j1},k+1} = u_{\mathrm{e}''_{j1},k+2}) \wedge \\
&\quad (u_{\mathrm{d}_{j2},k} = u_{\mathrm{d}'_{j2},k+1} \wedge \theta_{k+1}(\mathrm{d}'_{j2,k}) = \mathrm{e}'_{j2,k} \wedge u_{\mathrm{e}'_{j2},k+1} = u_{\mathrm{e}''_{j2},k+2}) \wedge \\
&\quad\quad \dots \qquad \dots \qquad \dots \\
&\quad (u_{\mathrm{d}_{jl},k} = u_{\mathrm{d}'_{jl},k+1} \wedge \theta_{k+1}(\mathrm{d}'_{jl,k}) = \mathrm{e}'_{jl,k} \wedge u_{\mathrm{e}'_{jl},k+1} = u_{\mathrm{e}''_{jl},k+2})) \qquad (8.14)
\end{aligned}
$$

where $\mathrm{d}'_{j1,k}$, $\mathrm{d}'_{j2,k}$, … , $\mathrm{d}'_{jl,k}$ enumerates all the types at level $k+1$ that are simultaneously common with types at level k, and (under a different name) are common with types at level $k+2$. Since this is a composition of a projection with an inverse projection just as before, it is a difunctional presentation of $\rho_{\mathsf{U},k,k+1} \mathbin{⨾} \rho_{\mathsf{U},k+1,k+2}$ and thus is total, surjective and regular. ☺

We will assume that mechanisms similar to the above hold for the input and output spaces $\mathsf{I}_{Op,k}$ and $\mathsf{O}_{Op,k}$, whose incarnations at various levels are related by total surjective regular relations $\rho_{\mathsf{I}_{Op},k,k+1}$ and $\rho_{\mathsf{O}_{Op},k,k+1}$ constructed by discarding some component types and incorporating new ones. In particular their compositions are also total, surjective and regular.[14]

One pathological situation that we must mention is when there are no common types at all between two levels which are either adjacent, or become related as a result of one or more compositions. In this case the ρ relation becomes universal (an empty conjunction). We will assume that the developments we are considering display enough coherence that this situation does not arise. Since in practical feature engineering situations, adding new features is far more prevalent than removing them completely, this is a reasonable assumption.

8.3 Change of Data Representation

The addition and removal of subspaces is a flexible and convenient method of manipulating the expansion and contraction of the state space as features come and go, and moreover, as we have seen, it enjoys very useful properties as regards regularity of the relations that arise between the various spaces. Neverthless it can sometimes lead to state spaces that look somewhat unnatural from the requirements perspective.

Example I (Light Switch) We saw an example of this above, where the level 0 state *off* was related to all the level 1 states (*off*, b), for $b \in \{1 \dots n\}$.[15] A further aspect of

14. There is a subtlety with input and output spaces that is largely hidden in the case of the system state. If the output space (say) needs to acquire a new cartesian factor, going from J to $\mathsf{J} \times \mathsf{K}$, because of the demands of some new feature f_{new} in the operation, then even when f_{new} is not being used, a value from K must be output for all steps of the operation, even if it has to be a default value, because outputs are now pairs. Although not inconceivable, this is quite a drastic redesign of the operation as a whole. What is more likely in practice is that f_{new} will output some hitherto unused values from J to accomplish its task (making the construction more like a sum than a product). Similar remarks apply to inputs. We show how this can be dealt with in the next section. State is different because it persists from step to step, so state components of no interest to the feature currently being invoked remain undisturbed, and do not impact on the current step. Only at system initialisation time do we see an effect as for inputs and outputs, when values have to be supplied simultaneously for all state components.

15. In a more sophisticated light switch design, the different n values in the extinguished state could retain the last illumination level so that it might be re-established when the light is next turned on. But we have a simpler model in mind.

the same example was that the *on* component of the various (*on*, *b*) states was not really playing a useful role. We can remedy both defects by changing the data representation, introducing at level 2, a state space $\mathsf{U}_2 = \{0 \dots n\}$. Now 0 can encode all the (*off*, *b*) states, and for the remainder, state (*on*, *b*) will correspond to state *b*, for all $b \in \{1 \dots n\}$. Clearly the natural relationship $\rho_{\mathsf{U},1,2}$ between level 1 and level 2 that we have just described is a projection from level 1 to level 2. Equally clearly, the level 1 equivalence classes for the earlier $\rho_{\mathsf{U},0,1}$ are the same as the level 1 equivalence classes for $\rho_{\mathsf{U},1,2}$. So the two regular relations $\rho_{\mathsf{U},0,1}$ and $\rho_{\mathsf{U},1,2}$ are RS compatible, and their sequential composition $\rho_{\mathsf{U},0,2} = \rho_{\mathsf{U},0,1} \mathbin{⨾} \rho_{\mathsf{U},1,2}$ is thus regular.

The ρ_{U} relations that we have discussed typically arise as the retrieve relations of retrenchments and refinements between (full descriptions of) the systems at the various levels concerned. A simple view of the development process thus characterises a relation like $\rho_{\mathsf{U},0,1}$ as the retrieve relation of a retrenchment that captures non-trivial feature manipulation, while a relation like $\rho_{\mathsf{U},1,2}$ is characterised as the retrieve relation of a refinement purely concerned with a change of data representation. In this paper we will persist with this simple distinction between retrenchments being for feature manipulation and refinements being for change of data representation —it makes matters more straightforward technically— while recognising that the composition theory of retrenchments and refinements in [Banach et al. (2008)] enables us to extend the straightforward composition of retrieve relations $\rho_{\mathsf{U},0,2} = \rho_{\mathsf{U},0,1} \mathbin{⨾} \rho_{\mathsf{U},1,2}$ into a composition of the requisite retrenchment and refinement in their entirety, yielding a (more complicated) retrenchment.

Evidently, in practice, in a simple example such as the above, we would write down the relation $\rho_{\mathsf{U},0,2}$ in one step, rather than breaking it up into substeps, but the route illustrated is enlightening as regards regularity. Specifically, whereas we showed that retrieve relations made by addition and removal of subspaces are naturally regular and RS compatible, it is (on the contrary) quite easy to write down arbitrary change-of-data-representation retrieve relations that do not enjoy these properties. Yet when examined in the context of a specific case, we found that the relation $\rho_{\mathsf{U},1,2}$ which we needed *was* regular and RS compatible with its predecessor. This is typical of the behaviour examined at length in [Banach (1995)], which argued, via many examples, that regularity is to be expected (and encouraged) in many aspects of practical system design. Accordingly, we axiomatise this as an assumption in the present context.

Assumption 8.4 All retrieve relations between state spaces are regular and RS compatible with their neighbours in the development process. Similar remarks apply for input and output relations.

8.4 All Together

Taking account of all of the above, the passage of a FOD from level k to level $k+1$ can now be conceptually subdivided into the following four steps.

1. Firstly, we add in any types newly required at level $k+1$ to the level k FOD in the manner described in Section 8.2.

2. Secondly, we modify the FOD to incorporate any new features and conditional alterations in the manner described in Section 8.1 (given that the state and I/O spaces are now adequate to accommodate them).

3. Thirdly, we project out any subspaces no longer needed due to their individual features having empty active domains as a consequence of their being completely overridden, using another application of the techniques of Section 8.2.
4. Fourthly, we can apply any needed change of data representation, using relations between spaces which are regular and RS compatible with their neighbours, in the manner described in Section 8.3.

When we have suitably formalised the whole FE process using retrenchments (including any needed refinements) below, the above represents the data space aspects of the passage from level k to level $k+1$ as a path round the square in Fig. 1.(a), in which the first three steps compose to give a retrenchment across the top, from A to B, and the fourth step is a refinement from B to D. In practice of course, any of these four steps can be amalgamated by composition.

Example II (Car Locking) Let us suppose that at level 2, the design has arrived at a more deterministic form, so that $UnLock_2$ always unlocks both doors. We summarise the design as follows. The state space is $\mathsf{U}_2 = \{dr\text{-}lck, dr\text{-}unl\} \times \{ps\text{-}lck, ps\text{-}unl\}$. Using an obvious abbreviation, the $Lock_2$ operation is given by $(\mathsf{U}_2 - (dr\text{-}lck, ps\text{-}lck))$ -($Lock_2$)-› $(dr\text{-}lck, ps\text{-}lck)$, while the $UnLock_2$ operation is given by the single transition $(dr\text{-}lck, ps\text{-}lck)$ -($UnLock_2$)-› $(dr\text{-}unl, ps\text{-}unl)$. Under the obvious equality retrieve relation, the passage from $Lock_1$ and $UnLock_1$ to $Lock_2$ and $UnLock_2$ constitutes a refinement according to the definition in Section 5.2 (provided we deal with all other operations analogously).

During the next level of development, it is decided that car locking and unlocking may also be done via short-range wireless, with the user pressing buttons on a key fob. To prevent situations in which the family dog, left in charge of an unlocked car, inadvertently activates the $Lock_3$ button of the key fob which itself has been inadvertently left inside the car, with the result that the family is not only locked out of the car but also deprived of the key fob, the system has to be aware of whether the fob is inside or outside the car, and must only permit the $Lock_3$ button to fully lock the car when there is no fob inside. To model this, we enhance the state space to indicate whether the fob is inside the car or not. The state space now becomes $\mathsf{U}_3 = \{dr\text{-}lck, dr\text{-}unl\} \times \{ps\text{-}lck, ps\text{-}unl\} \times \{fob\text{-}in, fob\text{-}out\}$. The $Lock_3$ operation has the transitions $(*, *, fob\text{-}out)$ -($Lock_3$)-› $(dr\text{-}lck, ps\text{-}lck, fob\text{-}out)$ which cover the usual behaviour when there is no fob in the car,[16] and to cover attempts to lock the car while the fob is inside we have $(*, *, fob\text{-}in)$ -($Lock_3$)-› $(dr\text{-}unl, ps\text{-}lck, fob\text{-}in)$ which ensure that at least the driver door remains open. The $UnLock_3$ operation has the transitions $(*, *, *)$ -($UnLock_3$)-› $(dr\text{-}unl, ps\text{-}unl, *)$. We also have the level 3 versions of the mechanical operations for individual doors: $(dr\text{-}unl, *, *)$ -($DrLck_3$)-› $(dr\text{-}lck, *, *)$ and $(dr\text{-}lck, *, *)$ -($DrUnl_3$)-› $(dr\text{-}unl, *, *)$ for the driver; and $(*, ps\text{-}unl, *)$ -($PsLck_3$)-› $(*, ps\text{-}lck, *)$ and $(*, ps\text{-}lck, *)$ -($PsUnl_3$)-› $(*, ps\text{-}unl, *)$ for the passenger. Of these, $DrLck_3$ applied after $Lock_3$, enables all the doors to be locked from the inside if that is what is required.

We can capture the development from the $Lock_2$ to $Lock_3$ using our feature calculus as follows. Firstly we add the fob type $\{fob\text{-}in, fob\text{-}out\}$ to the level 2 state space.

16. Note that there is a now a (skip) transition from the $(dr\text{-}lck, ps\text{-}lck, fob\text{-}out)$ state since there is nothing to prevent one from pressing the $Lock_3$ button, even when the car is locked. Consequently $(dr\text{-}lck, ps\text{-}lck, fob\text{-}out) \in \mathrm{ctxpre}(Lock_3)$.

Next we let $Lock_{2/3}$ be the natural translation of $Lock_2$ to level 3 variables (i.e. the level 3 door states are manipulated in the obvious way, and the operation skips on the fob states). Next, let $LockSkip_3$ be the feature that captures what happens when the $Lock_3$ button is pressed when the car is already locked; it has the transitions (*dr-lck*, *ps-lck*, ∗) -($LockSkip_3$)-› (*dr-lck*, *ps-lck*, ∗). Also, let $FobAccessible_3$ be the feature that prevents the fob from being inadvertently locked in the car, having the transitions (∗, ∗, *fob-in*) -($FobAccessible_3$)-› (*dr-unl*, *ps-lck*, *fob-in*). Then we can write $Lock_3$ as:

$$Lock_3 = ((Lock_{2/3} \uplus LockSkip_3) \lessdot FobAccessible_3) \tag{8.15}$$

which is of a form practically identical to an application of our prototypical feature rewrite rule above.

9 Feature Evolution via Retenchment

Having described how we can move from level to level both in terms of how FODs alter and how we can describe the relationship between the relevant state and other spaces, we now turn our attention to describing the retrenchments that capture this process.

9.1 The Retrieve, Within, and Output Relations, and Regularity

We define the retrieve, within, and output relations between successive layers thus:

$$G_{k+1}(u_k, u_{k+1}) \equiv \rho_{\mathsf{U},k,k+1}(u_k, u_{k+1}) \tag{9.1}$$

$$P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1}) \equiv \rho_{\mathsf{I}_{Op},k,k+1}(i_k, i_{k+1}) \tag{9.2}$$

$$O_{Op,k+1}(o_k, o_{k+1}; u'_k, u'_{k+1}, i_k, i_{k+1}, u_k, u_{k+1}) \equiv \rho_{\mathsf{O}_{Op},k,k+1}(o_k, o_{k+1}) \tag{9.3}$$

In (9.2) there is an implicit cartesian product with a universal relation from U_k to U_{k+1} on the right hand side, and in (9.3) there is on the right hand side an implicit cartesian product with a universal relation from the values of the variables u'_k, i_k, u_k, to the values of the variables u'_{k+1}, i_{k+1}, u_{k+1}. Recalling that we argued above that all these ρ relations could be taken to be regular and RS compatible, since the cartesian product of regular relations is obviously regular, we conclude that the within and output relations will also be total surjective regular relations whose compositions are total, surjective and regular.

Now, for each relevant Op, we form $G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})$ by forming the cartesian product of $G_{k+1}(u_k, u_{k+1})$ with a universal relation from the inputs at level k to those at level $k+1$, and taking the intersection of the resulting relation with $P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})$. Since the intersection of regular relations is regular, we see that the relations $G_{k+1} \wedge P_{Op,k+1}$ are total surjective regular relations whose compositions are total, surjective and regular. Similarly we see that the relations $G_{k+1}(u'_k, u'_{k+1}) \wedge O_{Op,k+1}(o_k, o_{k+1}; u'_k, u'_{k+1}, i_k, i_{k+1}, u_k, u_{k+1})$ also have this property.

9.2 Assumptions for the Concedes Relation

We have almost shown that the retrenchments we are developing have regular data. To say something about the concedes relations, we must first discuss the transition relations for Op_k and Op_{k+1}. We do so in the context of a number of assumptions.

Assumption 9.1 Model Completeness and Defaults. Above, we described model completeness of an operation Op, noting that it concerned a property, ctxpre(Op), which captured all the situations of practical interest for Op. We demand that this interacts smoothly with the $G \wedge P_{Op}$ just constructed, which we express as follows:

$$\mathrm{ctxpre}(Op_k)(u_k, i_k) \Rightarrow (\exists\, u_{k+1}, i_{k+1} \bullet \mathrm{ctxpre}(Op_{k+1})(u_{k+1}, i_{k+1}) \wedge G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})) \tag{9.4}$$

$$\mathrm{ctxpre}(Op_{k+1})(u_{k+1}, i_{k+1}) \Rightarrow (\exists\, u_k, i_k \bullet \mathrm{ctxpre}(Op_k)(u_k, i_k) \wedge G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})) \tag{9.5}$$

Lemma 9.2 For every $(u_k, i_k) \in$ ctxpre(Op_k), there exists (u_{k+1}, i_{k+1}) such that $P^{\mathrm{Def}}{}_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})$ holds, where $P^{\mathrm{Def}}{}_{Op,k+1}$ is given by (5.8) using the $G_{k+1} \wedge P_{Op,k+1}$ just constructed. Similarly with k and $k+1$ interchanged.

Proof. Suppose $(u_k, i_k) \in$ ctxpre(Op_k). Then $(u_k, i_k) \in$ dom(Op_k) by model completeness (3.1), thus also satisfying the healthiness condition (5.7). Similarly for the (u_{k+1}, i_{k+1}) asserted by (9.4). Adding these two domain conditions to (9.4) quickly derives $P^{\mathrm{Def}}{}_{Op,k+1}$ as required. The argument starting from (u_{k+1}, i_{k+1}) is similar. ☺

In the spirit of aiming to support software engineering intuitions, we said that we would assume that our models are model complete at every level k. Thus if some (u_k, i_k) is a point from which it is necessary for a level k transition to emerge, then its level $k+1$ counterpart (u_{k+1}, i_{k+1}), obtained via $G_{k+1} \wedge P_{Op,k+1}$, must be a point from which it is appropriate for a level $k+1$ transition to emerge, and vice versa. Lemma 9.2 shows that under simple conditions this corresponds nicely to the statement that the system has no capabilities that lie outside the remit of the default within relations $P^{\mathrm{Def}}{}_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})$.

Assumption 9.3 Interfeature Independence. By this we mean that distinct features do not encroach on each other's work. There is no point in designing a new feature to duplicate the work of an old one,[17] so if f_{c} and f_{d} are distinct features present in levels k and $k+j$, $(j \geq 0)$ then we will always have the negation of the analogue of $\overline{G}_{Op,(k+1,k+j)}$ for them (where $\overline{G}_{Op,(k+1,k+j)}$ captures refining simulation, as expressed earlier in (5.13)):

$$\neg\, (G_{(k+1,k+j)}(u_k, u_{k+j}) \wedge P_{Op,(k+1,k+j)}(i_k, i_{k+j}, u_k, u_{k+j}) \wedge stp_{f_{\mathrm{c}}}(u_k, i_k, u'_k, o_k) \wedge stp_{f_{\mathrm{d}}}(u_{k+j}, i_{k+j}, u'_{k+j}, o_{k+j}) \wedge G_{(k+1,k+j)}(u'_k, u'_{k+j}) \wedge O_{Op,(k+1,k+j)}(o_k, o_{k+j}; u'_k, u'_{k+j}, i_k, i_{k+j}, u_k, u_{k+j})) \tag{9.6}$$

In (9.6), $G_{(k+1,k+j)}$ is the composition $G_{k+1} \fatsemi G_{k+2} \fatsemi \ldots \fatsemi G_{k+j}$, defaulting to the identity if $j = 0$, and to G_{k+1} if $j = 1$. Similarly for the other relations.

Assumption 9.4 Interfeature Determinism. By this we mean that for any $(u_k, i_k) \in$ ctxpre(Op_k), there is exectly one feature f_{d} of the FOD of operation Op_k such that every transition u_k -($i_k, f_{\mathrm{e},k}, o_k$)-› u'_k, emerging from (u_k, i_k) is a transition of f_{d}, i.e. $f_{\mathrm{e}} = f_{\mathrm{d}}$. In other words, the active domains of distinct elements of the FOD of Op_k should not intersect.

17. This is perhaps a bit hasty. It may well be that from time to time during the life of a long-lived product, we want to 'clean up' the development by removing (i.e. overriding) some tired old features and replacing them with shiny new ones that (at least some of the time) do the same job as the old ones. But we will ignore this possibility here for simplicity.

We justify this by claiming that users have a right to expect predictable behaviour for a given starting condition, and we assume such nuggets of predictable behaviour are encapsulated within individual features. (That is not to say that individual features cannot themselves be nondeterministic when there are justifiable requirements reasons for them being so (for example seat allocation on budget airline flights) — but that is a different issue.)

Interfeature determinism excludes certain FODs, primarily ones containing 'naked unions' such as $(f_c \cup f_d)$ where the domains of f_c and f_d overlap and the union is not masked by some form of overriding. Not all FODs containing overlapping unions are disbarred. Referring to our earlier example of infiltration via a rewrite rule $\kappa \Rightarrow \gamma(\kappa)$, where γ was given by $\gamma[_] \equiv (([_] \cup f_c) \lessdot f_x)$ with the hole [_] being filled by a feature f_d with $\varnothing \neq \mathrm{dom}(f_c) \cap \mathrm{dom}(f_d) = \mathrm{dom}(f_x)$, we see that despite the union, it is interfeature deterministic because of the override on the overlap. Such FODs can be rewritten to remove the unions, eg. $((f_d \lessdot f_c) \lessdot f_x)$. Although equivalent to the former expression this could be less appropriate as regards eloquence in expressing requirements, as we noted before.

Let us reflect a little on these assumptions. While model completeness can easily be understood as an uncontroversial requirement of the development methodology, the status of interfeature independence and interfeature determinism is more open to question. For example, one can certainly imagine designing features that partially duplicate each other's work, as noted already. But then one could focus the analysis lower down, by introducing a notion of subfeature, such that each feature would be a union asserted disjoint of a family of subfeatures, and such that individual subfeatures capture the unique pieces of functionality shared among more than one parent feature. One could then consider the legitimacy of the idea of requiring intersubfeature independence. Similarly one can imagine designing operations requiring features that partially overlap in a common subdomain where both are active. In such a case one could again refocus the analysis on subfeatures that capture the common behaviour and consider the legitimacy of intersubfeature determinism.

In such a scenario the crucial issue amounts to 'What ought a (sub)feature to be?', which amounts to introducing an extra layer of structure analogous to the structuring we were already working with. It is related to the naturalness or otherwise of different subdivisions of the functionality offered by an operation. Such structural variations add complexity to the formalism for describing operations via features —complexity which may be justifiable in an application context— without fundamentally improving the theoretical properties of this approach. So, in order to limit the technical complexity of our retrenchment framework, we remain with the assumptions as stated.

9.3 The Concedes Relation and Regularity

We now consider a development step from level k to level $k+1$, given either by modifying some condition in the FOD $\Phi_k[\phi]$ of Op_k, or by applying a rewrite rule like $\kappa \Rightarrow \gamma(\kappa)$ in (8.1) to some subexpression ϕ of $\Phi_k[\phi]$. The fact that neither option need act at the root of the parse tree of $\Phi_k[\phi]$ by an application of a FE constructor, blocks the analysis of what can happen via the structural induction route, at least in any straightforward way. Fortunately, the NF theorem allied with the assumptions above gives us another route towards an analysis.

Consider some (u_k, i_k) from which a level k transition emerges. By model completeness, there will be a (u_{k+1}, i_{k+1}), related to (u_k, i_k) by $P^{\text{Def}}{}_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})$, from which a level $k+1$ transition emerges, and vice versa. We fix (u_k, i_k) and (u_{k+1}, i_{k+1}) for the next few paragraphs. By interfeature determinism, both transitions will belong to features $f_{\text{a},k}$ and $f_{\text{y},k+1}$, each unique within the context of its level. There are now three possibilities, which by interfeature determinism again are mutually exclusive, (P1), (P2), (P3):

(P1) $f_{\text{y},k+1}$ is the image under G_{k+1}, $P_{Op,k+1}$, $O_{Op,k+1}$ of $f_{\text{a},k}$.

In this case for every level $k+1$ transition u_{k+1} -$(i_{k+1}, f_{\text{a},k+1}, o_{k+1})$-› u'_{k+1} there will be a level k transition u_k -$(i_k, f_{\text{a},k}, o_k)$-› u'_k related to it by:

$$\begin{aligned}&(G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1}) \wedge \\ &stp_{f_\text{a}}(u_k, i_k, u'_k, o_k) \wedge stp_{f_\text{a}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1}) \wedge \\ &G_{k+1}(u'_k, u'_{k+1}) \wedge O_{Op,k+1}(o_k, o_{k+1}; u'_k, u'_{k+1}, i_k, i_{k+1}, u_k, u_{k+1})) \qquad (9.7)\end{aligned}$$

because all of the level $k+1$ transitions are constructed precisely by mapping all of the level k transitions through G_{k+1}, $P_{Op,k+1}$, G'_{k+1}, $O_{Op,k+1}$. In this case we reestablish the retrieve relation, and $\text{pre}^{\text{Ret}}{}_{Op,k+1}(u_k, i_k, u_{k+1}, i_{k+1})$, the refining simulation given by (5.11), holds.

(P2) $f_{\text{y},k+1}$ is the image under G_{k+1}, $P_{Op,k+1}$, $O_{Op,k+1}$ of some level k feature $f_{\text{b},k} \neq f_{\text{a},k}$.

This can arise because a condition somewhere in $\Phi_k[\phi]$ was modified, making $f_{\text{b},k+1}$ active at (u_{k+1}, i_{k+1}) whereas $f_{\text{a},k}$ was active at (u_k, i_k). In this case, interfeature independence, in the shape of (9.6), ensures that for suitable u'_k, o_k, u'_{k+1}, o_{k+1}, we have:

$$\begin{aligned}\neg\, &(G_{k+1}(u_k, u_{k+j}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1}) \wedge \\ &\quad stp_{f_\text{a}}(u_k, i_k, u'_k, o_k) \wedge stp_{f_\text{b}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1}) \wedge \\ &\quad G_{k+1}(u'_k, u'_{k+1}) \wedge O_{Op,k+1}(o_k, o_{k+1}; u'_k, u'_{k+1}, i_k, i_{k+1}, u_k, u_{k+1})) \qquad (9.8)\end{aligned}$$

i.e. the negation of refining simulation, which is equivalent to:

$$\begin{aligned}&G_{k+1}(u_k, u_{k+j}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1}) \wedge \\ &stp_{f_\text{a}}(u_k, i_k, u'_k, o_k) \wedge stp_{f_\text{b}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1}) \;\Rightarrow \\ &\quad \neg\, (G_{k+1}(u'_k, u'_{k+1}) \wedge O_{Op,k+1}(o_k, o_{k+1}; u'_k, u'_{k+1}, i_k, i_{k+1}, u_k, u_{k+1})) \qquad (9.9)\end{aligned}$$

This allows us to conclude that the hypotheses of (9.9) are sufficient to imply the whole of what would be the default concedes relation (as given by (5.9)) for this (u_k, i_k) and (u_{k+1}, i_{k+1}). As a result, when building the complete concedes relation, including these hypotheses will be enough to express what is needed.

(P3) $f_{\text{y},k+1}$ is *not* the G_{k+1}, $P_{Op,k+1}$, $O_{Op,k+1}$ image of some level k feature $f_{\text{b},k} \neq f_{\text{a},k}$.

In this case $f_{\text{y},k+1}$ is a new feature in the FOD, freshly introduced via $\kappa \Rightarrow \gamma(\kappa)$. The same arguments as in the (P2) case apply, and the analogues of (9.8) and (9.9) hold.

From now on we can treat cases (P2) and (P3) simultaneously, calling the level $k+1$ feature for these two cases f_e regardless of its origins. Either way, for a related pair of features f_a and f_e, the given u_k, i_k, u_{k+1}, i_{k+1}, and suitable u'_k, o_k, u'_{k+1}, o_{k+1}, we have:

$$\begin{aligned}&(G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1}) \wedge \\ &stp_{f_\text{a}}(u_k, i_k, u'_k, o_k) \wedge stp_{f_\text{e}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1})) \qquad (9.10)\end{aligned}$$

Since the above just dealt with a single pair of individual features at the two related levels, to build the complete concedes relation we must accumulate all instances of such pairs into a single entity. We are guided by the partitions of $\mathsf{U}_k \times \mathsf{I}_{Op,k}$ and $\mathsf{U}_{k+1} \times \mathsf{I}_{Op,k+1}$ induced from the NF theorem, which tell us how to fit the pieces together. The next definition captures pairs of pieces of the active domains (calculated with respect to $\mathsf{U}_k \times \mathsf{I}_{Op,k}$ and $\mathsf{U}_{k+1} \times \mathsf{I}_{Op,k+1}$) of distinct features at two adjacent levels of the development such that $G_{k+1} \wedge P_{Op,k+1}$ hold.

Definition 9.5 The offdiagonal active domain of a pair of distinct features $f_{\mathrm{b},k}$ and $f_{\mathrm{g},k+1}$ with respect to the given development step is defined by:

$$
\begin{aligned}
&\mathrm{dom}_{\mathrm{ODact}}(f_{\mathrm{b},k}, f_{\mathrm{g},k+1}) \equiv \\
&\quad \{(u_k, i_k, u_{k+1}, i_{k+1}) \mid f_{\mathrm{b}} \neq f_{\mathrm{g}}, \\
&\qquad (u_k, i_k) \in \mathrm{dom}_{\mathrm{act}}(f_{\mathrm{b},k}) \wedge (u_{k+1}, i_{k+1}) \in \mathrm{dom}_{\mathrm{act}}(f_{\mathrm{g},k+1}) \wedge \\
&\qquad G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1})\}
\end{aligned}
\tag{9.11}
$$

We can now define the complete concedes relation for levels k and $k+1$ as follows. In effect we write out the default concedes relation that relates Op_k and Op_{k+1} except that instead of writing it in its original form, we decompose it into the pieces belonging to pairs of distinct features restricted to the relevant offdiagonal active domains, giving rise to the big disjunction in (9.12).

Definition 9.6 The concedes relation appropriate to a FO development step in the style described is:

$$
\begin{aligned}
&C_{Op,k+1}(u'_k, u'_{k+1}, o_k, o_{k+1}; i_k, i_{k+1}, u_k, u_{k+1}) \equiv \\
&\quad (G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1}) \wedge \\
&\quad (\textstyle\bigvee_{(f_{\mathrm{b}} \neq f_{\mathrm{g}})} ((u_k, i_k, u_{k+1}, i_{k+1}) \in \mathrm{dom}_{\mathrm{ODact}}(f_{\mathrm{b},k}, f_{\mathrm{g},k+1}) \wedge \\
&\qquad\qquad stp_{f_{\mathrm{b}}}(u_k, i_k, u'_k, o_k) \wedge stp_{f_{\mathrm{g}}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1}))))
\end{aligned}
\tag{9.12}
$$

It is easy to see that in the (P2) and (P3) cases we have $\mathrm{pre}^{\mathrm{Con}}{}_{Op,k+1}(u_k, i_k, u_{k+1}, i_{k+1})$, i.e. conceding simulation as given by (5.12), holding for the concedes relation (9.12), as we would expect.

Theorem 9.7 The concedes relation (9.12) is regular.

Proof. Consider $C_{Op,k+1}$. Let $\pi_k(u'_k, o_k; i_k, u_k)$ be the projection that takes (u'_k, o_k, i_k, u_k) to (i_k, u_k); and let $\pi_{k+1}(u'_{k+1}, o_{k+1}; i_{k+1}, u_{k+1})$ be the projection that takes $(u'_{k+1}, o_{k+1}, i_{k+1}, u_{k+1})$ to (i_{k+1}, u_{k+1}). Let $\iota_k(u'_k, o_k, i_k, u_k)$ be the injection of $\mathrm{dom}(C_{Op,k+1})$ into $\mathsf{U}_k \times \mathsf{O}_{Op,k} \times \mathsf{I}_{Op,k} \times \mathsf{U}_k$, and let $\iota_{k+1}(u'_{k+1}, o_{k+1}, i_{k+1}, u_{k+1})$ be the injection of $\mathrm{rng}(C_{Op,k+1})$ into $\mathsf{U}_{k+1} \times \mathsf{O}_{Op,k+1} \times \mathsf{I}_{Op,k+1} \times \mathsf{U}_{k+1}$. Let $f \fatsemi g^{-1}$ be a difunctional presentation of $G_{k+1} \wedge P_{Op,k+1}$. Then we have a difunctional presentation of $C_{Op,k+1}$ given by:

$$
\begin{aligned}
&C_{Op,k+1}(u'_k, u'_{k+1}, o_k, o_{k+1}; i_k, i_{k+1}, u_k, u_{k+1}) \equiv \\
&\quad (\iota_k(u'_k, o_k, i_k, u_k) \fatsemi \pi_k(u'_k, o_k; i_k, u_k) \fatsemi f) \fatsemi \\
&\quad (\iota_{k+1}(u'_{k+1}, o_{k+1}, i_{k+1}, u_{k+1}) \fatsemi \pi_{k+1}(u'_{k+1}, o_{k+1}; i_{k+1}, u_{k+1}) \fatsemi g)^{-1}
\end{aligned}
\tag{9.13}
$$

This shows that $C_{Op,k+1}$ is regular. ☺

So our retrenchments have regular data. The next question that naturally arises is to ask whether they respect their regular data. Here the answer is no. For consider the following situation in which there is no I/O. We have at level k, a value u of the state

variable u_k, from which a transition of feature $f_{a,k}$ issues. At level level $k+1$ the state variables are u_{k+1} which consist of a pair of values (u, w) where the value w is needed by feature $f_{e,k+1}$, newly introduced at level $k+1$. The retrieve relation $G_{k+1}(u_k, u_{k+1})$ is just the inverse projection that relates u at level k to (u, w) for any w at level $k+1$. Now we suppose that for a value w_1 there is a level $k+1$ transition of $f_{a,k+1}$, namely (u, w_1) -($f_{a,k+1}$)-› (u', w'_1), and for a value w_2 there is a level $k+1$ transition of $f_{e,k+1}$, namely (u, w_2) -($f_{e,k+1}$)-› (u'_2, w'_2). (This is quite reasonable since there is no requirement for any individual feature to be model complete. In fact we saw such behaviour above when $FobAccessible_3$ overrode part of the behaviour of $Lock_{2/3}$ in the car locking example.) In such a case, for the (u, w_1) -($f_{a,k+1}$)-› (u', w'_1) transition, we would establish $G_{k+1}(u', (u', w'_1))$, since the level $k+1$ transition would just be a copy of a level k transition, u -($f_{a,k}$)-› u'; and for the (u, w_2) -($f_{e,k+1}$)-› (u'_2, w'_2) transition we would have $C_{Op,k+1}(u'_k, u'_{k+1} \ldots)$ because feature $f_{e,k+1}$ is active. Now (u, w_1) and (u, w_2) are in the same $(G_{k+1} \wedge P_{Op,k+1})$ equivalence class, since they are both related by $G_{k+1} \wedge P_{Op,k+1}$ to u at level k. But (u', w'_1) and (u'_2, w'_2) are not in the same $C_{Op,k+1}$ equivalence class, since $C_{Op,k+1}$ is only defined when the level k feature and level $k+1$ feature are different, by (9.12); in particular $C_{Op,k+1}$ is not defined for (u', w'_1). So condition (4) of Definition 6.7 is violated and retrenchments with concedes relations such as (9.12) do not respect their regular data. (In particular, the tidiness property discussed in [Banach and Jeske (2009b)] is too strong a property to expect in feature engineering.)

9.4 The Neatness Theorem

Theorem 9.8 The retrenchments given by (9.1)-(9.3) and (9.12) are neat.

Proof. Consider some (u_k, i_k) related to some (u_{k+1}, i_{k+1}) by $G_{k+1} \wedge P_{Op,k+1}$. If we have the same feature active at both levels, then we have $\mathrm{pre}^{\mathrm{Ret}}{}_{Op,k+1}(u_k, i_k, u_{k+1}, i_{k+1})$, defined in (5.11). By interfeature determinism, no other features can be active for this (u_k, i_k) and (u_{k+1}, i_{k+1}) so for the after-states and outputs which are related to these (u_k, i_k) and (u_{k+1}, i_{k+1}), $C_{Op,k+1}$ will not be defined (because we were careful to relate only offdiagonal active domains via $C_{Op,k+1}$), and so $\mathrm{pre}^{\mathrm{Con}}{}_{Op,k+1}(u_k, i_k, u_{k+1}, i_{k+1})$ will be false, making (5.10) true in this case. By contrast, suppose for (u_k, i_k) and (u_{k+1}, i_{k+1}) that two different features are active, and we therefore have $C_{Op,k+1}$, and $\mathrm{pre}^{\mathrm{Con}}{}_{Op,k+1}(u_k, i_k, u_{k+1}, i_{k+1})$. Then interfeature determinism says no other features can be active for (u_k, i_k) and (u_{k+1}, i_{k+1}), and so interfeature independence allows us to conclude that $G'_{k+1} \wedge O_{Op,k+1}$ will not hold 'fortuitously'. The latter is a possibility we must guard against, since $G'_{k+1} \wedge O_{Op,k+1}$ is a globally defined relation from $\mathsf{U}_k \times \mathsf{O}_{Op,k} \times \mathsf{I}_{Op,k} \times \mathsf{U}_k$, to $\mathsf{U}_{k+1} \times \mathsf{O}_{Op,k+1} \times \mathsf{I}_{Op,k+1} \times \mathsf{U}_{k+1}$. If $G'_{k+1} \wedge O_{Op,k+1}$ does not hold then $\mathrm{pre}^{\mathrm{Ret}}{}_{Op,k+1}(u_k, i_k, u_{k+1}, i_{k+1})$ cannot hold, making (5.10) true in this case also. Neatness thus follows. ☺

Example II (Car Locking) Above, we showed that we could express the development step from the level 2 locking operation to the level 3 operation via the FE:

$$Lock_3 = ((Lock_{2/3} \uplus LockSkip_3) \lessdot FobAccessible_3) \tag{9.14}$$

Given the theory just developed, we can calculate the retrenchment that takes us from the level 2 to the level 3 operation as follows. We start by writing down all the ingredients.

States: $\mathsf{U}_2 = \{dr\text{-}lck, dr\text{-}unl\} \times \{ps\text{-}lck, ps\text{-}unl\}$
$$\mathsf{U}_3 = \{dr\text{-}lck, dr\text{-}unl\} \times \{ps\text{-}lck, ps\text{-}unl\} \times \{fob\text{-}in, fob\text{-}out\} \quad (9.15)$$

Abstract Features:
$$(\mathsf{U}_2 - (dr\text{-}lck, ps\text{-}lck))\ \text{-}(Lock_2)\text{-}\!\!\rightarrow (dr\text{-}lck, ps\text{-}lck) \quad (9.16)$$

Concrete Features:
$$(\mathsf{U}_2 - (dr\text{-}lck, ps\text{-}lck), *)\ \text{-}(Lock_{2/3})\text{-}\!\!\rightarrow (dr\text{-}lck, ps\text{-}lck, *)$$
$$(dr\text{-}lck, ps\text{-}lck, *)\ \text{-}(LockSkip_3)\text{-}\!\!\rightarrow (dr\text{-}lck, ps\text{-}lck, *)$$
$$(*, *, fob\text{-}in)\ \text{-}(FobAccessible_3)\text{-}\!\!\rightarrow (dr\text{-}unl, ps\text{-}lck, fob\text{-}in) \quad (9.17)$$

Retrieve Relation: $G((d_2, p_2), (d_3, p_3, f_3)) \ \equiv\ (d_2 = d_3 \wedge p_2 = p_3)$ (9.18)

Within Relation: true
Output Relation: true (9.19)

Offdiagonal Active Domains:
$$\mathrm{dom}_{\mathrm{ODact}}(Lock_2, LockSkip_3) \ \equiv\ \varnothing$$
$$\mathrm{dom}_{\mathrm{ODact}}(Lock_2, FobAccessible_3) \ \equiv$$
$$\{((d_2, p_2), (d_3, p_3, f_3)) \mid G((d_2, p_2), (d_3, p_3, f_3)) \wedge f_3 = fob\text{-}in_3\} \quad (9.20)$$

Concedes Relation: $C_{Lock}((d_2, p_2), (d_3, p_3, f_3)) \ \equiv$
$$(((d_2, p_2), (d_3, p_3, f_3)) \in \mathrm{dom}_{\mathrm{ODact}}(Lock_2, FobAccessible_3) \wedge$$
$$(d_2, p_2) \neq (dr\text{-}lck, ps\text{-}lck) \wedge (d'_2, p'_2) = (dr\text{-}lck, ps\text{-}lck) \wedge$$
$$(d_3, p_3) \neq (dr\text{-}lck, ps\text{-}lck) \wedge (d'_3, p'_3) = (dr\text{-}unl, ps\text{-}lck) \wedge f'_3 = fob\text{-}in) \quad (9.21)$$

It is clear that the retrenchment operation PO (5.2) will be provable with the above data. The straightforward results align pleasantly with the fact that while at level 2, $Lock_2$ is its own normal form, at level 3, the normal form of $Lock_3$ can be written as:

$$Lock_3 = \{(\mathsf{U}_2 - (dr\text{-}lck, ps\text{-}lck), fob\text{-}out)\} \lhd Lock_{2/3} \uplus$$
$$\{(dr\text{-}lck, ps\text{-}lck, fob\text{-}out)\} \lhd LockSkip_3 \uplus$$
$$\{(\mathsf{U}_2, fob\text{-}in)\} \lhd FobAccessible_3 \quad (9.22)$$

We see a three-way split in the NF of $Lock_3$; the $Lock_{2/3}$ component needs no provision in C_{Lock} since it is handled by G; the $LockSkip_3$ component also needs no provision in C_{Lock} since its active domain is not related via G to the active domain of $Lock_2$; finally the $FobAccessible_3$ component does make a non-trivial contribution to C_{Lock}.

9.5 Neatness and Composition

Even though we have neatness of the retrenchments for individual development steps, we do not in general have strong enough properties to be able to deduce that compositions of such steps (done according to the specific recipe for composing neat retrenchments discussed in [Banach and Jeske (2009b)]) yield neat retrenchments, nor that the composition of such retrenchments, even if neat, is necessarily associative. (Because neat retrenchments are retrenchments, and the normal composition of retrenchments is associative, the normal composition of neat retrenchments will be associative of course.)

Counterexample VI (Non-Neatness of compositions) Consider an operation defined at level 0 by $(f_{\mathrm{b}} \lessdot_p f_{\mathrm{a}})$, where we suppose that the domains of the two interfeature independent features f_{a} and f_{b} are equal and p is just $(u, i) \in \mathrm{dom}(f_{\mathrm{a}})$. Consider

the development step in which p is replaced by $\neg p$, giving at level 1, $(f_\mathrm{b} \lessdot_{\neg p} f_\mathrm{a})$. A nontrivial concedes relation will be needed to describe the fact that one behaviour is replaced by another in the whole domain. Consider the further development step in which $\neg p$ is replaced by p, giving at level 2, $(f_\mathrm{b} \lessdot_p f_\mathrm{a})$ again. Another nontrivial concedes relation will be needed here to undo the damage caused by the first one; in fact it will be the transpose of the first concedes relation. However the composition of these development steps is the identity development step, and for an identity development step we can say two things that are both obvious even without examining the details. Firstly, we can be completely confident that an identity development step can be described using only the refining part of the simulation capabilities of a retrenchment (i.e. $\overline{G}$, as in (5.13)), so the concedes relation of such a retrenchment does not need to be non-empty. Secondly, we can be equally confident that any composed concedes relation for a composition of retrenchments whose component concedes relations consist of a given relation C and its transpose C^T will not itself be non-empty — predominantly because the composed concedes relation will contain the obviously non-empty sequential composition of C and C^T, a fact that can be checked explicitly by reference to the composition schemes in [Banach et al. (2008)] or [Banach and Jeske (2009b)]. So, in the trivial identity development situation we are discussing, the structure of such composed concessions will always relate a transition back to itself. Thus it is not hard to see that a typical transition of an operation will validate both the $\overline{G}$ and $\overline{C}$ relations of the composed retrenchment, and that the composition will therefore not be neat.

9.6 FODs that are Sequences of Overrides

The general form of modification to FODs that we allow in the passage from one level to another makes it difficult to say anything too specific about how the partition of the $\mathsf{U}_k \times \mathsf{I}_{Op,k}$ promised by the NF theorem evolves from level to level. Of course in any particular case, the calculational strategy of Definition 7.6 *et seq*. will yield a particular answer. However there are some special cases, in which the partition evolves in a more systematic manner, and we illustrate one of these.

Suppose all the FODs for Op are simply sequences of overrides, such as:

$$Op_k \;=\; f_{\mathrm{Def},k} \lessdot f_{\mathrm{q},k} \lessdot f_{\mathrm{p},k} \ldots f_{\mathrm{b},k} \lessdot f_{\mathrm{a},k} \tag{9.23}$$

Here f_Def is some default feature that guarantees model completeness no matter what. (Thus we tacitly assume that at each level the limits of its domain serve to define model completeness, and that no feature is defined beyond the domain of f_Def at any level.) On this basis, we can restrict the progression from development level k to development level $k+1$ to just the insertion of some $f_{\mathrm{g},k+1}$ into the above sequence, changing $(\ldots f_{\mathrm{d},k} \lessdot f_{\mathrm{c},k} \ldots)$ into $(\ldots f_{\mathrm{d},k+1} \lessdot f_{\mathrm{g},k+1} \lessdot f_{\mathrm{c},k+1} \ldots)$.

It is now clear that the only alteration to the definition of Op at level $k+1$ will occur on the subset of the state and input spaces given by $\mathrm{dom}(\mathrm{g}_{k+1} - \mathrm{c}_{k+1} \ldots \mathrm{a}_{k+1}) \equiv \mathrm{dom}(f_{\mathrm{g},k+1}) - \mathrm{dom}(f_{\mathrm{c},k+1} \cup \ldots \cup f_{\mathrm{a},k+1})$. So we have the situation that on the one hand, on the subset of the state and input spaces given by $\mathrm{dom}(f_{\mathrm{c},k+1} \cup \ldots \cup f_{\mathrm{a},k+1})$, the behaviour will be identical to that at level k, since the features occur in the same priority order at both levels, and at both levels their behaviour overrides anything that f_g might do, meaning that the relationship between the behaviours at the two levels will be (trivially) refining, and the retrieve and output relations will take care of things. On the other hand, in the subset of the state and input spaces given by

$\mathrm{dom}(\mathrm{g}_{k+1} - \mathrm{c}_{k+1}\ldots\mathrm{a}_{k+1})$, we know we will need to capture what happens using the concedes relation of a retrenchment, but without delving into the details of offdiagonal active domains, we cannot be sure which pieces of which abstract features will be related to which pieces of the newly infiltrated feature $f_{\mathrm{g},k+1}$ (since $f_{\mathrm{g},k+1}$ can override any of them). Nevertheless we know enough to be able to write down a generic concession $C_{Op,k+1}$ which appears thus:

$$\begin{aligned}
&C_{Op,k+1}(u'_k, u'_{k+1}, o_k, o_{k+1}; i_k, i_{k+1}, u_k, u_{k+1}) \equiv \\
&\quad (G_{k+1}(u_k, u_{k+1}) \wedge P_{Op,k+1}(i_k, i_{k+1}, u_k, u_{k+1}) \wedge \\
&\quad (((u_k, i_k) \in \mathrm{dom}(\mathrm{d}_k - \mathrm{c}_k\ldots\mathrm{a}_k) \wedge stp_{f_\mathrm{d}}(u_k, i_k, u'_k, o_k)) \vee \\
&\quad\quad \ldots \qquad \ldots \vee \\
&\quad ((u_k, i_k) \in \mathrm{dom}(\mathrm{Def}_k - \mathrm{q}_k\ldots\mathrm{a}_k) \wedge stp_{f_\mathrm{Def}}(u_k, i_k, u'_k, o_k))) \wedge \\
&\quad ((u_{k+1}, i_{k+1}) \in \mathrm{dom}(\mathrm{g}_{k+1} - \mathrm{c}_{k+1}\ldots\mathrm{a}_{k+1}) \wedge stp_{f_\mathrm{g}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1}))) \qquad (9.24)
\end{aligned}$$

In (9.24) we have assumed that $\mathrm{dom}(f_{\mathrm{g},k+1}) \subseteq \mathrm{dom}(f_{\mathrm{Def},k+1})$, and that the dissection of the concedes relation into specific feature *stp* relations on the various domains is just the decomposition of $stp_{Op_k}(u_k, i_k, u'_k, o_k)$ and $stp_{Op_{k+1}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1})$ into their constituents.

When we have a sequence of such modifications, the concedes relations that describe the resulting operation depend on the order in which different features are inserted into the overall FOD. Eg. suppose after inserting $f_{\mathrm{g},k+1}$ above, we next insert feature $f_{\mathrm{h},k+2}$. Then we have two different outcomes depending on whether $f_{\mathrm{h},k+2}$ is overridden by $f_{\mathrm{g},k+2}$ or not, i.e. whether it occurs lower down the override hierarchy.

Suppose $f_{\mathrm{h},k+2}$ is inserted next in the chain of priority immediately beneath $f_{\mathrm{g},k+1}$, giving $(\ldots f_{\mathrm{d},k+2} \lessdot f_{\mathrm{h},k+2} \lessdot f_{\mathrm{g},k+2} \lessdot f_{\mathrm{c},k+2} \ldots)$. Then following (9.24), the corresponding concedes relation reads:

$$\begin{aligned}
&C_{Op,k+2}(u'_{k+1}, u'_{k+2}, o_{k+1}, o_{k+2}; i_{k+1}, i_{k+2}, u_{k+1}, u_{k+2}) \equiv \\
&\quad (G_{k+2}(u_{k+1}, u_{k+2}) \wedge P_{Op,k+2}(i_{k+1}, i_{k+2}, u_{k+1}, u_{k+2}) \wedge \\
&\quad (((u_{k+1}, i_{k+1}) \in \mathrm{dom}(\mathrm{d}_{k+1} - \mathrm{g}_{k+1}\ldots\mathrm{a}_{k+1}) \wedge stp_{f_\mathrm{d}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1})) \vee \\
&\quad\quad \ldots \qquad \ldots \vee \\
&\quad ((u_{k+1}, i_{k+1}) \in \mathrm{dom}(\mathrm{Def}_{k+1} - \mathrm{q}_{k+1}\ldots\mathrm{a}_{k+1}) \wedge stp_{f_\mathrm{Def}}(u_{k+1}, i_{k+1}, u'_{k+1}, o_{k+1}))) \wedge \\
&\quad ((u_{k+2}, i_{k+2}) \in \mathrm{dom}(\mathrm{h}_{k+2} - \mathrm{g}_{k+2}\ldots\mathrm{a}_{k+2}) \wedge stp_{f_\mathrm{h}}(u_{k+2}, i_{k+2}, u'_{k+2}, o_{k+2})) \qquad (9.25)
\end{aligned}$$

We can see that since in (9.24), $(u_{k+1}, i_{k+1}) \in \mathrm{dom}(\mathrm{g}_{k+1} - \mathrm{c}_{k+1}\ldots\mathrm{a}_{k+1})$ —to which $\mathrm{dom}(\mathrm{g}_{k+1})$ contributes positively— guards $f_{\mathrm{g},k+1}$, and in (9.25), $(u_{k+1}, i_{k+1}) \in \mathrm{dom}(\mathrm{d}_{k+1} - \mathrm{g}_{k+1}\ldots\mathrm{a}_{k+1})$ —to which $\mathrm{dom}(\mathrm{g}_{k+1})$ contributes negatively— guards $f_{\mathrm{d},k+1}$ (with similar effects for all the other features that contribute to the level $k+1$ values of $C_{Op,k+2}$), then (9.24) and (9.25) compose to give the empty relation, and other contributions to the composed concession have to be relied on for the soundness of the retrenchment.

On the other hand, if $f_{\mathrm{h},k+2}$ overrides $f_{\mathrm{g},k+2}$ on a nonempty overlap of domains, then some of the modification captured by $C_{Op,k+1}$ earlier, will be undone by the new modification, and this will be captured by $C_{Op,k+2}$, such that in a subsequent composed concession, the composition of $C_{Op,k+1}$ and $C_{Op,k+2}$ will contribute non-trivially. That this appears different from the previous case is not at all surprising since $(f_{\mathrm{d},k+2} \lessdot f_{\mathrm{h},k+2} \lessdot f_{\mathrm{g},k+2} \lessdot f_{\mathrm{c},k+2})$ is not the same as $(f_{\mathrm{d},k+2} \lessdot f_{\mathrm{g},k+2} \lessdot f_{\mathrm{h},k+2} \lessdot f_{\mathrm{c},k+2})$, either syntactically or semantically. However these two feature expressions have *the same*

shape, so one might anticipate a similar overall shape to emerge for the concedes relations from $(\dots f_{\mathrm{d},k+2} \lessdot f_{\mathrm{c},k+2} \dots)$ either to $(\dots f_{\mathrm{d},k+2} \lessdot f_{\mathrm{h},k+2} \lessdot f_{\mathrm{g},k+2} \lessdot f_{\mathrm{c},k+2} \dots)$ or to $(\dots f_{\mathrm{d},k+2} \lessdot f_{\mathrm{g},k+2} \lessdot f_{\mathrm{h},k+2} \lessdot f_{\mathrm{c},k+2} \dots)$. In benign cases, appropriate manipulations of the two compositions can bring out the expected similarity.

9.7 Telephone Feature Interaction

We conclude this lengthy section by pointing out that in [Banach and Poppleton (2003)], there is a toy feature engineering case study, focused on telephone system feature interaction, which is done largely along the lines of the theory above. It is similar in spirit to Example III although it is much more extensive. The fact that it is very much a toy is a consequence of using a formalism similar to the one in this paper, which relates a single step at one level to a single step at the next level. Such an approach has a very real drawback in that the behavioural or multistep aspects of genuine telephony applications are abstracted away, since in a typical interaction with a real telephone system one goes through a number of phases before the interaction completes. Disregarding this finer level of granularity undoubtedly undermines the credibility of any such decription — still, the main point of [Banach and Poppleton (2003)] was to illustrate retrenchment, not to advance the state of the art in telephony. Neverthless, both that paper and the present paper, support the view that a development of retrenchment based ideas more accurately targeted at the needs of realistic telephone feature engineering problems, would enjoy a good measure of success.

Aside from the previous point, there is a crucial difference between the theory of this paper and that of [Banach and Poppleton (2003)] since the case study there is done using primitive retrenchment[18] rather than the output retrenchment of this paper.[19] At a number of points, especially when we want to distinguish between system transitions that differ only in their outputs, the insensitivity of primitive retrenchment to this kind of situation inhibits its use in giving a fluent account of the matter. It would be an undemanding exercise to repeat the case study in [Banach and Poppleton (2003)] in the framework of the present paper, and to carry out successfully the programme discussed there, but only partially carried through.

10 Conclusions

In the preceding sections, we started out by defining operations and features, and then overviewed the feature engineering approach of the paper as a whole, giving a number of examples. We then reviewed what we needed of retrenchment and refinement machinery, including their interworking via the *Tower Pattern*. We then examined regular relations, and presented general properties, including results on sequential composition. We were then able to embark on feature engineering proper, and

---

18. Primitive retrenchment is unlike conventional retrenchment (which is called output retrenchment to distinguish it from the primitive kind), in that primitive retrenchment does not have a separate output relation. Both kinds of retrenchment are introduced in [Banach et al. (2007)], where the relationship between them is discussed. The retrenchments of this paper are all output retrenchments.

19. There is another technical difference between this paper and [Banach and Poppleton (2003)]. In the latter input, output, and state spaces were assumed fixed *ab initio*, and large enough to accommodate all features needed at any point in the development; thus making G and P identities (there was no O of course).

presenting a language for feature combination which enabled us to build operations out of arbitrarily complex combinations of features. An important milestone was the normal form theorem, which transformed the complexity of arbitrary feature expressions to that of a fixed schema. Operation evolution, tackled via the rewriting of feature expressions, was dealt with next, the rewrites allowing the transformation of feature oriented descriptions of operations at arbitrary deeply nested places in the feature expression.

The objective for all this was to capture the semantic consequences of the manipulation of the feature oriented description using retrenchment. For this, the normal form theorem proved useful since it allowed generic retrenchment data to be constructed on the basis of a fairly flat description of an arbitrary operation. Under quite mild conditions, the generic retrenchments needed for this were shown to be neat default retrenchments. We illustrated various points of the theory being constructed using a couple of running examples.

The focus of the paper was obviously to develop the retrenchment perspective on feature engineering. However, as noted in the Introduction, refinement based approaches using superposition refinement can be used when the features of interest are appropriately compatible and are being assembled in a suitable order. Given an arbitrary feature engineering development process, while there is no reason to assume that the whole of it necessarily enjoys these benign properties, there is equally no reason to assume that parts of it cannot enjoy them. The *Tower Pattern* of Fig. 1 is what allows these two perspectives on feature engineering to cooperate smoothly. For those parts of the process that can be conveniently handled via refinement, we can use refinement; for those parts that cannot, we can use retrenchment. The results of the two approaches can then be composed to give an overall consistent result. *Apropos* composition, we made various remarks to the effect that the stronger properties of the retrenchments we built, did not necessarily persist through composition, but we emphasise that it is only those stronger properties that might fail — the robustness of the description via cooperating retrenchments and refinements is unaffected.

References

Abrial J.-R. (2010); Modeling in Event B. Cambridge University Press.

Back R. J. R. (2002); Software Construction by Stepwise Feature Introduction. *in*: Proc. ZB-02, Bert, Bowen, Henson, Robinson (eds.), LNCS **2272**, 162-183, Springer.

Back R. J. R., Sere K. (1996); Superposition Refinement of Reactive Systems. Form. Asp. Comp. **8**, 324-346.

Banach R. (1994); Regular Relations and Bicartesian Squares. Theor. Comp. Sci. **129**, 187-192.

Banach R. (1995); On Regularity in Software Design. Sci. Comp. Prog. **24**, 221-248.

Banach R., Jeske C. (2009a); Retrenchment and Refinement Interworking: the Tower Theorems. *submitted*.

Banach R., Jeske C. (2009b); Stronger Compositions for Retrenchments. J. Log. Alg. Prog., **79**, 215-232.

Banach R., Poppleton M., Jeske C., Stepney S. (2005); Retrenching the Purse: Finite Sequence Numbers, and the Tower Pattern. *in*: Proc. FM-05, Fitzgerald, Hayes, Tarlecki (eds.), LNCS **3582**, 382-398, Springer.

Banach R., Jeske C., Poppleton M. (2008); Composition Mechanisms for Retrenchment. J. Log. Alg. Prog., **75**, 209-229.

Banach R., Poppleton M. (2003); Retrenching Partial Requirements into System Definitions: A Simple Feature Interaction Case Study. Req. Eng. J. **8**, 266-288.

Banach R., Poppleton M., Jeske C., Stepney S. (2007); Engineering and Theoretical Underpinnings of Retrenchment. Sci. Comp. Prog., **67**, 301-329.

de Roever W-P., Engelhardt K. (1998); Data Refinement: Model-Oriented Proof Methods and their Comparison. Cambridge University Press.

Francez N., Forman I. R. (1990); Superimposition for Interactive Processes. *in*: Proc. CONCUR-90, Baeten, Klop (eds.), LNCS **458**, 230-245, Springer.

Jeske C. (2005); Algebraic Integration of Retrenchment ans Refinement. PhD. Thesis, School of Computer Science, University of Manchester.

Katz S. (1993); A Superimposition Control Construct for Distributed Systems. ACM Trans. Prog. Lang. Sys. **15**, 337-356.

Schmidt G., Ströhlhein T. (1993); Relations and Graphs, Discrete Mathematics for Computer Scientists. Springer.

Suppes P. (1972); Axiomatic Set Theory. Dover.

Tarski A. (1941); On the Calculus of Relations. J. Sym. Logic **6**, 73-89.

Zave P. (2001); Requirements for Evolving Systems: A Telecommunications Perspective. *in*: Proc. 5th IEEE Int. Symp. Requirements Engineering, 2-9.