Defaults in Specifications

Mark Ryan
Department of Computing
Imperial College
London SW7 2BZ
E-mail: mdr@doc.ic.ac.uk

Abstract

A formalism is motivated and described for representing *defaults in specifications*. The formalism is called Ordered Theory Presentations. The ability to represent defaults narrows the gap between a customer's initial requirements and a formal specification, and supports reuse on both a small and a large scale. We illustrate the issues throughout with reference to the lift example.

We also consider the application of the formalism to *specification revision*.

1 Introduction

Imagine specifying a lift system. There is a lift, with n buttons and n indicator lights inside, and there are doors. The buttons inside the lift are for requesting where the lift goes, and the indicator lights register such requests. There are floors, each with two buttons and indicator lights (one for requesting to go up and one for down). The indicator lights switch on in response to button pressings and off when the lift arrives at a floor; and the lift goes from floor to floor depending on the state of the lights. Sometimes it opens its doors to let people in and out.

Here are some of the *customer's requirements*:

1. If the lift is at the ith floor and it goes down by one floor, then it is at the $(i-1)$th floor.
2. Pressing the alarm button causes the alarm to sound.
3. The lift may not move up or down unless the doors are closed.
4. When the lift is at the ith floor, the indicator light for the ith floor is off.
5. Pressing[1] a button for a floor causes the corresponding indicator light to come on.

[1] Throughout this paper we take the press action to be an atomic press-and-release.

Although they are initially appealing, examination reveals that these statements are (in the context of other even more plausible statements) contradictory. For example, statements 4 and 5 contradict if we assume, as we should, that pressing a button does not directly affect the position of the lift among the floors. The contradiction arises when we consider what happens when the ith button is pressed and the lift is already at the ith floor. Statement 5 says that the light should come on, while 4 (together with the assumption that the pressing did not instantly change the floor) says that it should be off.

In this case it is clear that 4 should override 5 when they conflict. However, that does not mean that we can remove 5 from the specification altogether, as it is required for all the other circumstances in which the lift is *not* at the relevant floor. That is to say, statement 4 should only partially override statement 5.

The situation for the lift's indicator lights is in fact even worse, for some others of the customer's requirements are

6. Holding down any button makes the light turn on (until the button is released).
7. In the event of a power failure, all the lights will turn off.
8. If the lift was between floors when the power failure took place, the alarm will sound (it is battery operated).

And so on. These statements further interfere with statements 1 to 5, overriding them in certain circumstances.

The upshot of all this, then, is that statement 5 should be thought of as a *default* — a statement which is true unless some *stronger* sentence overrides it. Much of this paper is about how such defaults in specifications should be handled formally.

Among the questions addressed in this paper are the following:

- Can we make sense of such *defaults* in specifica-

tions?

- How formally can we allow one partially to override another, in the way that 4 partially overrides 5 in the example?
- How do we know what should override what, anyway?

The benefit of answering these questions is to narrow the gap between the customers initial requirements and the formal specification. The idea is the following. In the usual simple model of the software development process the formal specification stands between the informally stipulated requirements and the final code. There are thus two components to the process, one which takes the requirements and constructs the specification, and the other taking the specification to the program code. We are interested in the first process, and we believe that if we can represent more of the original intentions of the customer in the specification then that process becomes easier. Part of those original intentions includes defaults and the implication that certain statements override others when they conflict. The issue, then, is to represent these formally in the specification, thus making the specification a more accurate reflection of the requirements.

Also in this paper, we consider another related topic, which is how revisions of the requirements may be formally represented. Revisions may come about during the software development process when the customer is made aware of the consequences of his or her stipulations, which he or she may find undesirable. In that case, they are introduced in order to remove these undesirable consequences. Revising the specification then consists of adding a new sentence which overrides (if necessary, ad only partially) the other sentences.

Defaults and revisions in specifications are very similar from the formal point of view, because they both involve this crucial notion of *overriding* between sentences.

The remainder of the paper is organised as follows. In **section 2** we review a standard formal language and logic for describing state-based systems, known as Modal Action Logic, or MAL. This logic forms the basis of the system described in this paper. In **section 3** we discuss how specifications should be *structured* (or modularised) and how this gives a basis for defining the overriding relation between statements. In this section, and throughout the paper, we make continued reference to the lift specification example. In **section 4** we develop the formal machinery for the overriding relation, and apply it to the lift example. Finally, objections to the approach are considered and, I hope, quashed, in **section 6**, and comparisons with other default systems are touched upon in the last section.

2 Modal action logic

Modal action logics (also known as *dynamic logics* or *multi-modal logics*), have for over a decade been used to specify state-based software systems. The basic idea of modal action logics is to represent actions moving the system from one state to another as 'modalities.' Such a logic has a family of modal operators, one for each action that the system can undergo, and its semantics is given by a set of states and a family of relations on the states, one interpreting each modality. For example, the fact that the action a if executed in a state satisfying condition ϕ results in a state satisfying ψ is expressed by the axiom

$$\phi \rightarrow [a]\psi.$$

There are many accounts of modal action logics [3, 4, 5, 14]. We now describe a simple version which we refer to as MAL below.

A component within a specification is, in logical terms, a *signature* together with a *theory presentation* over the signature. A MAL signature is a pair consisting of a set of action symbols and a set of atomic proposition symbols; the action symbols are used to describe the actions which the system may perform, and the proposition symbols are used to represent the state of the system. Thus, actions update the values of the propositions.

A MAL signature $S = \langle A, P\rangle$ consists, then, of two sets; a set of actions A and a set of propositions P. For example, here are the signatures for some of the objects of the lift system:

button has the signature $\langle\{\text{press}, \text{cancel}\}, \{\text{lit}\}\rangle$. The button may be pressed or cancelled, and has a light which may be on or off.

door has the signature $\langle\{\text{open}, \text{close}\}, \{\text{doors-open}\}\rangle$.

lift-position has the signature $\langle\{\text{up}, \text{down}\}, \{\text{floor}_1, \ldots, \text{floor}_n\}\rangle$. floor_i represents whether the lift is at the ith floor or not.

lift has the signature $\langle\{\text{press}_1, \ldots, \text{press}_n, \text{cancel}_1, \ldots, \text{cancel}_n, \text{open}, \text{close}, \text{up}, \text{down}\}, \{\text{lit}_1, \ldots, \text{lit}_n, \text{floor}_1, \ldots, \text{floor}_n, \text{doors-open}\}\rangle$. Notice the renaming of the press actions.

Given a signature, atomic propositions are composed to form more complex sentences using the usual boolean operators $\wedge$, $\vee$, $\rightarrow$, $\neg$ etc. There is also the

construct $[\cdot]\cdot$ which, as already mentioned, is used to describe the effects of actions. If a is an action symbol and ϕ a sentence (which may also contain action terms) then $[a]\phi$ expresses the fact that ϕ holds after a has taken place. The syntax of formulas is therefore as follows:

$$\begin{array}{l} a \in A, \quad p \in P \\ \phi \ ::= \ p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \\ \qquad \phi_1 \leftrightarrow \phi_2 \mid \neg\phi \mid [a]\phi \end{array}$$

To illustrate this syntax, here again are the five statements about the lift.

1. $\text{floor}_i \rightarrow [\text{down}]\text{floor}_{i-1}$ (for $2 \leqslant i \leqslant n$)
2. [press-alarm]alarm
3. doors-open $\rightarrow$ ([up]$-$ $\wedge$ [down]$-$)
4. $\text{floor}_i \rightarrow \neg\text{lit}_i$
5. $[\text{press}_i]\text{lit}_i$

The semantics of the language we have introduced so far is given by its interpretations. An interpretation M for a signature is a function which takes states to an assignment of truth values to the atomic propositions. States are represented by *traces*. A trace is a sequence of actions in the signature; a finite trace denotes the state which results by performing the actions in the sequence, starting in the initial state. Thus, if σ is a trace and p an atomic proposition, then $M(\sigma)(p)$ is a truth value which says whether p is true or false in the state resulting from performing the actions in σ in the initial state.

Satisfaction in states is defined in the following way:

$$\begin{array}{rll} M(\sigma) \Vdash p & \text{if} & M(\sigma)(p) = \mathbf{t} \\ M(\sigma) \Vdash \neg\phi & \text{if} & M(\sigma) \nVdash \phi \\ M(\sigma) \Vdash \phi \wedge \psi & \text{if} & M(\sigma) \Vdash \phi \text{ and } M(\sigma) \Vdash \psi \\ \text{and similar clauses for } \vee, \rightarrow, \leftrightarrow & & \\ M(\sigma) \Vdash [a]\phi & \text{if} & M(\sigma \circ a) \Vdash \phi \end{array}$$

(In the last clause, $\sigma \circ a$ is σ with a appended.) In this way of handling actions, the logic cannot support concurrent actions. In the papers to which we have already referred a more complex logic is described which handles concurrency. For our present purposes, however, concurrency is not an issue and it is better to avoid unnecessary technicalities.

Satisfaction in *interpretations* is then defined as follows:

$$M \Vdash \phi \text{ if for each } \sigma, M(\sigma) \Vdash \phi.$$

This means that a sentence is true overall in an interpretation iff it is true in every state of the interpretation.

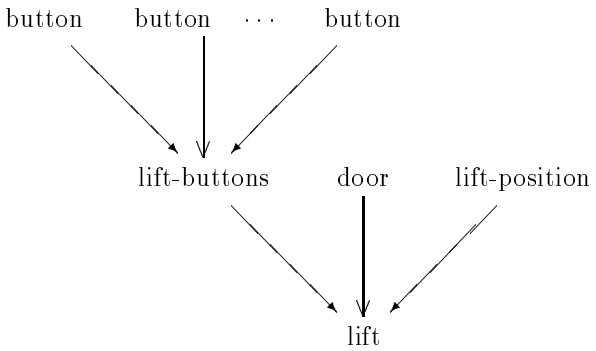
If ? is a set of sentences and ϕ a sentence, ? $\models \phi$ holds if for every M, if $M \Vdash \psi$ for each $\psi \in$? then $M \Vdash \phi$. ? $\models \phi$ is read ? entails ϕ. If ? is the set of axioms of a specification and ? $\models \phi$, then ϕ is a consequence of the specification.

We are now equipped with the syntax and semantics necessary to specify action-based systems. Our aim is to develop the machinery which allows defaults and revisions to be expressed. First, we examine an important element of this, which is how specifications are structured into their components.

3 Structuring MAL specifications

The language and logic introduced so far allows us to specify state-based systems. In cases that there is a large number of action symbols or proposition symbols, however, it makes sense to split up the signature into smaller signatures and specify them separately. This not only enhances readability and writability, but also provides a means of stipulating *locality conditions* to constrain the effects of actions. This is the idea of object encapsulation, or object-oriented specification.

In the last section, we gave the signatures of four of the objects which make up the lift. The lift structure is the following:



The lift is split into three objects: lift-buttons, door and lift-position. The panel of buttons, called lift-buttons, is composed of n copies of button. The button object represents the button and the light together.

Each object of this diagram has a signature, and the arrows in the diagram are inclusion maps between these signatures. An object also comes with a set of sentences over its signature, which expresses the object's behaviour. For example, the ith button object has the sentences

$$[\text{press}_i]\text{lit}_i \quad \text{and} \quad [\text{cancel}_i]\neg\text{lit}_i$$

which, as already noted, means that the ith light is switched on and off by the press and cancel actions.

The lift-position object has the sentences

$$\begin{array}{ll} \text{floor}_i \rightarrow [\text{up}]\text{floor}_{i+1} & \text{for each } i = 1, \ldots, n-1 \\ \text{floor}_i \rightarrow [\text{down}]\text{floor}_{i-1} & \text{for each } i = 2, \ldots, n \\ \text{floor}_i \leftrightarrow \bigwedge_{j \neq i} \neg \text{floor}_j & \text{for each } i = 1, \ldots, n \end{array}$$

which respectively express the effects of the actions up and down and the fact that the lift is always at precisely one floor.

The locality conditions which structuring provides are axioms which express the fact that the actions of an object can only affect the values of the propositions also declared in that object. For example, for each $1 \leqslant i, j \leqslant n$ there is the axiom

$$\text{floor}_i \leftrightarrow [\text{press}_j]\text{floor}_i$$

which says that press_j leaves floor_i unchanged. That is because press_j can only affect the value of lit_j; its sole effect is to switch on the light.

As well as enhancing readability and providing the locality constraints as described, structuring specifications also provides a way of resolving conflicts between sentences, such as the conflict between sentences 5 and 4 of the lift. As noted, the first of these asserts that the button's light comes on when the button is pressed. However, we know that this is only the light's usual behaviour, because there are circumstances when the light does not come on when the button is pressed. We have already noted one such circumstance, which is when the lift is already at the relevant floor. The fact that the light does not illuminate in that case is a consequence of sentence 4. It says that the ith light is *never* on when the lift is at the ith floor. Intuitively, these two sentences conflict[2]. Sentence 5 says that the ith light *will* come on when the ith button is pressed, even if we are already at the ith floor, while 4 says it will not.

The conflict between 4 and 5 can be resolved by the lift structure by appealing to the *specificity principle*. It states that

> *Defaults about a specific class of objects override those about a more general class when there is conflict.*

This principle is well-known in artificial intelligence [2, 15]. It applies in this case because statement 4 is about lifts (i.e. it is in the lift object), while statement 5 is about buttons (it is in the button object). The structure of the lift specification shows that the lift object incorporates the button object. Therefore, the class of lifts is more specific than that of buttons. The specificity principle says then that statements about lifts override those about buttons, so 4 overrides 5.

In this way the specificity principle tells us that the relation of 'overriding' between sentences is given by the structure of the specification. That is to say, we may derive from it an ordering of the axioms which shows which ones may override which other ones in the case of a conflict. We call such orderings of axioms *ordered theory presentations*. To obtain the ordered theory presentation for the lift, we replace each object in the structure diagram above by the axioms which come with it, and reverse the direction of the arrows: see figure 1. (We have left out some axioms not relevant to this discussion.) We reversed the arrows so that we can read them as 'dominates' or 'overrides'. Thus, they happen to go in the reverse direction to the arrows of the structure diagram, but this is merely a matter of convention.

An ordered theory presentation, such as this one, is a finite set of sentences equipped with a partial order. Sentences lower in the ordering are to be treated as having greater weight or priority. They override higher sentences when there is a conflict. The exact nature of this overriding is the subject of the next section. But to prime our intuitions, notice that it is not simply the case that if two sentences conflict, we can ignore the upper one; some 'instances' of it may still be needed. For example, although sentence 4 overrides 5 of the lift, we still need sentence 5 for those cases that it is not overridden. This feature of ordered theory presentations is expanded upon in section 4.

Not all axioms express behaviour which may be overridden. For example, we may wish to keep locality axioms inviolable. This would be prudent, for if we override such axioms we may lose our intuitive understanding of the specification. Therefore, a specification seen from a logical point of view must be a pair $\langle \Delta, ? \rangle$ consisting of an ordinary theory presentation Δ (the inviolable axioms) and an ordered theory presentation ? (the defaults). We argue that these ordered theory presentations are 'first-class' logical entities. Their meaning is the subject of section 4.

[2] To obtain a formal contradiction it is necessary to invoke the locality axiom mentioned earlier. Take any M satisfying the axioms. By virtue of the third axiom in lift-position, we have that $M(\cdot) \Vdash \text{floor}_k$ for some k. (The symbol $\cdot$ is the empty sequence of action symbols.) Then, since $M(\cdot) \Vdash \text{floor}_k \leftrightarrow [\text{press}_k]\text{floor}_k$ (since that is an instance of the locality axiom), we obtain $M(\cdot) \Vdash [\text{press}_k]\text{floor}_k$ and so $M(\text{press}_k) \Vdash \text{floor}_k$. Therefore, since $\text{floor}_k \rightarrow \neg\text{lit}_k$ is an axiom, $M(\text{press}_k) \Vdash \neg\text{lit}_k$. But $[\text{press}_k]\text{lit}_k$ is an axiom, so $M(\cdot) \Vdash [\text{press}_k]\text{lit}_k$ and so $M(\text{press}_k) \Vdash \text{lit}_k$, a contradiction.

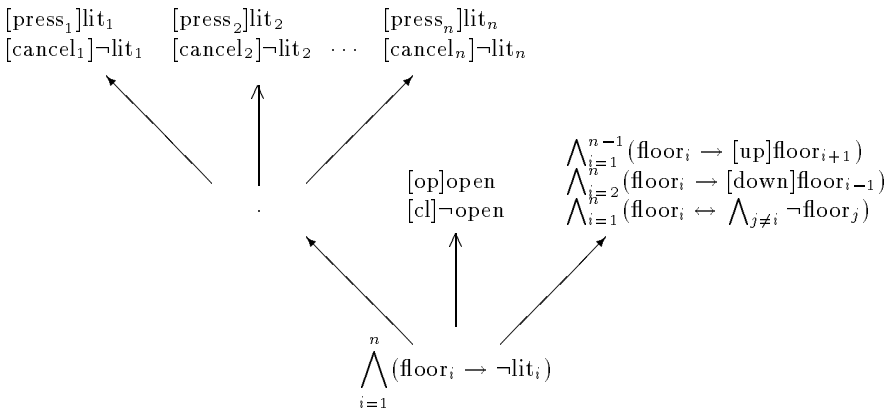


Figure 1: Part of the OTP for the lift

4 Ordered theory presentations

The concept of ordered theory presentation was first introduced in [12], under the name 'structured theory'. Since then the definitions and notations for OTPs have been improved and new results have been obtained. See also [11]. The present paper applies the cumulation of this work to the problem at hand, namely, the representation of defaults and revisions in specifications.

Let $\mathcal{M}$ be the set of interpretations as described in section 2 of a fixed signature L, and let $\Vdash$ be the satisfaction relation also described there.

To define the meaning of an ordered presentation $?$ we need to define an ordering $\sqsubseteq^{\Gamma}$ on interpretations in $\mathcal{M}$ which measures the extent to which interpretations satisfy the ordered presentation $?$. This relies on a set of orderings $\sqsubseteq_{\phi}$, one for each sentence ϕ of the language. To define $\sqsubseteq_{\phi}$, it is necessary to define natural entailment, which is written $\models\!\!\!\sim$. This definition in turn relies on the notion of the *monotonicities* of a sentence. Lest the reader be daunted by these nestings of definitions, we repeat the list here. We define, in order,

1. *Monotonicities* of a sentence ϕ, written $\langle \phi^{+}, \phi^{-} \rangle$.
2. *Natural entailment*, a relation $\models\!\!\!\sim$ between sentences, being a sub-relation of ordinary entailment $\models$.
3. For each sentence ϕ, a reflexive and transitive order $\sqsubseteq_{\phi}$ on the interpretations of the language; as will be seen, this order grades interpretations according to how nearly they satisfy ϕ.
4. An ordering $\sqsubseteq^{\Gamma}$ on the interpretations of the language; it grades interpretations according to how well they satisfy $?$.

When all these have been defined, we stipulate that the models of a specification $\langle \Delta, ? \rangle$ are the $\sqsubseteq^{\Gamma}$-maximal elements in the set of interpretations which satisfy Δ.

The motivation for this series of definitions may be seen in terms of the discussion of 'partial overriding' given in section 4. We note there that one of the criteria for the semantics of ordered theory presentations was that higher sentences be accepted at least in part, even when they are overridden by lower sentences. In other words, we want as much of the higher sentences as we can have, given that we must have the lower sentences and must avoid contradictions. This is the motivation for the ordering of item 3 in the above list; it is an ordering which allows us to chose models which *nearly* satisfy a particular sentence, even if we have already limited our choice to models which do not quite satisfy it *fully*. It is harder to motivate items 1 and 2 of the above list in intuitive terms, except to the extent that they are needed in order to arrive at the definition of the orderings $\sqsubseteq_{\phi}$. Item 4 may be seen in the same way as the orderings of item 3, except that it works at the level of entire theory presentations instead of individual sentences. Its definition is in terms of the subsidiary orderings, but it takes account of the ordering of the sentences in $?$ as well.

For the definition of the monotonicities of ϕ, we need the following notation. If M is an interpretation of L, σ a trace of L and p a propositional symbol in L, then $M(\sigma)^{[p \mapsto t]}$ is an interpretation identical with $M(\sigma)$ except possibly that it assigns **true** to p. (If $M(\sigma)$ already assigns **true** to p then $M(\sigma)^{[p \mapsto t]}$ is simply $M(\sigma)$.) $M(\sigma)^{[p \mapsto f]}$ is defined analogously.

Definition 1 Let ϕ be a sentence *other than* $-$ and

p any propositional symbol.

1. ϕ is *monotonic in* p if for each M, σ: $M(\sigma) \Vdash \phi$ implies that $M(\sigma)^{[p \mapsto t]} \Vdash \phi$.
2. ϕ is *anti-monotonic in* p if $M(\sigma) \Vdash \phi$ implies that $M(\sigma)^{[p \mapsto f]} \Vdash \phi$.
3. ϕ^+ and ϕ^- are the sets of symbols in which ϕ is monotonic and anti-monotonic respectively.

The case that $\phi = -$ is handled separately; we define $-^+ = -^- = \varnothing$.

Thus, ϕ is monotonic in p if "increasing" the truth value of p in a model of ϕ preserves satisfaction of ϕ. Similarly, ϕ is anti-monotonic in p if decreasing the truth value so preserves satisfaction.

Having defined monotonicities, we turn to point 2 of the four-point plan mentioned above, i.e. the definition of natural entailment. Let ϕ and ψ be sentences of L.

Definition 2 ϕ *naturally entails* ψ, written $\phi \mathrel{\dot{\models}} \psi$, if

1. $\phi \models \psi$, and
2. $\phi^+ \subseteq \psi^+$ and $\phi^- \subseteq \psi^-$

Natural entailment is a sub-relation of ordinary entailment; in addition to ordinary entailment we require that the monotonicities of the premise be preserved by the conclusion. This definition is really the core of the work described in this section, for the natural consequences of a sentence are its 'components' which may be individually accepted or rejected in the overall theory. In Brass/Ryan/Lipeck [1] we discuss other ways of describing such instances.

Proposition 3 $\mathrel{\dot{\models}}$ is reflexive and transitive.

The proofs of this proposition and others in this section which are given without proofs may be found in [12].

Definition 4 $M \sqsubseteq_\phi N$, if for each ψ,

$$\phi \mathrel{\dot{\models}} \psi \Rightarrow (M \Vdash \psi \Rightarrow N \Vdash \psi)$$

We can show that $\sqsubseteq_\phi$ has precisely the mathematical behaviour we want:

Proposition 5

1. $\sqsubseteq_\phi$ is a pre-order, that is to say, it is reflexive and transitive.
2. If $\phi \neq -$, the maximal elements of $\sqsubseteq_\phi$ (which are in fact maxi*mum*) are just the models of ϕ.

The proofs of these assertions, together with many examples of $\sqsubseteq_\phi$ for various sentences ϕ, can be found in [12].

We have defined, for each sentence ϕ, an ordering on interpretations $\sqsubseteq_\phi$ which measures the extent to which interpretations satisfy ϕ. If M satisfies ϕ to the fullest extent (that is, if it simply satisfies it) then M is $\sqsubseteq_\phi$-maximum. If M does not fully satisfy ϕ then it may satisfy it to a greater, lesser, equal or incomparable extent than some N which perhaps also fails fully to satisfy ϕ. As stated, examples of this ordering on models for various sentences ϕ can be found in [12].

All that remains of our four-point plan at the beginning of this section is to define the global ordering $\sqsubseteq^\Gamma$ for an ordered presentation ? in terms of the relations $\sqsubseteq_\phi$ for each $\phi \in ?$. The idea is that $M \sqsubseteq^\Gamma N$ means that if a sentence in ? makes the 'wrong' choice between M and N (i.e. if $M \not\sqsubseteq_\phi N$) then there is a sentence with greater priority which makes the 'right' choice.

Definition 6 $M \sqsubseteq^\Gamma N$ if for each $\phi \in ?$, $M \not\sqsubseteq_\phi N$ implies there exists $\psi \leqslant \phi$ in ? such that $M \sqsubset_\psi N$.

Proposition 7 $\sqsubseteq^\Gamma$ is a pre-order.

As already said, the models of a specification $\langle \Delta, ? \rangle$ are the $\sqsubseteq^\Gamma$-maximal elements in the set of interpretations which satisfy Δ.

Returning to the lift example, the question which should be answered is: what are the models of $\langle \Delta, ? \rangle$ when Δ is the locality axioms and ? is the structured theory presentation for the lift example, given at the end of section 3? More practically, we can ask: what are the interesting consequences of the specification (that is, what sentences hold in these maximal interpretations)?

As one would expect, all the axioms except the $[\mathrm{press}_i]\mathrm{lit}_i$ family are consequences of the specification. That is to say, it is the only axiom which is overridden by others or by locality axioms. Instead, the sentence

$$\neg \mathrm{floor}_i \wedge \mathrm{press}_i \rightarrow \mathrm{lit}_i,$$

which says that the button lights when pressed if the lift is at another floor, is also a consequence.

5 Revising specifications

In this section we briefly describe the application of this work to the topic of *specification revision*. Fuller details will be given in another paper. We use the word 'revision' to mean any change to a specification, which in general can remove some of its properties as well as adding others. This is not to be confused with 'refinement', which (like 'enrichment' or 'specialisation') is simply adding properties. The formal machinery required for revision is exactly the same as

the machinery for defaults which is described in the preceeding sections. Revisions arise inherently in the software process, and may also be taken as a design methodology. We consider each of these in turn.

The software development process. Revisions of requirements occur during the construction of a specification. Indeed, flushing them out is part of the motivation for formal specifications in the first place. During the requirements elicitation phase, the specifier typically brings to the attention of the customer undesirable consequences of the specification so far, and the customer requests revisions.

Design method. We may reuse specifications by revising them to fit the new context we have in mind. This has both small-scale and large-scale applications.

In the small, one can consider re-using components from a library of standard objects. If a component doesn't quite fit the application because it has unwanted properties, revise it with the desired properties.

In the large, whole specifications may be constructed in this way. For example, the recent Rover TV advertisement showed how the Metro motor-car was conceived as a Mini with certain properties added. These properties conflicted with the old ones, which means the *revision* is not merely a matter of *refinement* or *enrichment*. Thus, the Metro is specified by stipulating its *differences* from a Mini.

In practical terms one may envisage a software engineering environment (implemented on a computer) which allows one to explore a 'design space' by both small-scale and large-scale revisions of the type described here.

The obvious difference in the case of *specification revision* as against specifications with defaults is that the ordering in the resulting OTP comes not from the structure of the specification, as it did for defaults, but from the process by which the specification was obtained (the revision history). But in fact, these genealogies are not so different. One can think of a revision history as showing the structure (through refinement) of a component; for example, one can think of the structure diagram for the Metro

as a revision history or one can view the earlier objects as the components of which the Metro is made. On the other hand, a non-linear structure diagram such as that of the lift represents a revision history in more than one dimension. For example, the manufacturer's intention is that the button's light illuminates when the button is pressed. This is encoded in the button's specification. But the specification was revised for incorporation in the lift, since in that context it is to have the property that it does *not* light when pressed in certain circumstances; namely, when the lift is in a state in which the request made by the user by pressing the button is inappropriate. The revision is implemented via a complicated interface between the components which may not even be part of the specification—that is why an OTP is needed.

6 Objections

In the preceeding sections we have motivated the idea that a specification can be a rather 'loose' object, containing sentences which may be overridden by others. The reader may dislike this idea; this section is devoted to presenting objections to defaults and revision in specifications and, I hope, to allaying them.

The most common objection raised is that specifications are by nature exact, and it goes against the grain to introduce the slack which comes with defaults. I have much sympathy with this view, but I believe that the benefits gained outweigh the disadvantages. Among the benefits are the ability to represent default behaviour when it really is a characteristic of the object being specified; the ability to explore a design space; the *improvement* in modularisation which can be obtained (see below); and freedom from the chore of filling in every little detail, instead being able to allow conflicts to resolve automatically. Furthermore, from a methodological point of view, we narrow the gap between the informal requirements and the specification; without, I believe, the price of widening the gap on the other side, between specification and code. This is because the specifier has an improved medium for expressing the intuitions and intentions behind his or her specifications.

The improvement in modularisation referred to above can be seen by considering the effect of coding in the exception to sentence 5 of the lift specification. Sentence 5 expresses the fact that the buttons light when pressed, and is an axiom of the button object. The exception noted is when the lift is already at the relevant floor, so taking account of this the axiom would become:

$$\neg \mathrm{floor}_i \rightarrow [\mathrm{press}_i]\mathrm{lit}_i.$$

But this cannot now be an axiom of the button object after all, but must be an axiom of the complete lift system. This is because the vocabulary it uses is not available in the button signature. Thus the motivation for structuring (that is, dividing the specification into constituent objects and axiomatising them individually) in the first place is foiled: every axiom has to be part of the biggest object in order to list all the exceptions.

It might be objected that if some axioms are allowed to override others, we may quickly get into a mess in which we do not know which axioms are being affected by which others. To counter this objection, it should be possible to check at any stage whether a certain axiom expressing a default is being overridden or not, by checking whether it is a consequence of the specification. And again, the *advantage* is that one can explore the design space by changing the order around until the desired effect is achieved. This gives great flexibility to the specifier. Of course, the ability to do these things assumes a sophisticated interactive software environment which supports OTPs; such a thing is yet to be developed.

7 Comparison with other default logics

The idea of using default information in specifications has been motivated in sections 1 and 3. However, there are other default systems which might also be considered as candidates in specification theory. A full investigation is given in [11]; here we summarise our findings.

We have analysed several default systems, and found them wanting from our point of view. For example, Reiter's 'Default Logic' [10] represents defaults as rules of inference which interfere with the underlying logic, and cannot cope with the modalities of MAL. Conflicting defaults are handled by coding their precedence as consistency checks, as the specificity principle cannot be used directly. Also, Reiter's system enjoys insufficiently many of the formal properties of [7]. McCarthy's 'Circumscription' [6, 8], which has better formal properties of the kind described in [7], fares better from our point of view, representing defaults as ordinary first order sentences. But it is still unclear how modalities should be handled, and the resolution of conflicts between conflicting defaults has to be coded up explicitly, this time by manipulating relationships between abnormality predicates.

References

[1] S. Brass, M. Ryan, and U. Lipeck. Hierarchical defaults in specification. To appear, 1992?

[2] D. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *Proc. Third National Conference on Artificial Intelligence*, pages 104–108, 1983.

[3] J. Fiadeiro and T. Maibaum. Describing, structuring and implementing objects. In *Proc. REX Workshop on Foundations of Object-Oriented Languages.* Springer-Verlag, 1991.

[4] J. Fiadeiro and T. Maibaum. Towards object calculi. Technical report, Deprtment of Computing, Imperial College, London, 1992.

[5] R. Goldblatt. *Logics of Time and Computation.* CSLI Lecture Notes, 1987.

[6] V. Lifschitz. Computing circumscription. In *Ninth International Joint Conference on Artificial Intelligence*, pages 121–127, 1985.

[7] D. Makinson. General patterns in non-monotonic reasoning. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence.* Oxford University Press, 1992.

[8] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[9] J. McCarthy. Applications of circumscription to formalising common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[10] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[11] M. Ryan. *Ordered Presentations of Theories: Default Reasoning and Belief Revision.* PhD thesis, Department of Computing, Imperial College, 1992. Copies available from author.

[12] M. D. Ryan. Defaults and revision in structured theories. In *Proc. Sixth IEEE Symposium on Logic in Computer Science (LICS)*, pages 362–373, 1991.

[13] M. D. Ryan. Representing defaults as sentences with reduced priority. In B. Nebel and W. Swartout, editors, *Proc. Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92).* Morgan Kaufmann, 1992.

[14] M. D. Ryan, J. Fiadeiro, and T. Maibaum. Sharing actions and attributes in modal action logic. In T. Ito and A. Meyer, editors, *Theoretical Aspects of Computer Software*, pages 569–593. Lecture Notes in Computer Science 526, Springer Verlag, 1991.

[15] D. Touretzky. Implicit ordering of defaults in inheritance systems. In *Proc. Fifth National Conference on Artificial Intelligence*, pages 332–325, 1984.